

Projet ACTES IA

Traitements IA

Documentation technico-fonctionnelle

11 janvier 2021



Table des matières

[1. Prérequis techniques et installation du projet 3](#_Toc61260772)

[1.1. Versions des librairies 3](#_Toc61260773)

[1.2. Import et installation du projet de Gitlab 3](#_Toc61260774)

[2. Documentation technique 3](#_Toc61260775)

[3. Description des traitements IA 4](#_Toc61260776)

[3.1. Classification de la nature et matières des actes 4](#_Toc61260777)

[3.2. Recherche d’entités dans l’objet et le corps de l’acte 5](#_Toc61260778)

[3.3. Classification de la transmissibilité d’un acte 7](#_Toc61260779)

# Prérequis techniques et installation du projet

A date, les traitements d’intelligence artificielle sont exécutables manuellement. Leur automatisation n’est pas dans le périmètre de la présente expérimentation. Dans cette section, nous décrivons les librairies utilisées et le processus d’installation du projet dans un nouvel environnement.

## Versions des librairies

L’ensemble des traitements est réalisé en **Python 3.7**. Les bibliothèques suivantes sont nécessaires à la bonne exécution du pipeline.

|  |  |  |
| --- | --- | --- |
| **Bibliotèque** | **Version** | **Fonction** |
| **flair** | 0.6.1 | Bibliothèque de traitements NLP |
| **fuzzywuzzy** | 0.18.0 | Bibliothèque pour la comparaison de texte |
| **lime** | 0.2.0.1 | Bibliothèque pour l’explicabilité des modèles |
| **matplotlib** | 3.3.1 | Bibliothèque pour la visualisation des données |
| **nltk** | 3.4.5 | Bibliothèque de pretraitement du texte |
| **numpy** | 1.19.1 | Bibliothèque scientifique de calculs |
| **pandas** | 1.0.1 | Bibliothèque de gestion et transformation de DataFrames |
| **PyYAML** | 5.3.1 | Bibliothèque pour la gestion des fichiers YAML |
| **scikit\_learn** | 0.24.0 | Bibliothèque pour les modèles de Machine Learning |
| **seaborn** | 0.10.0 | Bibliothèque d’affichage de graphiques |
| **sqlalchemy** | 1.3.16 | Bibliothèque pour requêter et écrire dans des bases de données SQL |
| **spacy** | 2.2.4 | Bibliothèque de traitements NLP |
| **tqdm** | 4.50.2 | Bibliothèque permettant l’affichage des barres de progression |
| **Unidecode** | 1.1.1 | Bibliothèque permettant l’encodage et le décodage des caractères |
| **wordcloud** | 1.7.0 | Bibliothèque pour l’affichage de wordcloud |
| **xgboost** | 1.1.0 | Bibliothèque pour les modèles XGBoost |

## Import et installation du projet de Gitlab

* Le projet est disponible sur le Gitlab de Starclay à l’adresse <https://gitlab.starclay.fr/ms9/dgcl/actes-ia-data>
* L’accès à ce dépôt de code est donné par Joseph Assu Ondo (jassuondo@starclay.fr), le pilote du projet.
* Pour importer le projet sur un nouvel environnement, dans un dossier taper la commande git clone [git@gitlab.starclay.fr:ms9/dgcl/actes-ia-data.git](mailto:git@gitlab.starclay.fr:ms9/dgcl/actes-ia-data.git) après avoir configuré votre clé ssh sur Gitlab

# Documentation technique

Toute la documentation technique du projet est disponible sur Gitlab. Ci-après les éléments que vous y trouverez.

|  |  |  |
| --- | --- | --- |
| **Document** | **Description** | **Lien Gitlab** |
| ReadME.md | Document principal rappelant le fonctionnement du dépôt Gitlab et faisant référence aux autres documents | <https://gitlab.starclay.fr/ms9/dgcl/actes-ia-data/blob/develop/README.md> |
| docs/CONTRIBUTING.md | Décrit le processus pour contribuer (ajout de code) sur le projet | <https://gitlab.starclay.fr/ms9/dgcl/actes-ia-data/blob/develop/docs/CONTRIBUTING.md> |
| docs/README.md | Description du processus de génération de la documentation du code | <https://gitlab.starclay.fr/ms9/dgcl/actes-ia-data/blob/develop/docs/README.md> |
| docs/\_build/pdf/Python.pdf | Contient la description des éléments implémentés dans le code (fonctions, classes, modules) | <https://gitlab.starclay.fr/ms9/dgcl/actes-ia-data/blob/develop/docs/_build/pdf/Documentation%20Actes%20IA.pdf> |

# Description des traitements IA

## Classification de la nature et matières des actes

**Problématique**

L’objectif de ce chantier est de pouvoir contrôler automatiquement l’exactitude des métadonnées suivantes saisies par les collectivités : la nature et les matières de niveau 1 et 2. Ce traitement permettra aux préfectures de vérifier plus rapidement si l’acte correspond bien à ce qui a été mentionné dans les métadonnées.

Pour ce faire, nous avons implémenté des algorithmes de classification supervisée en nous basant uniquement sur le texte de l’acte.

**Méthode et résultats obtenus**

Le texte a d’abord été vectorisé à l’aide de différentes méthodes (sac de mots et tf-idf) ainsi que différents algorithmes (SVM, Random Forest, XGBoost) et nous avons obtenu les meilleurs résultats pour le sac de mots avec un modèle XGBoost. Les meilleurs paramètres sont trouvés parmi une grille de valeur pour obtenir les performances les plus hautes.

Les modèles ont été entrainés dès qu’il y avait de nouvelles annotations disponibles et les résultats présentés ci-dessous représentent l’itération la plus récente.

|  |  |  |  |
| --- | --- | --- | --- |
|  | **Précision** | **Rappel** | **Support** |
| Nature | 87.61% | 86.61% | 1706 |
| Matière 1 | 86.9% | 86.84% | 1700 |
| Matière 2 – 1 | 58.88% | 60.71% | 239 |
| Matière 2 – 2 | 85% | 80% | 41 |
| Matière 2 – 3 | 65.42% | 64% | 320 |
| Matière 2 – 4 | 88.81% | 90% | 245 |
| Matière 2 – 5 | 68.22% | 68.85% | 272 |
| Matière 2 – 6 | 88.24% | 88.24% | 78 |
| Matière 2 – 7 | 77.88% | 82.08% | 472 |

**Pistes d’amélioration**

La plupart des modèles présentés ci-dessus ont des résultats pouvant être exploitables (précision et rappel supérieurs à 80%) mais ce n’est pas le cas pour tous les modèles. Certaines pistes d’améliorations sont envisageables pour cette partie :

* Augmentation du nombre d’annotations : la majorité des types de métadonnées contient moins de 200 annotations, ce qui signifie que pour un type donnée, le nombre d’annotations par classe est très faible. L’augmentation des annotations permettrait un meilleur apprentissage.
* Uniformisation des annotations : certains actes pouvaient être plus subjectifs à annoter et donc l’annotation pouvait dépendre de l’annotateur. Une uniformisation des annotations permettrait d’avoir des ensembles plus cohérents pour chaque classe.

**Description des principaux scripts et modules Python**

Tous les scripts Python sont contenus dans le répertoire **ML/scripts** et les modules dans **ML/classifier**

* **classifier/classifier\_preprocessing.py**

Description : contient les fonctions permettant le prétraitement du texte pour la classification. Ce traitement se fait lors de l’import des annotations mais peut également se faire lors de l’entrainement du modèle si nécessaire.

* **classifier/classifier\_model.py**

Description :contient la classe Classifier permettant l’entrainement d’un modèle de classification

* **scripts/train\_classifier.py**

Description : script permettant d’entrainer un modèle sur un type de métadonnée (nature, matière 1, matière 2, transmissibilité)

Arguments :

-d <chemin\_dataset> : Chemin vers un fichier pickle contenant au minimum la colonne correspond au type de métadonnée ainsi que la colonne « texte » ou « preprocessed\_text »

-t <type\_label > : Type de métadonnée pour l’entrainement (nature, matiere\_1, matiere\_2\_1, …)

-c <chemin\_vers\_configuration> : Chemin vers le fichier de configuration pour les paramètres du modèle (**config/ConfigClassifier.yaml**)

-s <chemin\_model\_entrainé> : Chemin vers un dossier où sauvegarder le nouveau modèle

## Recherche d’entités dans l’objet et le corps de l’acte

**Problématique**

Pour continuer la vérification des métadonnées, il est également nécessaire de contrôler l’objet saisi par les collectivités. Cet objectif nécessite de détecter les entités dans le corps de l’acte (conformément à une ontologie coconstruite avec la DGCL) et de les confronter avec les éléments présents dans l’objet saisi par les collectivités.

**Méthode et résultats obtenus**

Les éléments du corps et de l’objet de l’acte sont détectés à l’aide d’un algorithme de NER (**named entity recognition**). Deux librairies sont testées pour retrouver les entités dans le texte : SpaCy et Flair. L’entrainement de ces modèles donne des résultats similaires mais chaque algorithme à ses points forts pour certaines entités. Dans le cadre de ce projet, on ne garde que les résultats de SpaCy pour sa rapidité d’entrainement et d’utilisation.

Les métriques utilisées sont la précision et le rappel partiel, c’est-à-dire qu’une entité prédite est prise en compte à partir du moment où elle à au moins un mot commun avec le label. Cela permet de prendre en compte certaines prédictions même si elles ne sont pas exactes.

Les résultats pour les entités retrouvées dans l’objet de l’acte sont présentés dans le tableau suivant :

|  |  |  |  |
| --- | --- | --- | --- |
|  | Précision partielle | Rappel partiel | Support |
| Actes Individuels | 68.4% | 59.6% | 949 |
| Actes réglementaires | 39.7% | 33.8% | 302 |
| Contrats et conventions | 51.8% | 35.8% | 374 |
| Délibérations | 45.8% | 38.7% | 1958 |

Pour les entités dans le corps du texte, le nombre d’annotations n’étant pas suffisant pour correctement apprendre les modèles, les résultats ne peuvent pas être exploités. Par conséquent, la vérification de l’objet de l’acte avec le corps de l’acte n’est pas possible avec le nombre d’annotations actuel.

**Description des principaux scripts et modules Python**

Tous les scripts Python sont contenus dans le répertoire **ML/scripts** et les modules dans **ML/ner**

* **ner/evaluator.py**

Description : contient la classe Evaluator permettant d’évaluer les performances des modèles NER

* **ner/ner\_model.py**

Description : contient les classes FlairModel et SpacyModel permettant l’entrainement d’un modèle NER

* **ner/ner\_preprocessing.py**

Description : contient les fonctions permettant le prétraitement du texte pour la classification. Ce traitement se fait lors de l’import des annotations mais peut également se faire lors de l’entrainement du modèle si nécessaire.

* **scripts/train\_ner.py**

Description : script permettant d’entrainer un modèle sur un type de métadonnée (objet et corps)

Arguments :

-d <chemin\_dataset> : Chemin vers un fichier pickle contenant au minimum les colonnes « texte » / « corps » ou « objetacte » / « metaObjet »

-t <type\_label > : Type de métadonnée pour l’entrainement (corps, objet)

-c <chemin\_vers\_configuration> : Chemin vers le fichier de configuration pour les paramètres du modèle (**config/ConfigNER.yaml**)

-s <chemin\_model\_entrainé> : Chemin vers un dossier où sauvegarder le nouveau modèle

## Classification de la transmissibilité d’un acte

**Problématique**

Il est nécessaire de trier les actes selon leur transmissibilité (oui ou non) pour le contrôle de légalité. L’algorithme pourra utiliser un ensemble de paramètres tels que la nature et la matière de l’acte ainsi que son contenu. Les données disponibles ne sont pas reparties uniformément et seul une petite partie des données concerne les actes non transmissibles.

**Méthode et résultats obtenus**

Le manque d’uniformité dans les données a été pallié par les annotations sur la transmissibilité. Pour pouvoir distinguer un acte transmissible d’un acte non transmissible, plusieurs approches ont été testées :

* Utilisation d’un ensemble de règles sur les métadonnées de l’acte comme la nature et la matière ainsi qu’une vérification de mots clefs à l’intérieur du texte et de l’objet.
* Utilisation du corps et de l’objet de l’acte avec une classification binaire
* Utilisation du corps et de l’objet de l’acte ainsi que les métadonnées de l’acte. Un modèle multimodal est ensuite entrainé sur ces données.

L’approche la plus efficace a été celle qui utilise un ensemble de règles pour permettre la classification de l’acte bien qu’elle ne couvre pas la totalité du périmètre. A date, environ 33% des actes annotés sont dans le périmètre des règles implémentées. Les approches de classification binaire ou multimodale n’ont pas atteint un niveau de performance suffisant du fait du déséquilibre des actes non transmissibles et du volume d’annotation faible sur la transmissibilité (moins de 2000 actes au moment des modélisations).

**Pistes d’amélioration**

Certaines annotations concernant les actes non transmissibles n’ont pas pu être terminés et il y a eu ainsi très peu d’actes non transmissibles exploitables, ce qui a pu rendre l’approche multimodale obsolète. L’augmentation du volume d’annotation pourrait permettre de rendre la classification multimodale plus performante

**Description des principaux scripts Python**

Tous les scripts Python sont contenus dans le répertoire **ML/scripts**

* **scripts/train\_classifier.py**

Description : script permettant d’entrainer un modèle sur la transmissibilité

Arguments :

-d <chemin\_dataset> : Chemin vers un fichier pickle contenant au minimum la colonne correspond au type de métadonnée ainsi que la colonne « texte » ou « preprocessed\_text »

-t <type\_label > : Type de métadonnée pour l’entrainement (« isTransmissible » dans ce cas)

-c <chemin\_vers\_configuration> : Chemin vers le fichier de configuration pour les paramètres du modèle (**config/ConfigClassifier.yaml**)

-s <chemin\_model\_entrainé> : Chemin vers un dossier où sauvegarder le nouveau modèle

* **scripts/regles\_transmissibilite.py**

Description : script permettant de déterminer la transmissibilité d’un acte à partir d’une suite de règles concernant la nature, la matière et l’objet/le corps de l’acte

Argument :

-d <chemin\_dataset> : Chemin vers un fichier pickle contenant les colonnes « objetacte », « texte », « nature », « matière 1 », « matière 2 »