PROJET ING2

[ETU22] Disambiguation of inventor datasets

Bacarie TCHABO
Iyad BEN MOSBAH
Salim OUESLATI
Charles LIU

Responsable : François Maublanc
Encadrant : Pierre Andry

2024-2025

Contents

1 Introduction

Contexte général

Dans un contexte où l'innovation technologique est un moteur majeur de croissance économique, l'étude des données de brevets constitue un levier stratégique pour analyser les dynamiques de l'innovation, les trajectoires des inventeurs et les stratégies de propriété intellectuelle. Les bases de données publiques de brevets, notamment celles de l'Office Européen des Brevets (EPO), fournissent une quantité massive d'informations sur les inventions et les acteurs impliqués. Cependant, ces données présentent une limite de taille : les inventeurs n'y sont pas identifiés par un identifiant unique, ce qui empêche de les suivre de manière fiable.

En effet, un même inventeur peut apparaître dans la base sous différentes variantes de nom (J. Smith, John Smith, J. Smyth), selon les conventions de dépôt, les erreurs de saisie ou les traductions. À l'inverse, plusieurs inventeurs différents peuvent partager le même nom. Ce phénomène rend difficile toute analyse statistique ou qualitative fiable sur les inventeurs, et constitue le cœur de la problématique de la *désambigusation des inventeurs*.

Problématique

Dans ce contexte, la question que nous nous posons est la suivante :

Comment identifier de manière fiable un inventeur à partir d'un ensemble d'informations partielles (nom, prénom, pays, année...), et retrouver l'ensemble des brevets qu'il a déposés dans la base ?

Il s'agit ici de mettre en œuvre une solution qui permette d'éviter deux types d'erreurs :

- **Les faux positifs** : regrouper à tort des individus distincts sous un même identifiant.
- **Les faux négatifs** : scinder un même individu en plusieurs identifiants.

Objectifs du projet

L'objectif du projet est de concevoir un outil informatique qui, à partir des informations saisies par un utilisateur (nom, prénom de l'inventeur), exploite un modèle de désambigusation existant pour :

- retrouver l'identifiant unique de l'inventeur ;
- afficher la liste des brevets associés à cet inventeur dans la base.

Cet outil s'appuie sur une base relationnelle contenant plusieurs millions d'enregistrements, fournie sous SQL Server, et organisée autour de tables liées aux inventeurs, aux brevets, aux domaines technologiques et aux descriptions textuelles.

Démarche générale

Notre approche repose sur les étapes suivantes :

1. Analyse et indexation de la base de données fournie.
2. Intégration d'un modèle de désambigusation existant (notamment celui du projet PatentsView USPTO).

3. Conception d'un outil de recherche (script ou interface utilisateur).
4. Evaluation des résultats obtenus, test sur cas réels, et discussion des limites.

Structure du rapport

Ce rapport est structuré comme suit :

- La première partie présente la base de données mise à disposition, ses caractéristiques, ainsi que les difficultés rencontrées lors de son exploration et de son traitement.
- La deuxième partie décrit l'algorithme de désambiguïsation utilisé (issu de PatentsView), ainsi que les adaptations nécessaires pour l'appliquer à notre propre base.
- La troisième partie présente l'interface graphique développée pour exploiter le modèle, ainsi que les choix techniques réalisés (notamment l'usage d'ElasticSearch).
- Enfin, la dernière partie propose une discussion critique sur les limites rencontrées et les pistes d'amélioration envisageables pour la suite du projet.

2 Analyse de la base de données

2.1 Structure des tables de données

La base de données mise à disposition dans le cadre du projet est structurée en six tables principales, contenant plusieurs millions de lignes chacune. Toutes ces tables sont reliées entre elles par une clé commune : `appln_id`, identifiant unique d'une demande de brevet. L'ensemble constitue une base relationnelle riche, permettant d'associer des informations techniques, des données d'inventeurs, de demandeurs, ainsi que des éléments textuels (titres et résumés).

Nous présentons ci-dessous une analyse détaillée de chaque table.

1. `patents` – Table principale des demandes de brevets

Cette table regroupe les informations administratives et chronologiques des demandes de brevets :

Champ	Description
`appln_id`	Identifiant unique de la demande de brevet (clé primaire)
`appln_auth`	Code de l'autorité de dépôt (EP pour EPO dans tous les cas ici)
`appln_kind`	Type de brevet : B1, A1, D, A2, etc.
`appln_filing_year`	Année de dépôt
`appln_filing_date`	Date exacte de dépôt
`earliest_filing_year` / `earliest_filing_date`	Informations sur le dépôt prioritaire
`internat_appln_id`	Identifiant de la demande internationale (s'il y a lieu)

Cette table constitue le noyau central de la base, utilisé pour toutes les jointures.

2. `patents_applicants` – Donneurs d'ordre (entreprises, institutions)

Champ	Description
`person_id`	Identifiant initial du demandeur
`person_name`	Nom du demandeur
`psn_id`	Identifiant unique harmonisé
`psn_name`	Nom du demandeur harmonisé
`psn_sector`	Secteur (ex. : COMPANY, UNIVERSITY, GOV, INDIVIDUAL)
`han_name`	Variante harmonisée du nom
`person_address`	Adresse du déposant
`person_ctry_code`	Code pays (ex. : FR, DE, US)

Ce tableau est utile pour enrichir les contextes des brevets (qui dépose, dans quel secteur, avec quelle localisation...).

3. `patents_inventors` – Liste des inventeurs

Champ	Description
`person_name`	Nom complet de l'inventeur (souvent avec variations)
`person_id`	Identifiant de l'inventeur
`psn_id`	Identifiant harmonisé de l'inventeur
`han_name / han_id`	Nom et identifiant harmonisés via algorithme
`invt_seq_nr`	Position de l'inventeur dans la liste
`person_ctry_code`	Pays de l'inventeur (souvent manquant ou partiel)

Exemple concret :

```
SELECT * FROM patents_inventors
WHERE person_name LIKE 'Dupont, Jean-Fabien%';
```

Ce type de requête renvoie plusieurs lignes avec des noms proches et des adresses différentes (ex. Kodak Industrie à Chalon-sur-Saône), tous liés à trois `psn_id` différents : 7483079, 7483080, 7483081.

Ces cas sont typiques de la problématique de désambiguïsation.

4. `patents_tech` – Informations technologiques

Champ	Description
`ipc_maingroup_symbol`	Code de classification IPC (ex. : H04L)
`techn_field`	Domaine technologique (ex. : Intelligence Artificielle)
`techn_sector`	Secteur plus large (ex. : Digital Communication)
`weight`	Score de pertinence entre le brevet et le domaine

Ces informations sont utiles pour évaluer la proximité technologique entre plusieurs brevets ou inventeurs.

5. `patents_title` – Titre des brevets

Champ	Description
`appln_title`	Titre du brevet
`appln_title_lg`	Langue du titre (ex. EN ou DE)

Ces données servent à présenter les résultats à l'utilisateur, mais aussi à identifier des mots-clés récurrents ou spécifiques.

6. `patents_abstracts` – Résumé des brevets

Champ	Description
`appln_abstract`	Résumé du brevet (parfois très long)
`appln_abstract_lg`	Langue du résumé

Ces informations sont optionnelles dans le cadre de notre projet, mais peuvent enrichir l'analyse sémantique ou servir dans des travaux de NLP (Natural Language Processing).

Relations entre les tables

```
patents (appln_id)
 > patents_inventors (appln_id, psn_id)
 > patents_applicants (appln_id, psn_id)
 > patents_tech (appln_id)
 > patents_title (appln_id)
 > patents_abstracts (appln_id)
```

Chaque table est liée au brevet principal par la clé `appln_id`*, et les tables* `inventors` *et* `applicants` *permettent une désambiguïsation via* `psn_id` *ou* `han_id`*.*

2.2 Le volume des données

L'un des premiers obstacles rencontrés lors de ce projet a été la taille extrêmement importante de la base de données. En effet, les six tables mises à notre disposition contiennent plusieurs millions de lignes, ce qui rend toute manipulation directe avec des outils classiques particulièrement difficile, voire impossible.

Fichiers impossibles à ouvrir avec des outils standards

Notre premier réflexe a été d'essayer de consulter les fichiers `.csv` avec des outils courants comme Excel ou un éditeur de texte. Très vite, nous nous sommes rendu compte que cela ne fonctionnait pas :

- Excel ne parvenait pas à ouvrir les fichiers les plus volumineux, ou alors seulement partiellement (limité à 1 048 576 lignes).
- Les éditeurs de texte (comme Notepad++ ou même VSCode) ralentissaient fortement à cause du poids des fichiers et des nombreuses colonnes.

Cette situation nous empêchait de faire une première exploration des données pour comprendre leur structure.

Tables particulièrement lourdes

Certaines tables se sont avérées plus problématiques que d'autres, notamment :

- `patents_inventors` : elle contient des millions d'entrées reliant chaque inventeur à un ou plusieurs brevets. Il n'est pas rare qu'un seul inventeur apparaisse des centaines de fois dans cette table.
- `patents_abstracts` : cette table comprend de longues chaînes de texte (résumés de brevets), ce qui augmente considérablement la charge mémoire à chaque requête.

Le simple fait de tenter une visualisation complète de ces tables provoquait des ralentissements importants, voire des blocages.

Difficulté à tester des hypothèses

Ce volume de données rendait également compliqué le développement :

- Chaque test d'une requête ou d'un filtrage prenait beaucoup de temps.
- Il était difficile d'avoir une idée rapide de la structure des données ou de détecter des doublons sans outils adaptés.

Solution mise en place : utilisation de SSMS

Pour faire face à ces contraintes, nous avons rapidement choisi de basculer vers un outil professionnel de gestion de base de données : **SQL Server Management Studio (SSMS)**.

Cet environnement permet :

- de charger et manipuler efficacement de grandes quantités de données,
- d'exécuter des requêtes complexes sur des tables volumineuses sans faire planter la machine,
- de visualiser les résultats de manière partielle, en limitant le nombre de lignes affichées.

Nous avons également optimiser nos requêtes pour interroger uniquement ce qui était nécessaire. Pour cela, nous avons utilisé :

- des clauses WHERE pour filtrer les résultats selon certains critères (nom, année, pays...),
- la commande TOP pour n'afficher que les premières lignes,
- des jointures ciblées entre deux tables au lieu de croiser plusieurs tables d'un coup.

```
1    SELECT TOP 2000* FROM dbo.patents
2    SELECT * FROM dbo.patents WHERE appln_filing_year=2002
```

Listing 1: Exemple en SQL

Bénéfices de cette approche

Cette décision de travailler avec SSMS et de structurer nos requêtes a été déterminante pour la suite du projet. Elle nous a permis de :

- prendre connaissance des données de manière progressive,
- tester nos idées de manière rapide et fiable,
- préparer efficacement le terrain pour intégrer le modèle de désambiguïsation.

En somme, face à la contrainte du volume, nous avons su adapter nos outils et nos méthodes, ce qui a considérablement fluidifié notre travail.

2.3 Lenteur de certaines requêtes

Même après avoir importé les données dans SQL Server Management Studio (SSMS), nous avons rapidement constaté que certaines requêtes mettaient beaucoup de temps à s'exécuter, voire ne renvoyaient pas de résultat du tout si elles mobilisaient plusieurs tables volumineuses en même temps.

Exemples de cas problématiques

- Requêtes avec des jointures entre plusieurs tables comme `patents_inventors`, `patents`, `patents_tech` et `patents_title`.
- Requêtes sans conditions précises, qui parcouraient l'intégralité des millions de lignes.
- Requêtes combinant plusieurs champs textuels longs (titres, abstracts, noms) avec des clauses `LIKE`, ce qui ralentit fortement l'exécution.

Ces lenteurs posaient problème pour le bon déroulement du projet car :

- elles ralentissaient notre phase d'exploration de la base,
- elles complexifiaient les tests pour évaluer les performances du modèle de désambiguïsation,
- elles rendaient plus difficile l'intégration dans un outil interactif (type interface utilisateur).

Analyse des causes

Nous avons identifié plusieurs raisons expliquant ces lenteurs :

- L'absence d'index sur certaines colonnes souvent utilisées dans nos filtres ou jointures (`psn_id`, `appln_id`, `person_name`, etc.).
- Une structure de requêtes pas encore bien optimisée au départ (utilisation excessive de `SELECT *`, absence de filtres).
- Le fait que certaines requêtes appelaient trop de lignes d'un coup, sans limite ni pagination.

Solutions mises en place

1. **Utilisation d'index**
   Nous avons créé des index sur les colonnes clés les plus utilisées :
   - `appln_id` : utilisé dans toutes les jointures entre tables,
   - `person_name` : utile pour les recherches utilisateur,
   - `appln_filing_year` : utile pour restreindre une période temporelle.

```
CREATE INDEX idx_person_name ON dbo.patents_inventors (person_name);
CREATE INDEX idx_appln_id ON dbo.patents (appln_id);
```

Listing 2: Création d'INDEX sur SQL Server Management

2. **Requêtes plus ciblées**
   Nous avons appris à éviter les requêtes générales de type `SELECT *`, en limitant le nombre de colonnes et de lignes retournées, et en ajoutant des clauses `WHERE` pertinentes.

Résultat obtenu

Grâce à ces ajustements :

- Le temps d'exécution moyen de nos requêtes a nettement diminué,
- Nous avons pu interagir avec la base en temps raisonnable,
- Les temps de chargement dans notre prototype d'interface sont devenus acceptables pour l'utilisateur.

2.4 Données mal encodées, incohérentes ou absentes

Au-delà du volume et des lenteurs, une autre difficulté importante s'est imposée très rapidement : la qualité et l'homogénéité des noms dans la base de données. De nombreuses incohérences rendaient difficile l'identification fiable des inventeurs, étape pourtant centrale pour notre projet de désambiguïsation.

Exemple concret

L'un des cas typiques rencontrés est celui de `Jean-Fabien Dupont`. Dans notre base, cet inventeur apparaît sous au moins trois variantes de nom dans la table `patents_inventors` :

- `DUPONT, Jean-Fabien Kodak Industrie`
- `Dupont, Jean-Fabien`
- `Dupont, Jean-Fabien, c/o Kodak Industrie, Dep-Brev`

Pourtant, chacune de ces lignes a été associée à un identifiant `psn_id` différent : 7483080, 7483079 et 7483081. Cela illustre les limites du système de désambiguïsation automatique déjà en place : des noms très proches, appartenant manifestement au même individu, sont interprétés comme trois inventeurs distincts.

Problèmes rencontrés

- Multiples variations d'un même nom, dues à la présence d'affiliations ou de formulations différentes (ordre prénom/nom, majuscules, etc.).
- Ajouts de mentions internes aux entreprises (ex. c/o Kodak Industrie, Dep-Brev) incluses dans le champ `person_name`.
- Absence complète du nom de l'inventeur dans certaines lignes, rendant toute désambiguïsation impossible faute d'information principale.

Limites des solutions simples

Nous avons essayé de corriger certaines incohérences à l'aide de fonctions SQL, mais ces solutions restent limitées :

- Le nettoyage avec `LOWER()`, `REPLACE()` ou `TRIM()` améliore la cohérence mais ne permet pas d'identifier avec certitude deux variantes comme appartenant à la même personne.
- Le champ `psn_name`, généré automatiquement par un système de normalisation, est utile pour harmoniser certaines écritures, mais il reste sensible à des différences minimes (présence d'une virgule, d'un nom d'entreprise, etc.).

Rôle essentiel de l'algorithme de désambiguïsation

Ces limites soulignent l'importance d'utiliser un algorithme dédié pour la désambiguïsation. Contrairement aux méthodes basées uniquement sur des requêtes SQL ou des transformations simples, l'algorithme prend en compte un ensemble de caractéristiques contextuelles (nom, co-inventeurs, intitulés de brevets, domaines technologiques...) pour estimer si deux enregistrements désignent la même personne.

Autrement dit, les solutions que nous avons explorées (nettoyage, harmonisation, filtrage) sont nécessaires mais insuffisantes. Elles permettent de réduire le bruit dans les données, mais c'est bien l'algorithme qui joue le rôle central pour regrouper efficacement les inventeurs similaires sous un identifiant unique.

Bénéfices des prétraitements appliqués

- Réduction des erreurs typographiques et des doublons évidents.
- Allègement de la charge de travail pour l'algorithme en limitant le nombre de cas incohérents à traiter.
- Amélioration globale de la précision attendue du modèle de désambiguïsation.

```
DELETE FROM dbo.patents_inventors
WHERE person_name IS NULL
   OR psn_name = '-NOT AVAILABLE-'
   OR psn_id IS NULL;
```

Listing 3: Requête pour supprimer les lignes non exploitables

Nous verrons dans la section suivante comment nous avons adapté l'algorithme de désambiguïsation de PatentsView pour travailler avec notre base SQL Server et quelles difficultés ont été rencontrées lors de sa mise en œuvre.

3 Adapation de l'algorithme de désambigusation

3.1 Préambule

L'algorithme (entièrement codé en python) utilisée lors de notre travail est issu du *inventor disambiguation workshop* de PatentsView de 2015.

Le principe détaillé de l'algorithme peut être obtenu ici. Le dépot Github de l'algorithme est accessible ici.

Dans la partie suivante, nous proposerons une explication succincte de la méthodologie de l'algorithme.

3.2 Fonctionnement de l'algorithme de désambigusation

1. Motivations
Comme expliqué lors de l'introduction, les données concernant les brevets, le déposant et les inventeurs ne suivent pas une nomenclature stricte. De ce fait, les bases de données de brevet présentent des ambiguïtés, ce qui rend difficile l'exploitation de ces dernières. Le but de l'algorithme présenté ci-après est de lever ces ambiguïtés en assignant un identifiant unique à chaque inventeur.

2. Étape 1 : Création de Canopées

La première étape du processus de désambiguïsation consiste à identifier les enregistrements qui pourraient représenter la même entité (dans notre cas, cela correspond à un inventeur). Ce processus utilise une série de règles pour regrouper les mentions en "canopées". Les résultats sont intégrés à travers les canopées une fois le processus de *clustering* terminé. Les règles de formation des canopées ont été développées empiriquement en testant de nombreuses approches différentes jusqu'à ce qu'un ensemble de règles divisant efficacement les mentions soit établi.

Pour les inventeur, le regroupement en canopées se fait via la règle suivante : les personnes possèdent le même nom et la même initial de prénom appartiennent à la même canopée.

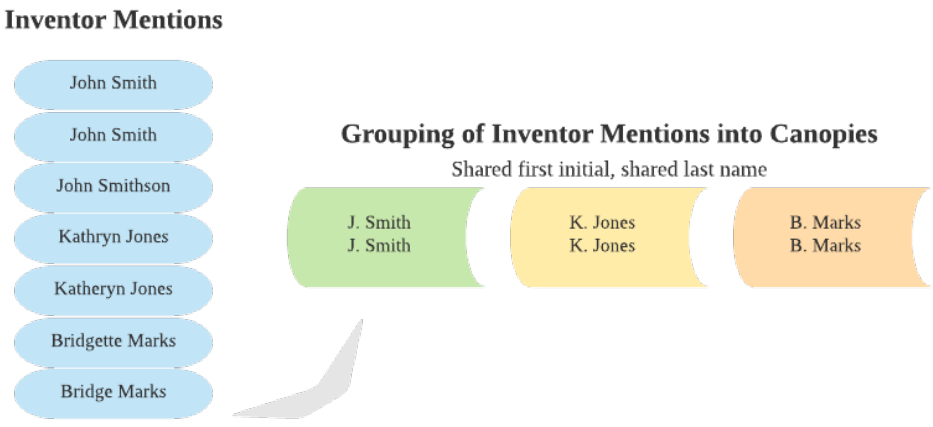


Figure 1: procédure de regroupement en canopées pour les inventeurs

3. Étape 2 : *Clustering*

Dans cette étape, on effectue un *clustering* dans chacune des canopées. A l'issue de ce *clustering*, les individus présent dans un même *cluster* sont considérés comme étant la même personne (la métrique de précision de l'algorithme donne la probabilité que les individus d'un même cluster soient la même personne).

Pour cela, on calcule un score de similarité entre les individus à partir de plusieurs caractéristiques (domaine technologique du brevet de l'individu, adresse de l'individu entre autres).

La méthode de désambigusation d'inventeur utilise un modèle linéaire qui détermine la similarité entre deux ensembles d'enregistrements. Chaque caractéristique est calculée comme une fonction linéaire de sa valeur et d'un terme de biais. Les scores résultants de chacune des caractéristiques sont additionnés pour produire un score final. Pour le calcul de la similarité de nom, nous utilisons une fonction name match score basée sur des règles. La fonction est conçue pour déterminer la probabilité que les noms d'un groupe de prénoms ou de deuxième prénoms avec le même nom de famille correspondent. La fonction prend en entrée une liste de prénoms ou de seconds prénoms et un nom de famille qui est commun à tous les noms du groupe. Les étapes suivantes sont les suivantes

- Nous vérifions le nombre de cas de pénalité. Si la liste des noms est vide, nous renvoyons une "pénalité d'absence de nom". De même, si la liste des noms est plus grande qu'une taille maximale définie, nous renvoyons une "pénalité de trop nombreux noms". Enfin, nous vérifions si le nom de famille correspond aux noms communs de ce groupe. Si ce n'est pas le cas, nous renvoyons une "pénalité de non-correspondance du nom de famille commun".
- Une fois ces vérifications de pénalité terminées, nous commençons à calculer les distances par paires entre les noms. Pour toutes les paires de noms, nous commençons par vérifier si les premiers caractères des deux chaînes correspondent. S'ils ne correspondent pas, nous augmentons la variable firstLetterMismatches de 1.
- Ensuite, nous exécutons notre fonction editDistance pour calculer la différence entre la paire de noms donnée et nous stockons cela dans une variable nameMismatches.
- La valeur firstLetterMismatches est ensuite multipliée par un intial_mismatch _weight et soustraite du score.
- La valeur nameMismatches est multipliée par un name_mismatch_weight, et cette valeur est également soustraite du score.
- Le score final est renvoyé par la fonction.

3.3 Adaptation de l'algorithme

Étant donnée que l'algorithme ne possède aucune documentation, nous avons du procéder à un travail de recherche et d'adaption dont nous présentont le resultat ci-après.

Dans un premier temps, nous avons essayé de faire fonctionner l'algorithme sur un échantillon de nos données. L'algorithme commence par les lignes de commande suivantes :

```
1    python -m pv.disambiguation.inventor.build_assignee_features_sql
2    python -m pv.disambiguation.inventor.build_coinventor_features_sql
3    python -m pv.disambiguation.inventor.build_title_map_sql
```

Listing 4: Lignes de commandes pour construire les caractéristiques des inventeurs

Chaque code à un rôle bien défini :

- *build_assignee_features_sql.py* permet de construire un fichier contenant une map sérialisée qui elle meme contient les caractéristiques des cessionnaire des brevets.
- *build_coinventor_features_sql* permet de construire le même fichier, mais cette fois-ci avec les caractéristiques des co-inventeurs d'un brevet.
- *build_title_map_sql* idem, une map avec comme clé l'identifiant du brevet et comme valeur le titre du brevet est sérialisée.

En lisant les lignes de codes des fichiers ci-dessus, nous nous apercevons de plusieurs choses :

- La table SQL sur laquelle la requête est appliqué doit comporter les colonnes suivantes : | uuid | patent_id | assignee_id | rawlocation_id | type | name_first | name_last | organization | sequence |
- les codes sont écrits pour émettre des requêtes SQL dans des base de données d'inventeur MySQL.

Du premier point, nous tirons la conclusion suivante : Nos données semblent posséder la structure requise au bon fonctionnement de l'algorithme. En effet :

- l'attribut *uuid* correspond à *Unique User IDentifier*, ce qui dans notre cas s'apparente à l'attribut person_id de la table patents_applicants.
- l'attribut patent_id correspond à appln_id de la table patents_applicants.
- pour obtenir les attributs name_first et name_last, une manipulation s'impose. En effet, dans nos données, le nom et le prénom de l'inventeur sont réunis sous le meme attribut, à savoir person_name. Cependant, cet attribut possède une nomenclature claire : Nom, prénom. Ainsi avec la lignes de code 10 à 13 de la figure 3, on peut récupérer le nom et le prénom séparément.
- Pour les autres attributs, nous ne les possédons pas dans nos données. On décide de les omettre pour le moment car nous estimons qu'ils ne sont pas des éléments cruciaux pour la suite de l'algorithme.

Du second point, nous tirons la conclusion suivante : du fait que nos données sont hébergées sur Microsoft SQL Server, deux choix s'offrent à nous.

- effectuer une migration des données vers un hébergeur MySQL.
- réécrire l'intégralité des requêtes SQL, ce qui peut s'avérer nettement plus long que la première option.

Nous avons conjointement décidé d'adopter la seconde option, car c'est l'option dont nous étions le plus à l'aise vis-à-vis de nos compétences en python. De plus, cette dernière nous laissait la liberté de modifier le nom des colonnes appelés lors des requètes SQL, ce qui réglait en partie certaines interrogations levés par le premier point.

Ainsi, le code originel :

```
def build_granted(config):
    # | uuid | patent_id | assignee_id | rawlocation_id | type | name_first | name_last | organization | sequence |
    feature_map = collections.defaultdict(list)
    cnx = pvdb.granted_table(config)
    # if there was no table specified
    if cnx is None:
        return feature_map
    cursor = cnx.cursor()
    query = "SELECT uuid, patent_id, assignee_id, rawlocation_id, type, name_first, name_last, organization, sequence FROM rawassignee"
    cursor.execute(query)
    idx = 0
    for rec in cursor:
        am = AssigneeMention.from_granted_sql_record(rec)
        feature_map[am.record_id].append(am.assignee_name())
        idx += 1
        logging.log_every_n(logging.INFO, 'Processed %s granted records - %s features', 10000, idx, len(feature_map))
    logging.log(logging.INFO, 'Processed %s granted records - %s features', idx, len(feature_map))
    return feature_map
```

Listing 5: fichier *build_assignee_features_sql.py* issu du dépôt Github de PatentsView

devient :

```
   def build_granted():
    # | uuid | patent_id | assignee_id | rawlocation_id | type | name_first | name_last | organization | sequence |
    cnx = pyodbc.connect(f'DRIVER={{SQL Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;')
    cursor = cnx.cursor()

    query = "SELECT person_id, appln_id, person_name from patents_applicants"
    cursor.execute(query)
    feature_map = collections.defaultdict(list)
    idx = 0
    for person_id, appln_id, person_name  in cursor:
        parts = list(map(str.strip, person_name.split(",")))
        nom = parts[0]
        prenom = parts[1] if len(parts) > 1 else ""

        rec = [str(person_id), str(appln_id), str(person_id), None, prenom, nom, None, None, None]
```

```
        am = AssigneeMention.from_granted_sql_record(rec)
        feature_map[am.record_id].append(am.assignee_name())
        idx += 1
        logging.log_every_n(logging.INFO, 'Processed %s granted records - %s features', 10000, idx, len(feature_map))
    return feature_map
```

Listing 6: fichier *build_assignee_features_sql.py* - version adaptée

Le code ci-dessus effectue dans un premier temps une connexion avec la base de donné (qui est hébergé localement). Ensuite, on récupère pour chaque ligne de la table *patents_applicants* les attributs person_id, appln_id et person_name à l'aide de la requete SQL. Après cela, on instencie un objet InventorMention. Cet objet est alors stocké dans dictionnaire dont la clé est l'attribut record_id de l'objet *InventorMention* (i.e. l'attribut person_id de la table patents_applicants).

Un objet *InventorMention* est instancié à partir d'une classe correspondant à un inventeur. Voici son constructeur ci-dessous :

```
    def __init__(self, uuid, patent_id, rawlocation_id, name_first, name_last, sequence, rule_47, deceased,
                 document_number=None, city=None, state=None, country=None):
        self.uuid = str(uuid)
        self.patent_id = str(patent_id).replace('\"', '') if patent_id else None
        self.rawlocation_id = str(rawlocation_id).replace('\"', '') if rawlocation_id else ''
        self.raw_last = str(name_last).replace('\"', '') if name_last else ''
        self.raw_first = str(name_first).replace('\"', '') if name_first else ''
        if type(sequence) is int:
            self.sequence = str(sequence)
        else:
            self.sequence = sequence.replace('\"', '') if sequence else ''
        self.rule_47 = str(rule_47).replace('\"', '') if rule_47 else ''
        self.deceased = str(deceased).replace('\"', '') if deceased else ''
        self.name = '%s %s' % (self.raw_first, self.raw_last)
        self.document_number = str(document_number)

        self.mention_id = '%s-%s' % (self.patent_id, self.sequence) if self.patent_id is not None else 'pg-%s-%s' % (
            self.document_number, self.sequence)
        self.assignees = []
        self.title = None
        self.coinventors = []

        self._first_name = None
        self._first_initial = None
        self._first_letter = None
        self._first_two_initials = None
        self._first_two_letters = None
        self._middle_name = None
        self._middle_initial = None
        self._suffixes = None
        self._last_name = None

        self.city = str(city)
        self.state = str(state)
        self.country = str(country)

```

```
37          self.record_id = self.patent_id if self.patent_id else 'pg-%s' % self.
    document_number
```

Listing 7: Constructeur de la classe InventorMention

Nous voyons que, dans sa conception, la classe *InventorMention* est très flexible et c'est cette flexibilité qui nous permet d'adapter le code à notre cas d'étude.

Chacun des codes suivants (*build_coinventor_features_sql* et *build_title_map*) procèdent de manière identique au code affiché ci-dessus.

La suite du code consiste à exécuter la ligne suivante :

```
1     python -m pv.disambiguation.inventor.build_canopies_sql
```

Listing 8: Lignes de commandes pour construire les canopies

Ainsi, à l'aide des quatre commandes énumérées précédemment, nous obtenons des fichier de type *pickle*. Ces fichiers sont des sérialisations d'objets *InventorMention* dont les attributs varient selon la nature :

- Le premier fichier, nommée *assignee_features.both.pkl* est issu de l'exécution du code *build_assignee_featur*
  Les objets *InventorMention* sont instanciés par les informations des cessionnaires des brevets.
- le deuxième fichier, nommée *coinventor_features.both.pkl* est issu de l'éxecution du code *build_coinventor_f*
  Les objets *IventorMention* sont instanciés par les informations de la table *patents_inventors*.
- le troisième fichier , nommée *title_features.both.pkl* est issu de l'éxecution du code *build_title_map_sql.py*.
  Cette fois-ci, nous ne serialisons pas un objet mais une map.
- enfin, le quatrième fichier nommé *canopies_pregranted.pkl* est un fichier issu de l'exécution du code buil_canopies_sql.py. On instancie une map dont la clé est un *String* : c'est la première lettre du nom de famille de l'inventeur. La valeur associé à cette lettre sont les identifiants uniques des inventeurs.

3.4 Compilation de l'algorithme

Nous possédons donc quatre fichiers *pickles* contenant diverses informations requises au fonctionnement de la seconde phase de l'algorithme, à savoir le *clustering*. Avant cela, nous avons du installer certains modules pour faire fonctionner ce dernier :

```
1     pip install git+https://github.com/iesl/grinch.git
```

Listing 9: installation du module Grinch

Ce module, dont le propriétaire est Nicholas Monath, également co-éditeur du code de désambiguisation, est disponible en open-source ici. Ce code permet d'implémenter du *clustering* hiérarchique. Ce dernier nécessite, lors de son installation, d'utiliser une option (-Wno-cpp) que le compilateur windows ne prend pas en charge. C'est à cet instant que nous avons compris que le code à été conçu pour être exclusivement compilable avec un compilateur GCC (c'est-à-dire, une machine avec un OS linux ou macOS). Il se trouve que l'on dispose d'une tel machine.

Par la suite, le début du *clustering* s'effectue par la commande suivante :

```
    wandb sweep bin/inventor/run_all.yaml
    wandb agent $sweep_id
```

Listing 10: Commandes pour initier la phase de *clustering*

Vous l'aurez peut-être compris, la phase de *clustering* se fait via l'utilisation de wandb. Wandb, abréviation de Weights & Biases, est une plateforme de développement d'IA conçue pour aider les chercheurs, les ingénieurs en apprentissage automatique et les data scientists à suivre, visualiser, comparer et reproduire leurs expériences d'apprentissage automatique. Elle permet de construire de meilleurs modèles plus rapidement en offrant une vue d'ensemble complète du processus d'entraînement et d'évaluation des modèles.

Il faut donc créer un compte sur la plateforme pour ensuite récupérer son *token_id*

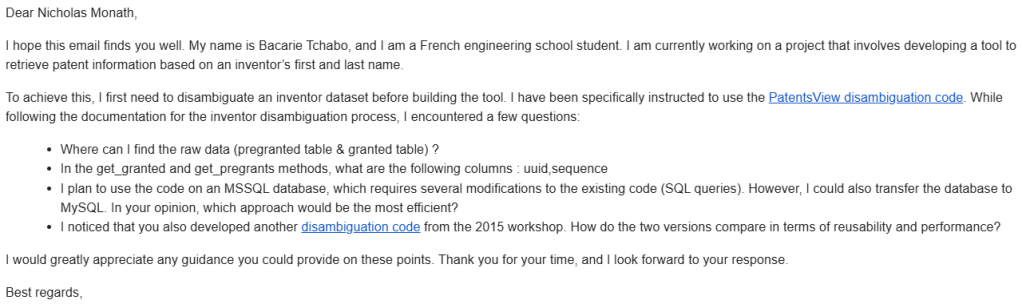
Après avoir effectué les configurations nécessaires, nous lançons la première commande et nous obtenons l'erreur suivante :

```
wandb: Starting wandb agent
2025-04-05 21:38:15,062 - wandb.wandb_agent - INFO - Running runs: []
2025-04-05 21:38:15,413 - wandb.wandb_agent - INFO - Agent received command: run
2025-04-05 21:38:15,413 - wandb.wandb_agent - INFO - Agent starting run with
    config:
        chunk_id: 0
        chunk_size: 10000
        min_batch_size: 1
        run_id: run_24
2025-04-05 21:38:15,416 - wandb.wandb_agent - INFO - About to run command: /usr/
    bin/env python pv/disambiguation/inventor/run_clustering.py --chunk_id=0 --
    chunk_size=10000 --min_batch_size=1 --run_id=run_24
Traceback (most recent call last):
  File "pv/disambiguation/inventor/run_clustering.py", line 12, in <module>
    from pv.disambiguation.inventor.load_mysql import Loader
ModuleNotFoundError: No module named 'pv'
```

Listing 11: Résultat de la compialtion de la commande de clustering

Pour une raison qui nous est inconnue, wandb ne reconnaît pas le module pv, module qui contient l'ensemble des codes nécessaires au cluster. Cette erreur aura été fatale dans l'avancée de cette partie de notre travail sur le sujet. Pris par le temps, nous avons décidé de mettre un terme à la mise en place de l'exécution de l'algorithme de désambiguïsation.

Nous avions pourtant, bien avant de rencontrer ces problèmes, intenté d'établir une communication avec l'un des co-auteurs du code - Nicholas Monath - afin d'éclaircir certaines zones d'ombres :



Dear Nicholas Monath,

I hope this email finds you well. My name is Bacarie Tchabo, and I am a French engineering school student. I am currently working on a project that involves developing a tool to retrieve patent information based on an inventor's first and last name.

To achieve this, I first need to disambiguate an inventor dataset before building the tool. I have been specifically instructed to use the PatentsView disambiguation code. While following the documentation for the inventor disambiguation process, I encountered a few questions:

- Where can I find the raw data (pregranted table & granted table) ?
- In the get_granted and get_pregrants methods, what are the following columns : uuid,sequence
- I plan to use the code on an MSSQL database, which requires several modifications to the existing code (SQL queries). However, I could also transfer the database to MySQL. In your opinion, which approach would be the most efficient?
- I noticed that you also developed another disambiguation code from the 2015 workshop. How do the two versions compare in terms of reusability and performance?

I would greatly appreciate any guidance you could provide on these points. Thank you for your time, and I look forward to your response.

Best regards,

Figure 2: Mail envoyé à l'un des co-auteurs du code

Malheureusement, ce mail n'aboutira à rien, probablement car *PatentsView*, l'organisme éditeur et en charge de la maintenance du code, à du mettre fin à ses activité le 28 mars 2025 Cette fin est

d'autant plus amère car nous étions à une étape du but final de cette partie. Nous avions prévu d'évaluer les performances de l'algorithme à l'aide de métriques tels que la précision et le rappel. Ce travail à déjà été effectué avec diverses données pour cet algorithme mais pas pour des donnés de brevet européen. Il aurait été fort intéressant d'observer le comportement de l'algorithme vis-à-vis de ce type de donnée.

Pour revenir au problème ; nous pensons que avec un peu plus de temps devant nous, nous aurions sûrement pu prendre en main la plateforme wandb et adapter une nouvelle fois l'algorithme pour avoir une meilleure mainmise sur ce dernier.

4 Interface graphique

4.1 Préambule

Bien que l'erreur de compilation ci-dessus rend en partie caduc la suite de notre travail, nous avons quand même voulu proposer une interface graphique dont nous présentons la conception dans la partie ci-après.

Cette section détaille la conception et le développement d'une interface graphique utilisateur conviviale pour faciliter la désambiguïsation des ensembles de données d'inventeurs. L'interface graphique, construite à l'aide de de la bibliothèque graphique de python *Tkinter*, permet aux utilisateurs de saisir les informations sur les inventeurs et de récupérer les résultats désambiguïsés. Cette interface est essentielle pour fournir aux chercheurs et autres utilisateurs un outil accessible pour interagir avec le système de désambiguïsation des inventeurs.

4.2 Technologies employées

- Tkinter : L'interface graphique est développée à l'aide de Tkinter, une bibliothèque Python standard pour la création d'interfaces graphiques utilisateur. Tkinter a été choisi pour sa facilité d'utilisation, sa compatibilité multiplateforme et son intégration avec Python.
- ElasticSearch : ElasticSearch est utilisé comme moteur de recherche et d'indexation principal. Il offre des capacités de recherche rapides, efficaces et flexibles, cruciales pour traiter de grands ensembles de données et effectuer des recherches de similarité.

4.3 Conception et Fonctionnalités de l'Interface

L'interface graphique comprend les éléments clés suivants :

Un formulaire de saisie des informations sur l'inventeur, qui permet aux utilisateurs de saisir les informations sur l'inventeur, y compris le prénom et le nom.

Affichage des Résultats de Désambiguïsation : Présente la correspondance d'inventeur la plus probable en fonction de la saisie. Affiche les détails de l'inventeur. Fournit un lien vers une page séparée listant tous les brevets associés à l'inventeur identifié. Liste des Inventeurs Similaires : Affiche une liste des 5 inventeurs les plus similaires. Inclut un score de similarité pour chaque inventeur, indiquant le degré de correspondance avec la requête saisie. Page de Liste des Brevets : Une page dédiée affiche une liste complète des brevets pour un inventeur sélectionné. Inclut les détails pertinents du brevet (par exemple, titre, résumé, date de publication).

4.4 Justification de l'utilisation d'ElasticSearch

ElasticSearch a été choisi plutôt que la recherche directe dans la base de données Microsoft SQL Server pour plusieurs raisons :

- Capacités de recherche en texte intégral avancées : ElasticSearch excelle dans la recherche en texte intégral, permettant une correspondance plus flexible et tolérante des noms d'inventeurs. Il peut gérer les variations d'orthographe, les abréviations et l'ordre des noms, ce qui est essentiel pour désambiguïser les noms d'inventeurs. Microsoft SQL Server Management, bien que capable de recherches textuelles via des requêtes SQL, n'est pas aussi optimisé à cet effet.
- Score de Pertinence et classement : ElasticSearch fournit un score de pertinence robuste, classant les résultats de recherche en fonction de la similarité avec la requête. Ceci est essentiel pour présenter les correspondances d'inventeurs les plus probables et la liste des inventeurs similaires avec des scores de similarité. Les capacités de recherche intégrées de Microsoft SQL Server Management offrent un classement de pertinence limité.
- Vitesse et Évolutivité : ElasticSearch est conçu pour la vitesse et l'évolutivité, capable de traiter de grands ensembles de données et des volumes de requêtes élevés. Ceci est crucial pour offrir une expérience utilisateur réactive, en particulier lors du traitement de millions d'enregistrements de brevets. Microsoft SQL Server Management peut devenir lent et inefficace avec de très grands ensembles de données, en particulier pour les requêtes de recherche complexes.

En résumé, ElasticSearch offre des capacités de recherche supérieures à celles des requêtes SQL directes, ce qui en fait le choix idéal pour l'application de désambiguïsation d'inventeurs. Sa recherche en texte intégral, son score de pertinence, sa vitesse, son évolutivité et ses fonctionnalités de recherche avancées sont essentielles pour fournir des résultats précis et efficaces.

5 Discussion critique et perspectives d'amélioration

5.1 Bilan critique du travail réalisé

Ce projet nous a permis de mieux comprendre les enjeux techniques liés à la désambiguïsation d'identités dans des bases de données massives, en particulier dans le domaine des brevets. À travers l'exploration et le nettoyage de la base EPO, l'adaptation d'un algorithme externe, et la conception d'une interface graphique, nous avons confronté un grand nombre de défis, tant sur le plan des données que sur le plan logiciel.

Malgré des efforts importants, l'exécution complète de l'algorithme de désambiguïsation n'a pas pu aboutir en raison de problèmes techniques liés à la plateforme WandB, à l'environnement d'exécution, et au code non maintenu depuis la fermeture de PatentsView. Ce blocage constitue une limite majeure à notre évaluation des performances finales du système.

Cependant, notre adaptation des scripts, nos tests d'intégration avec SQL Server, ainsi que le prétraitement et l'indexation des données montrent que les bases ont été solidement posées pour une future exécution complète.

5.2 Forces du projet

- Une bonne compréhension des structures de la base EPO et de ses limites (encodage, incohérences, volume).
- Une adaptation réussie de plusieurs scripts Python pour les rendre compatibles avec un environnement SQL Server local.
- Une interface utilisateur fonctionnelle, couplée à ElasticSearch, qui offre une expérience de recherche fluide et rapide.
- Une démarche rigoureuse dans l'analyse critique des solutions explorées (recherches textuelles, nettoyage, etc.).

5.3 Limites rencontrées

- La non-compatibilité du code original avec Windows, nécessitant un environnement Linux ou Mac pour la compilation.
- La dépendance forte à des outils externes (WandB, Grinch) mal documentés ou plus maintenus.
- L'impossibilité de tester les performances du modèle de désambiguïsation sur un échantillon significatif.
- L'absence d'une base de référence "gold standard" pour valider les regroupements produits par l'algorithme.

5.4 Pistes d'amélioration

- Déployer une machine virtuelle ou un conteneur Docker sous Linux pour garantir la portabilité et la reproductibilité de l'algorithme.
- Remplacer WandB par un outil local ou open-source plus simple à configurer pour le suivi des expériences.
- Tester un autre modèle de désambiguïsation, plus léger ou plus documenté, comme ceux utilisés dans la bibliométrie ou sur des jeux de données type ORCID.
- Intégrer une étape de validation manuelle sur un échantillon de résultats, afin de mesurer précisément les taux de faux positifs/négatifs.
- Explorer des approches de désambiguïsation hybrides, combinant apprentissage automatique et règles heuristiques simples (e.g. similarité de co-inventeurs, adresses...).

6 Conclusion

Ce projet a mis en lumière la complexité réelle d'un problème en apparence simple : identifier une personne à partir d'un nom. Il a montré que la désambiguïsation ne peut pas reposer uniquement sur du nettoyage ou des requêtes SQL, mais nécessite une approche algorithmique robuste, capable d'exploiter des signaux faibles et des relations contextuelles. Malgré les obstacles techniques rencontrés, les travaux réalisés constituent un socle solide pour aller plus loin dans l'exploitation des bases de données brevets à grande échelle.

PROJET ING2

[ETU22] Disambiguation of inventor datasets - Documentation

Bacarie TCHABO
Iyad BEN MOSBAH
Salim OUESLATI
Charles LIU

Responsable : François Maublanc
Encadrant : Pierre Andry

2024-2025

Contents

1 Introduction

Ce document est une documentation pour notre solution de désambiguïsation des inventeurs pour une base de donné. La première partie donne les indications pour appliquer l'algorithme de désambiguïsation. La seconde partie donne les indications pour utiliser l'interface graphique.

2 Algorithme de désambiguïsation

2.1 Prérequis

Notre version du code de désambiguïsation s'applique sur une base de donnée hébergé sur Microsoft SQL Server.

2.2 Indications

Cette partie de la documentation donne les indications pour appliquer la désambiguïsation. La première étape est en partie facultative.

Tout d'abord, il faut construire les fichiers pickles : ce sont des fichiers contenant des objets python sérialisés.

La construction de ces fichiers requiert d'abord quelques ajustements :

```
    server='Student-laptop'
    database='teseo_inventors'
```

Listing 1: Lignes de commandes pour cosntruire les caractéristiques des inventeurs

Dans le fichier build_assignee_features_sql.py situé dans le dossier PatentsView-Disambiguation-main -> pv -> disambiguation -> inventor , il faudra changer les informations de connexion à la base de donné.

Il faut faire de même pour les fichiers build_coinventor_features_sql.py, build_title_map_sql.py et build_canopies_sql.py. Ces trois fichiers sont situés dans le meme dossier.

Chacun de ces fichiers dispose d'une fonction build_granted(). Cette fonction constitue le squelette de chacun des fichiers. Si vous d'informations supplémentaires dans votre base de données d'inventeurs, i.e. si vous disposez de colonnes supplémentaires dans votre base de données, vous pouvez modifier les requêtes SQL à votre guise.

Voici la fonction build_granted() :

```
    def build_granted():
     # | uuid | patent_id | assignee_id | rawlocation_id | type | name_first | name_last | organization | sequence |
     cnx = pyodbc.connect(f'DRIVER={{SQL Server}};SERVER={server};DATABASE={database};Trusted_Connection=yes;')
     cursor = cnx.cursor()

     query = "SELECT person_id, appln_id, person_name from patents_applicants"
     cursor.execute(query)
     feature_map = collections.defaultdict(list)
     idx = 0
     for person_id, appln_id, person_name  in cursor:
         parts = list(map(str.strip, person_name.split(",")))
         nom = parts[0]
         prenom = parts[1] if len(parts) > 1 else ""

         rec = [str(person_id), str(appln_id), str(person_id), None, prenom, nom, None, None, None]

         am = AssigneeMention.from_granted_sql_record(rec)
         feature_map[am.record_id].append(am.assignee_name())
         idx += 1
         logging.log_every_n(logging.INFO, 'Processed %s granted records - %s features', 10000, idx, len(feature_map))
     return feature_map
```

Listing 2: fichier *build_assignee_features_sql.py*

Pour les autres fichiers, la fonction build_granted() est identique.

Une fois que ces modifications on été aporté, vous pouvez lancer les commandes :

```
    python -m pv.disambiguation.inventor.build_assignee_features_sql
    python -m pv.disambiguation.inventor.build_coinventor_features_sql
    python -m pv.disambiguation.inventor.build_title_map_sql
    python -m pv.disambiguation.inventor.build_canopies_sql
```

Listing 3: Lignes de commandes pour construire les caractéristiques des inventeurs

Une fois ces commandes lancées, vous aurez dans le repertoire quatre fichiers pickles.

Une fois que ces fichier on été construit, il faut procéder au clustering : Il faut dans un premier temps établir sur votre machine une connexion à wandb :

Étape 1 : Installation de la librairie wandb

La première étape consiste à installer la librairie Python wandb en utilisant pip. Ouvrez votre terminal ou invite de commandes et exécutez la commande suivante :

```
    pip install wandb
```

Cette commande téléchargera et installera la dernière version de la librairie wandb ainsi que ses dépendances.

Étape 2 : Se connecter à votre compte wandb

Une fois l'installation terminée, vous devez vous connecter à votre compte Weights Biases depuis votre machine. Exécutez la commande suivante dans votre terminal :

```
wandb login
```

Cette commande vous demandera votre clé API. Vous pouvez trouver votre clé API sur votre profil Weights Biases :

Connectez-vous à votre compte sur wandb. Cliquez sur votre photo de profil en haut à droite. Sélectionnez "Settings" (Paramètres) dans le menu déroulant. Faites défiler vers le bas jusqu'à la section "API Keys" (Clés API). Copiez votre clé API et collez-la dans le terminal lorsque la commande wandb login vous le demande. Après avoir entré votre clé API, wandb devrait afficher un message indiquant que vous êtes connecté avec succès.

Une fois que vous êtes connectés, vous pouvez lancer le clustering :

```
wandb sweep bin/inventor/run_all.yaml
wandb agent $sweep_id
```

Listing 4: Lignes de commandes pour le clustering

La dernière ligne de commande à effectuer est la suivante :

```
python -m pv.disambiguation.inventor.finalize
```

Listing 5: Lignes de commandes pour la finalisation de l'algorithme

3 Interface graphique

3.1 Utilisation de l'Interface

Lancement de l'Application :

Pour lancer l'application, exécutez le script Python principal (interfaceGraphique.py). Assurez-vous que Python et toutes les bibliothèques nécessaires (Tkinter, elasticsearch, pyodbc) sont installés. Le cas échéant, il faut les installer :

```
    pip install tkinter
    pip install elasticsearch
    pip install pyobdc
```

Listing 6: Lignes de commandes d'installations des modules

Saisie du Nom de l'Inventeur :

Dans la fenêtre de l'application, vous verrez un champ intitulé "Nom de la personne:" et "prénom de la personne" .

Entrez le nom (ou une partie du nom) de l'inventeur que vous recherchez. L'application est conçue pour être tolérante aux fautes d'orthographe et aux variations de nom.

Cliquez sur le bouton "Rechercher".

L'application affichera les résultats de la recherche dans la partie inférieure de la fenêtre. Le résultat le plus probable sera affiché en premier, avec un score de pertinence indiquant la qualité de la correspondance.

Une liste des 5 inventeurs les plus similaires sera également affichée, chacun avec son propre score de similarité. Cela vous permet d'explorer d'autres candidats potentiels. Si aucun résultat n'est trouvé, un message "Aucun résultat trouvé." sera affiché.

3.2 Récupération de vos informations Elastic Cloud

Ce tutoriel se concentre sur la configuration sécurisée des connexions à ElasticSearch, en particulier lors de l'utilisation d'Elastic Cloud.

Elastic Cloud : Si vous utilisez le service hébergé d'Elastic (Elastic Cloud), vous utiliserez généralement le cloud$_i detl'api_key(ounomd'utilisateur/motdepasse, maislesclsAPIsontprfres).Auto-hberg : Sivousavezinstall ElasticSearchsurvotrepropremachine, vousutiliserezleparamtrehostspourspci$

2. Obtenir les Identifiants (Elastic Cloud)

Se connecter : Allez sur cloud.elastic.co et connectez-vous. Choisissez le déploiement ElasticSearch auquel vous souhaitez vous connecter.Trouvez le cloud$_i ddanslavued'ensembledevotredploiement.$

Allez dans "Management" -> "Security" -> "API Keys". Créez une nouvelle clé API. Donnez-lui un nom descriptif. Important : Copiez l'api_key_id.

Vous devrez recopiez cette clé dans la partie suivante du code :

3.3 Modifications de Base de l'Interface

Ce guide couvre les modifications simples de l'interface graphique à l'aide de Tkinter.

Le code est organisé en plusieurs fonctions : recuperer_donnees_inventeurs_depuis_mssql(): Récupère les données des inventeurs depuis la base de données SQL Server.

connecter_a_elasticsearch(): Établit une connexion avec ElasticSearch.

creer_et_indexer_inventeurs(): Crée un index dans ElasticSearch et y indexe les données des inventeurs.

rechercher_inventeurs(): Effectue la recherche d'inventeurs dans ElasticSearch.

effectuer_recherche(): Fonction appelée lorsque l'utilisateur clique sur le bouton "Rechercher"; elle récupère l'entrée de l'utilisateur et lance la recherche.

display_results(): Efface et affiche les résultats de la recherche dans l'interface graphique.

Vous pouvez modifer ces fonctions pour qu'elles correspondent à vous éxigeances de recherche.