Classification Automatique de Plaintes Financières

De la donnée brute à l'application web : Une approche NLP fondamentale

Abdelli Farès
Page Github : github.com/Fares596

1 Le Contexte & Le Défi

Lorsqu'un consommateur soumet une plainte, celle-ci doit être orientée vers le service compétent. L'objectif de ce projet est de construire un modèle d'intelligence artificielle capable de **lire** une plainte en langage naturel et de **prédire** sa catégorie avec une haute précision, sans recourir au Deep Learning.

1.1 Le Jeu de Données (CFPB)

Nous avons utilisé la base de données publique du *Consumer Financial Protection Bureau* (US). Ce jeu de données labelisées est constitué de plaintes réelles déposées par des consommateurs américains contre des institutions financières.

— **Volume :** Environ 400 000 plaintes exploitables (non vides) après nettoyage.

— **Nature :** Texte non structuré, bruité (fautes, majuscules) et anonymisé (masques "XXXX").

1.2 Stratégie de Consolidation des Classes

Une analyse exploratoire a révélé un déséquilibre des catégories. Pour équilibrer nos classes, nous avons fusionné certaines petites classes proches sémantiquement(ex : *Credit Card* et *Prepaid Card*), aboutissant à une liste finale de **9 catégories cibles** :

1. **Credit reporting, credit repair services, or other consumer reports** (Problèmes de score crédit)
2. **Debt collection** (Recouvrement de dettes)
3. **Mortgage** (Prêts immobiliers)
4. **Credit card or prepaid card** (Cartes bancaires)
5. **Checking or savings account** (Comptes courants/épargne)
6. **Student loan** (Prêts étudiants)
7. **Payday loan, title loan, or personal loan** (Prêts à la consommation)
8. **Money transfer, virtual currency, or money service** (Virements et cryptomonnaies)
9. **Vehicle loan or lease** (Prêts automobiles - Classe minoritaire)

1.3 Exemple de Donnée Brute

Voici un exemple typique de plainte (catégorie *Credit Reporting*) telle qu'elle est reçue par le système. Notez l'anonymisation ("XXXX") qui pollue les données et qu'il faudra traiter lors de la phase de preprocessing.

Exemple Réel du Dataset

"XXXX and Transunion are reporting incorrectly that I am 120 days past due on loans with the XXXX - partial account numbers XXXX; XXXX; XXXX; XXXX.. These accounts reflect a {$0.00} balance and a {$0.00} past due. I have contacted the two bureaus and requested these coding errors be corrected. This incorrect reporting is harming my credit score."

2 Les 3 méthodes de Natural Language Processing (NLP) utilisées

Pour transformer le texte brut des plaintes en données exploitables par un algorithme de classification qui ne peut traiter que des valeurs numériques, nous avons appliqué un pipeline de traitement du langage naturel structuré en trois étapes fondamentales.

2.1 La Tokenization

La tokenization est l'étape initiale de segmentation. Elle consiste à découper la chaîne de caractères brute (le texte de la plainte) en unités sémantiques élémentaires appelées **tokens**. Dans notre approche, ces tokens correspondent principalement aux mots, débarrassés de la ponctuation et normalisés (mise en minuscules). Cette opération transforme une donnée non structurée (une phrase continue) en une structure de liste ordonnée, prélude indispensable à toute analyse statistique.

2.2 Les N-Grams

Afin de capturer le contexte local des mots, nous ne nous sommes pas limités aux mots isolés (*unigrammes*). Nous avons utilisé des **N-Grams**, qui sont des séquences contiguës de N tokens.

- **Unigrammes ($N = 1$)** : Capturent le vocabulaire simple (ex : "credit", "card").
- **Bigrammes ($N = 2$)** : Capturent les paires de mots consécutifs (ex : "credit card").

L'intégration des bigrammes est cruciale pour comprendre un texte. Elle permet au modèle de traiter une expression composée comme une entité unique, et ainsi mieux saisir le contexte des mots qu'il lit.

2.3 La Vectorisation TF-IDF

Pour convertir les tokens en valeurs numériques (vecteurs), nous avons utilisé la méthode **TF-IDF** (*Term Frequency - Inverse Document Frequency*). Cette technique pondère l'importance d'un mot en suivant l'heuristique suivante : un mot est important s'il est fréquent dans le document (ici la plainte) actuel, mais rare dans l'ensemble du corpus, ce qui permet d'éliminer les mots génériques.

Formulation mathématique Le poids $W_{t,d}$ d'un terme t dans un document d est le produit de deux métriques :

$$W_{t,d} = \mathrm{tf}_{t,d} \times \mathrm{idf}_t$$

1. **Term Frequency (TF)** : Mesure la fréquence locale.

$$\mathrm{tf}_{t,d} = \frac{\text{Nombre d'occurrences de } t \text{ dans } d}{\text{Nombre total de mots dans } d}$$

2. **Inverse Document Frequency (IDF)** : Mesure la rareté globale. Elle pénalise les mots omniprésents.

$$\mathrm{idf}_t = \log\left(\frac{N}{\mathrm{df}_t}\right)$$

   Où N est le nombre total de documents dans le corpus, et df_t est le nombre de documents contenant le terme t.

Résultat en sortie La vectorisation produit une **matrice creuse** (Sparse Matrix) de dimensions $(D \times V)$, où D est le nombre de documents et V la taille du vocabulaire retenu.

- Chaque ligne représente une plainte.
- Chaque colonne représente un mot ou un N-gramme du vocabulaire.
- La valeur de la cellule est le score **TF-IDF** (un nombre réel positif) si le mot est présent, et **0** si le mot est absent.

3 Implémentation et Résultats

3.1 Le Prétraitement (Preprocessing)

Nous avons développé une fonction de nettoyage spécifique qui effectue les opérations suivantes : normalisation, suppression des masques d'anonymisation ("XXXX") et lemmatisation.

```python
def clean_text(text):
    """
    Cleans the raw text:
    1. Removes 'XXXX' masks
    2. Removes punctuation/numbers
    3. Lemmatizes and removes stopwords
    4. Truncates to 500 words
    """
    if not isinstance(text, str):
        return ""

    # 1. Remove masks (XXXX)
    text = re.sub(r'X+', '', text)

    # 2. Remove non-alphabetical characters
    text = re.sub(r'[^a-zA-Z\s]', '', text)

    # 3. Lowercase
    text = text.lower()

    # 4. Tokenize
    tokens = text.split()

    # 5. Lemmatize & Remove Stopwords
    clean_tokens = [
        lemmatizer.lemmatize(token)
        for token in tokens
        if token not in stop_words and len(token) > 2
    ]

    # 6. Truncate (keep first 500 words)
    clean_tokens = clean_tokens[:500]

    return " ".join(clean_tokens)
```

Listing 1 – Fonction de nettoyage des plaintes (clean_text)

Exemple d'exécution : clean_text()

Entrée (Brut) :

`"The XXXX bank is charging me fees for late payments on my accounts!!"`

Opérations effectuées :

- **Regex :** Suppression du masque d'anonymisation "XXXX" et de la ponctuation "!!".
- **Stopwords :** Suppression des mots vides ("The", "is", "me", "for", "on", "my").
- **Lemmatisation :** Normalisation grammaticale.
  - *charging* → **charge** (Verbe ramené à l'infinitif)
  - *fees* → **fee**, *payments* → **payment** (Pluriels ramenés au singulier)

Sortie (Nettoyé) :

`"bank charge fee late payment account"`

3.2 Le Pipeline Scikit-Learn

L'architecture du modèle repose sur un objet `Pipeline` qui enchaîne le `TfidfVectorizer` (N-grams 1 à 2 pour capter le contexte local) et la `LogisticRegression` avec une pondération *balanced* pour gérer le déséquilibre des classes.

```
# Define the Pipeline: Vectorizer + Classifier
nlp_pipeline = Pipeline([
    # Step 1: Text to Numbers
    ('tfidf', TfidfVectorizer(
        ngram_range=(1, 2),  # Unigrams and Bigrams
        min_df=5,            # Ignore extremely rare words
        max_df=0.9,          # Ignore extremely common words
        max_features=10000,  # Limit vocabulary size
        stop_words='english'
    )),

    # Step 2: Classification Model
    ('clf', LogisticRegression(
        class_weight='balanced', # Handle class imbalance
        solver='liblinear',      # Optimized for high-dimensional data
        max_iter=1000            # Allow convergence
    ))
])
```

Listing 2 – Définition du Pipeline d'entraînement

3.3 Performance du Modèle

Le modèle a été évalué sur un ensemble de test de 76 655 plaintes. Voici les résultats détaillés par classe.

Catégorie	Precision	Recall	F1-Score	Support
Checking or savings account	0.80	0.83	0.81	5 553
Credit card or prepaid card	0.79	0.83	0.81	8 334
Credit reporting, repair, etc.	0.90	0.83	0.86	24 793
Debt collection	0.84	0.83	0.83	17 342
Money transfer, virtual currency	0.69	0.78	0.73	1 396
Mortgage	**0.92**	**0.93**	**0.93**	10 598
Payday loan, title loan or personal loan	0.53	0.53	0.53	3 128
Student loan	0.85	0.90	0.87	4 362
Vehicle loan or lease	0.35	0.56	0.43	1 149
Accuracy globale			**0.83**	76 655

TABLE 1 – Rapport de classification sur les données de test

Observations :

— On observe une performance correcte sur les catégories majeures (Mortgage, Credit Reporting).

— D'autres classes sont moins bien identifiées. La matrice de confusion révèle par exemple que la catégorie **Vehicle loan** est souvent classée à tort comme **Payday loan**. Cela souligne une limite intrinsèque du modèle : ces produits financiers partageant un vocabulaire très similaire, il est difficile de les différencier efficacement en se basant uniquement sur des unigrammes et des bigrammes, sans analyse du contexte global.

— La catégorie **Student loan**, elle, est beaucoup mieux reconnue par le modèle, certainement grâce au vocabulaire très spécifique qui lui est attaché (Studies, University, School,..).

4 Interface utilisateur pour tester le modèle

L'interface repose intégralement sur la librairie Python **Streamlit**. L'application charge le modèle entraîné (`.pkl`) et exécute le pipeline de nettoyage en temps réel lors de la saisie utilisateur.

4.1 Démonstration

L'utilisateur saisit simplement sa plainte dans la zone dédiée.

FIGURE 1 – Interface d'accueil de l'application Streamlit

Une fois l'analyse lancée, le modèle retourne la catégorie la plus probable ainsi que les scores de confiance associés à sa prédiction.

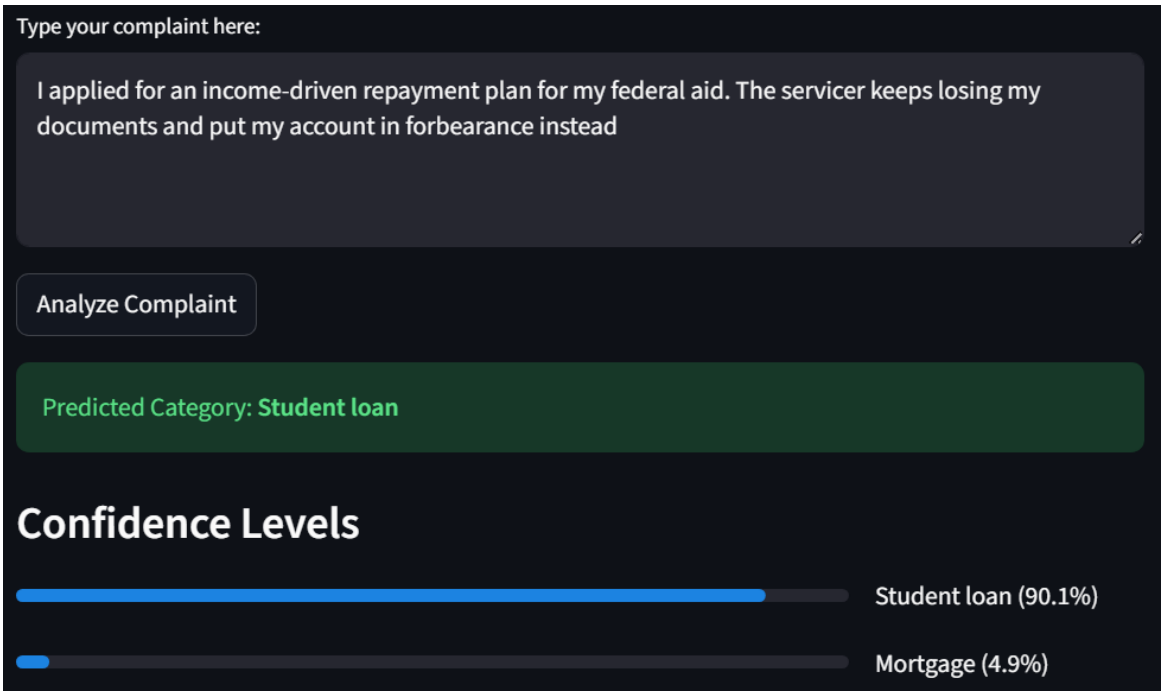

FIGURE 2 – Résultat de l'analyse avec degrés de confiance

5 Bilan d'Apprentissage & Perspectives

Ce projet de classification de plaintes financières a été conçu comme une mise en application des compétences acquises lors de la formation *"NLP in Python : Probability Models, Statistics , Text Analysis"* (Udemy).

5.1 Synthèse des compétences techniques

La réalisation de ce projet m'a permis de valider la maîtrise de la chaîne de traitement NLP fondamentale :

— **Preprocessing & Tokenization :** Nettoyage de données brutes non structurées via Regex, suppression des *stopwords* et normalisation par Lemmatisation.

— **Vectorisation :** Application du **TF-IDF** et des **N-Grams** pour transformer le texte en vecteurs numériques pertinents.

— **Modélisation :** Entraînement et évaluation d'un modèle supervisé (Régression Logistique) en contexte réel (gestion du déséquilibre des classes).

5.2 Certification

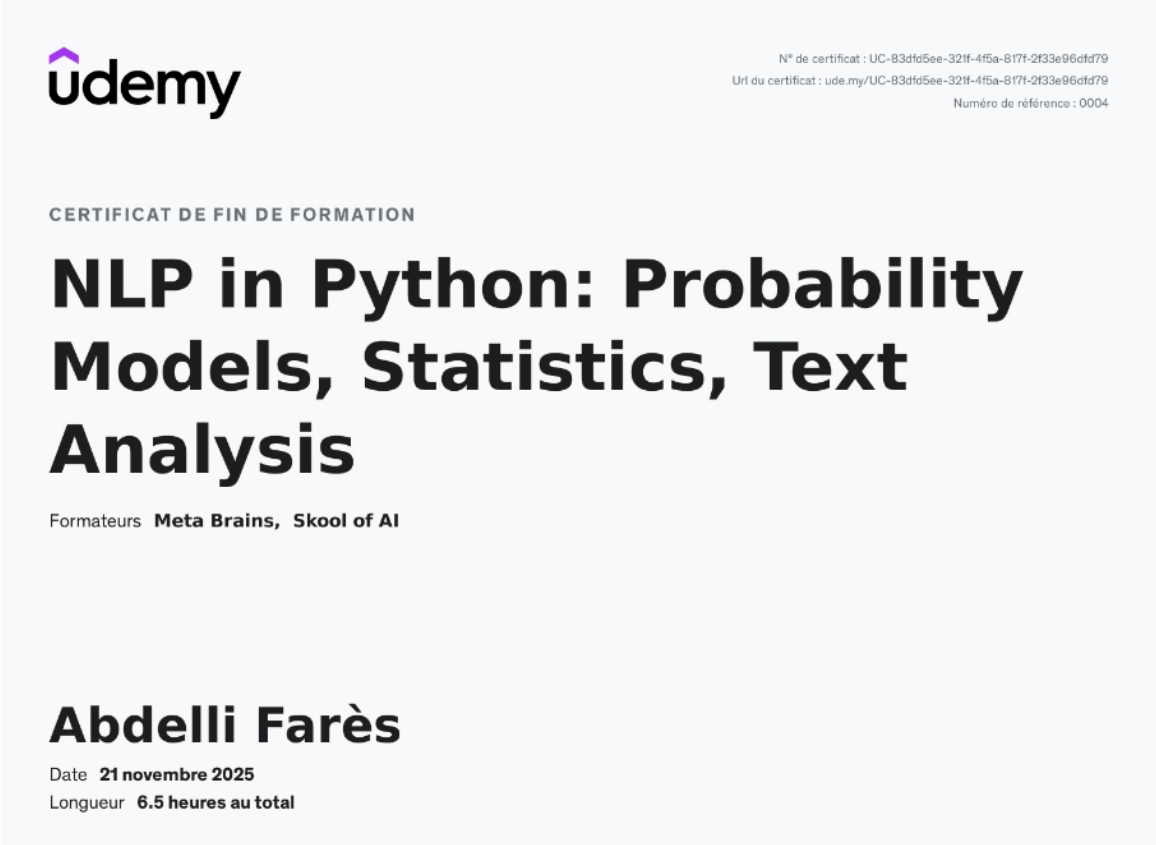

FIGURE 3 – Certification obtenue : NLP in Python (Udemy)

En conclusion, ce projet constitue une base solide pour mes futures explorations en NLP, notamment vers les architectures plus complexes et modernes de type Transformers (BERT, GPT) pour pallier les limites sémantiques identifiées lors de cette étude.