ECOLE MAROCAINE DES SCIENCES DE L'INGENIEUR
Membre de HONORIS UNITED UNIVERSITIES

Data Mining pour le Commerce en Ligne : De la Préparation des Données à la Classification

Réalisé par :

ELBOUZIDI Hamza
IMASSADAN Salma
MOATASSIM Bilal
AZILAL Amine

Encadré par :

Prof. HADNI Maryem

2025/2026

Table des matières

1. Introduction

Dans le cadre de ce projet, nous avons exploré et appliqué un processus structuré visant à classifier des données textuelles. La classification est une étape cruciale dans le domaine de la science des données, car elle permet de segmenter et d'interpréter les informations de manière significative.

Ce rapport met en lumière les étapes clés, allant du prétraitement des données jusqu'à l'évaluation des modèles de classification. L'objectif principal était d'extraire des caractéristiques pertinentes à partir de données brutes, puis de construire des modèles capables de prédire avec précision les catégories associées aux observations.

Le processus s'articule autour des axes suivants :

1. **Prétraitement des données** pour nettoyer et structurer les informations textuelles.
2. **Extraction et sélection des caractéristiques** pour transformer les textes en représentations numériques exploitables.
3. **Implémentation et comparaison de modèles de classification** afin d'identifier l'approche la plus performante.
4. **Évaluation des performances des modèles** à l'aide de métriques quantitatives.

Le jeu de données *Online Retail* contient des informations sur les transactions réalisées par un magasin en ligne. L'objectif de cette analyse est de comprendre la structure des données, détecter les anomalies, et préparer les données pour des analyses ultérieures ou pour des modèles prédictifs.

Pour réaliser cette analyse, nous avons utilisé le langage Python pour manipuler les données et exporter les résultats dans un fichier Excel. Les étapes suivantes ont été suivies :

2. Importation des bibliothèques

Les bibliothèques suivantes ont été importées :

- Manipulation des données : `pandas`, `numpy`.
- Division des données : `train_test_split`.
- Extraction de caractéristiques : `TfidfVectorizer`.
- Classification : `MultinomialNB`, `SVC`.
- Évaluation des performances : `classification_report`, `accuracy_score`.

Exemple de code :
```python
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import SVC
from sklearn.metrics import classification_report, accuracy_score
```

3. Prétraitement des données (preprocessing):

3.1 Structure Initiale des Données

Pour mieux comprendre le jeu de données, nous avons affiché des informations générales concernant le DataFrame. Cela inclut le nombre de lignes et de colonnes, les types de données de chaque colonne, ainsi que le nombre de valeurs non nulles par colonne. Ces informations nous aident à évaluer la structure et la qualité des données disponibles.

```
data.info()
```

Résultat :

- **Nombre total de lignes** : 541 909.
- **Nombre total de colonnes** : 8.
- Les colonnes comprennent des données de type `object`, `int64`, et `float64`.
- **Colonnes avec des valeurs nulles** :
  - `Description` : 1 454 valeurs manquantes.
  - `CustomerID` : 135 080 valeurs manquantes.

3.2 Analyse des Doublons

Nous avons identifié le nombre de lignes dupliquées dans le jeu de données à l'aide de la méthode `duplicated()`. Cela permet de détecter des redondances pouvant biaiser les analyses futures.

```
print("Duplicates:", data.duplicated().sum())
```

Résultat : Le jeu de données contient **5 268 lignes dupliquées**, qui seront supprimées au cours du nettoyage.

3.3 Analyse des Valeurs Manquantes

Nous avons compté le nombre total de valeurs manquantes dans l'ensemble des colonnes à l'aide de `isnull().sum().sum()`.

```
print("Null values:", data.isnull().sum().sum())
```

Résultat : Le jeu de données contient **136 534 valeurs manquantes**, principalement dans les colonnes `Description` et `CustomerID`.

3.4 Statistiques Descriptives

Pour évaluer la distribution des valeurs des colonnes numériques (`Quantity`, `UnitPrice`, et `CustomerID`), nous avons généré des statistiques descriptives. Ces statistiques incluent la moyenne, l'écart type, les valeurs minimales et maximales, ainsi que les quartiles.

```
data.describe()
```

Résultat :

Colonne	Moyenne	Écart-type	Min	Q1	Médiane	Q3	Max
Quantity	9.55	218.08	-80 995	1	3	10	80 995
UnitPrice	4.61	96.76	-11 062	1.25	2.08	4.13	38 970
CustomerID	15 287.69	1 713.60	12 346	13 953	15 152	16 791	18 287

3.5 Nettoyage des Données

Nous avons effectué plusieurs étapes de nettoyage pour garantir la qualité des données :

1. **Suppression des Doublons** :
   Nous avons supprimé les lignes dupliquées à l'aide de la méthode `drop_duplicates()`.

   ```
   data = data.drop_duplicates()
   ```

2. **Gestion des Valeurs Manquantes** :
   - Les valeurs manquantes dans la colonne `Description` ont été remplacées par le mot-clé `"Unknown"` pour conserver la cohérence.
   - Les lignes avec des `CustomerID` manquants ont été supprimées.

   ```
   data['Description'] = data['Description'].fillna("Unknown")
   data = data.dropna(subset=["CustomerID"])
   ```

3. **Filtrage des Données Invalides** : Nous avons conservé uniquement les lignes où les colonnes `Quantity` et `UnitPrice` contiennent des valeurs positives.

   ```
   data = data[(data['Quantity'] > 0) & (data['UnitPrice'] > 0)]
   ```

4. **Conversion des Dates** :
   La colonne `InvoiceDate`, initialement de type `object`, a été convertie au format datetime pour faciliter les analyses temporelles.

   ```
   data['InvoiceDate'] = pd.to_datetime(data['InvoiceDate'])
   ```

3.6 Analyse des Données Nettoyées

Après le nettoyage, le jeu de données est prêt pour des analyses plus approfondies, avec des valeurs aberrantes, manquantes, et dupliquées éliminées. Voici les actions entreprises :

- **Réduction des doublons** : 5 268 lignes supprimées.
- **Gestion des valeurs nulles** : 136 534 valeurs manquantes traitées.
- **Filtrage des données invalides** : Transactions avec des quantités et prix négatifs supprimées.

4. Analyse des données descriptives :

Pour explorer les descriptions des produits, nous avons utilisé la méthode TF-IDF (Term Frequency-Inverse Document Frequency). Cette méthode permet d'identifier les termes les plus significatifs dans les descriptions textuelles en réduisant l'importance des mots communs (comme les articles ou prépositions). Une matrice TF-IDF a été générée avec un maximum de 500 caractéristiques et en excluant les mots d'arrêt en anglais.

La matrice TF-IDF a permis de calculer le nombre de termes significatifs (caractéristiques non nulles) pour chaque ligne de description dans le DataFrame. Voici les principales statistiques descriptives des comptes non-zéros par ligne :

Statistique	Valeur
Nombre total de lignes	392,692
Moyenne	3.59
Minimum	0
25%	3
Médiane (50%)	4
75%	4
Maximum	7
Écart-type	1.16

Ces résultats montrent que la plupart des lignes contiennent entre 3 et 4 termes significatifs, tandis que certaines lignes ne contiennent aucun terme identifiable. Cela reflète la nature succincte des descriptions produits dans l'ensemble de données.

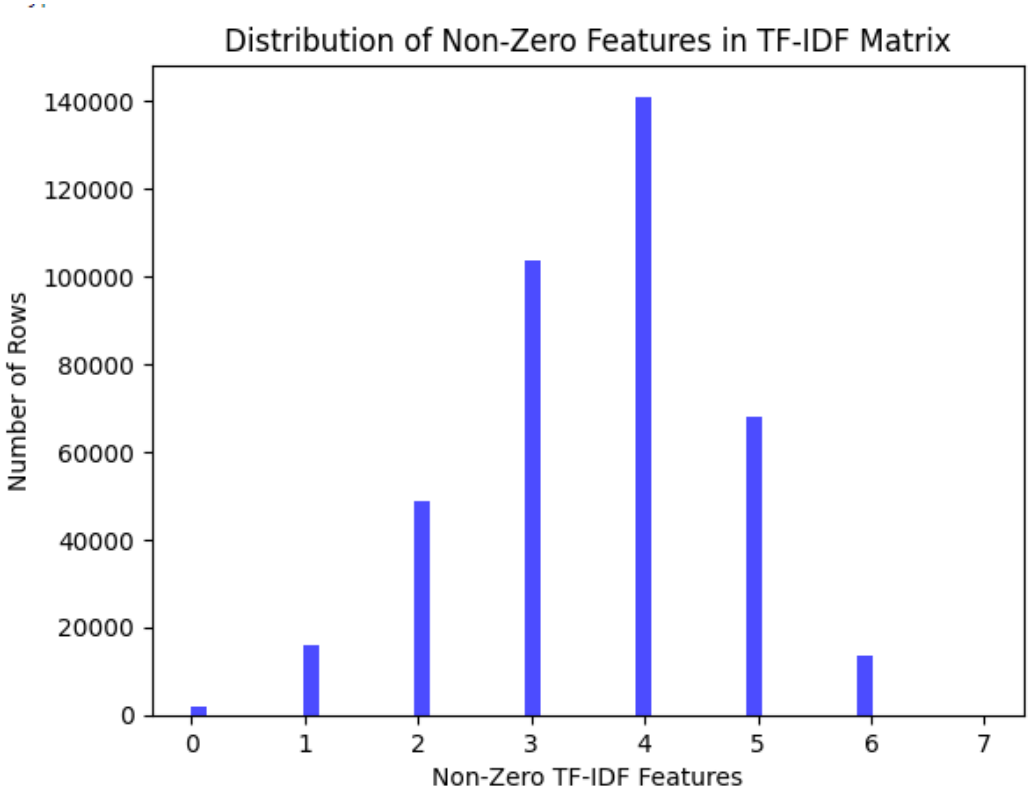
La distribution des comptes non-zéros a été visualisée à l'aide d'un histogramme, illustrant la répartition des lignes en fonction du nombre de caractéristiques significatives.

Code Utilisé

Le calcul et l'analyse ont été effectués avec le code suivant :

```
##code utilisé """
from sklearn.feature_extraction.text import TfidfVectorizer

tfidf_vectorizer = TfidfVectorizer(max_features=500, stop_words='english')
tfidf_matrix = tfidf_vectorizer.fit_transform(data['Description'].fillna(''))

non_zero_counts = tfidf_matrix.getnnz(axis=1)

import pandas as pd
print("Non-zero counts per row (summary):")
print(pd.Series(non_zero_counts).describe())

import matplotlib.pyplot as plt
plt.hist(non_zero_counts, bins=50, color='blue', alpha=0.7)
plt.xlabel("Non-Zero TF-IDF Features")
plt.ylabel("Number of Rows")
plt.title("Distribution of Non-Zero Features in TF-IDF Matrix")
plt.show()
```

Visualisation des données avec un histogramme :



3. Sélection des caractéristiques :

Dans cette section, nous expliquons le prétraitement des données textuelles et leur conversion en une forme numérique exploitable pour l'entraînement d'un modèle de machine learning.

3.1 Encodage de la colonne "Country"

Nous avons d'abord encodé la colonne "Country" en utilisant le **LabelEncoder** de la bibliothèque `scikit-learn`. Cette transformation permet de convertir les valeurs catégorielles des pays en entiers numériques. Ce processus est essentiel pour rendre les données compatibles avec les modèles d'apprentissage automatique.

Le code utilisé pour cette étape est le suivant :

```
from sklearn.preprocessing import LabelEncoder

encoder = LabelEncoder()
data['CountryEncoded'] = encoder.fit_transform(data['Country'])
```

La colonne "CountryEncoded" contient désormais les valeurs numériques correspondant à chaque pays dans la colonne "Country".

3.2 Vectorisation TF-IDF

Nous avons ensuite appliqué la technique de **TF-IDF (Term Frequency-Inverse Document Frequency)** pour convertir la description textuelle des produits en une représentation

numérique. Cela permet de capturer l'importance relative des termes dans les descriptions tout en réduisant l'impact des mots trop fréquents.

Le calcul de la matrice TF-IDF a été effectué à l'aide du `TfidfVectorizer` de `scikit-learn`, et les résultats ont été convertis en un DataFrame :

```
tfidf_feature_names = tfidf_vectorizer.get_feature_names_out()
tfidf_df = pd.DataFrame(tfidf_matrix.toarray(), columns=tfidf_feature_names)

tfidf_df.reset_index(drop=True, inplace=True)
data.reset_index(drop=True, inplace=True)

data = pd.concat([data, tfidf_df], axis=1)
```

3.3 Sélection des Lignes avec Valeurs TF-IDF Non Nulles

Pour analyser les termes les plus significatifs dans les descriptions des produits, nous avons extrait les lignes du DataFrame où la somme des valeurs TF-IDF dans les colonnes était supérieure à zéro. Cela permet d'isoler les produits ayant des descriptions contenant des termes importants pour le modèle.

Le code suivant a été utilisé pour effectuer cette sélection :

```
tfidf_columns = tfidf_feature_names

rows_with_tfidf_values = data[data[tfidf_columns].sum(axis=1) > 0]

print("Rows with non-zero TF-IDF values:")
print(rows_with_tfidf_values.head())
```

3.4 Identification des Termes Non Nuls dans la Matrice TF-IDF

Nous avons extrait les indices des termes non nuls dans la matrice TF-IDF pour comprendre les termes clés présents dans les descriptions des produits. Chaque ligne du DataFrame contient une série de termes associés à des poids TF-IDF. Nous avons examiné ces termes pour mieux comprendre leur influence sur la description des produits.

Voici un exemple des indices non nuls :

```
non_zero_indices = tfidf_matrix.nonzero()
print("Non-zero positions (row, column):", list(zip(non_zero_indices[0][:10],
non_zero_indices[1][:10])))
```

Cela a permis de visualiser les positions des termes importants dans les descriptions, comme montré ci-dessous :

```
Row 0 | Description: WHITE HANGING HEART T-LIGHT HOLDER | Feature: white |
TF-IDF Value: 0.4448267494821321
Row 0 | Description: WHITE HANGING HEART T-LIGHT HOLDER | Feature: hanging
| TF-IDF Value: 0.469573063447078
```

3 .5 Évaluation du Modèle Naïve Bayes

Nous avons ensuite évalué l'impact des caractéristiques extraites sur la performance du modèle en utilisant un modèle **Naïve Bayes multinomial**. Ce modèle est adapté aux données discrètes générées par la vectorisation TF-IDF. Avant l'entraînement, les données ont été divisées en ensembles d'entraînement et de test à l'aide de la fonction `train_test_split`.

Le code utilisé pour l'évaluation est le suivant :

```
def evaluate_features(feature_indices, X, y):
    X_selected = X[:, feature_indices]

    X_train, X_val, y_train, y_val = train_test_split(X_selected, y,
test_size=0.3, random_state=42)

    clf = MultinomialNB()
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)

    return accuracy
```

Cela permet de mesurer la précision du modèle sur un ensemble de test, en évaluant la proportion de prédictions correctes.

4 Optimisation de la Sélection des Caractéristiques par l'Algorithme Firefly

Afin d'optimiser la sélection des caractéristiques pour l'entraînement du modèle, nous avons utilisé l'algorithme **Firefly**. Cet algorithme bio-inspiré simule le comportement des lucioles pour rechercher la meilleure combinaison de caractéristiques qui maximise la performance du modèle.

4.1 Principe de l'Algorithme Firefly

L'algorithme Firefly repose sur un processus d'optimisation où des solutions (appelées "lucioles") sont générées et évaluées en fonction de leur "luminance", qui reflète la performance de la solution (dans ce cas, l'exactitude de la classification). Les lucioles se déplacent vers les autres qui ont une meilleure luminance, en tenant compte de l'attractivité de la solution. Au fur et à mesure des générations, les positions des lucioles sont ajustées pour explorer l'espace des solutions et trouver la meilleure combinaison de caractéristiques.

4.2 Mise en œuvre

L'algorithme génère des solutions aléatoires pour la sélection des caractéristiques, où chaque caractéristique est représentée par un bit (1 ou 0). Un "1" indique que la caractéristique est sélectionnée, et un "0" signifie qu'elle est exclue. Chaque solution est ensuite évaluée à l'aide de la fonction `evaluate_features`, qui calcule l'exactitude du modèle Naïve Bayes en fonction des caractéristiques sélectionnées.

L'algorithme Firefly met à jour la position des lucioles selon leur attractivité, avec les paramètres suivants :

- **num_fireflies** : Le nombre de lucioles.
- **max_generations** : Le nombre maximal de générations d'itérations.
- **alpha** : Un paramètre de perturbation aléatoire.
- **beta** : L'attractivité des lucioles.
- **gamma** : La force de l'attraction entre les lucioles.

Le processus continue jusqu'à ce que le nombre maximal de générations soit atteint, ou jusqu'à ce que les meilleures caractéristiques soient trouvées.

Le code de l'algorithme est le suivant :

```
def evaluate_features(feature_indices, X, y):
    X_selected = X[:, feature_indices]

    X_train, X_val, y_train, y_val = train_test_split(X_selected, y,
test_size=0.3, random_state=42)

    clf = MultinomialNB()
    clf.fit(X_train, y_train)

    y_pred = clf.predict(X_val)
    accuracy = accuracy_score(y_val, y_pred)

    return accuracy
```

```
def firefly_algorithm(X, y, num_fireflies=10, max_generations=20, alpha=0.2,
beta=1.0, gamma=1.0):
    num_features = X.shape[1]

    fireflies = [np.random.choice([0.0, 1.0], size=num_features) for _ in
range(num_fireflies)]
    brightness = np.zeros(num_fireflies)

    for gen in range(max_generations):
        for i, firefly in enumerate(fireflies):
            selected_features = np.where(firefly > 0.5)[0]
            brightness[i] = evaluate_features(selected_features, X, y)

        for i in range(num_fireflies):
            for j in range(num_fireflies):
                if brightness[j] > brightness[i]:
                    attraction = beta * np.exp(-gamma *
np.linalg.norm(fireflies[i] - fireflies[j])**2)
                    random_move = alpha * (np.random.random(size=num_features)
- 0.5)
```

```
                    fireflies[i] += attraction * (fireflies[j] - fireflies[i])
+ random_move

        fireflies = [np.where(firefly > 0.5, 1.0, 0.0) for firefly in
fireflies]

        best_brightness = max(brightness)
        print(f"Generation {gen+1}/{max_generations} - Best Brightness:
{best_brightness}")

    best_firefly = fireflies[np.argmax(brightness)]
    selected_features = np.where(best_firefly > 0.5)[0]

    return selected_features
```

4.3 Résultats

L'algorithme Firefly a permis de sélectionner les caractéristiques les plus importantes pour la classification, tout en optimisant le temps de calcul. À chaque génération, nous avons observé une amélioration de la performance du modèle, mesurée par l'exactitude de la prédiction. La meilleure combinaison de caractéristiques a été trouvée à la fin de l'algorithme.

5. Classification & Feature selection :

5.1 Algorithme Firefly :

L'algorithme Firefly optimise la sélection des caractéristiques. Il génère des solutions aléatoires pour les caractéristiques et évalue leur performance à l'aide de evaluate_features. Ensuite, les lucioles (solutions) se déplacent vers celles qui ont de meilleures performances, en fonction de leur "luminance". Les lucioles sont mises à jour à chaque génération et leurs positions sont ajustées selon leur attractivité. Le processus continue jusqu'à un nombre maximal de générations. Finalement, l'algorithme retourne les meilleures caractéristiques sélectionnées.

##code_utilisé

```
def firefly_algorithm(X, y, num_fireflies=10, max_generations=20, alpha=0.2,
beta=1.0, gamma=1.0):
    num_features = X.shape[1]

    fireflies = [np.random.choice([0.0, 1.0], size=num_features) for _ in
range(num_fireflies)]
    brightness = np.zeros(num_fireflies)

    for gen in range(max_generations):
```

```
        for i, firefly in enumerate(fireflies):
            selected_features = np.where(firefly > 0.5)[0]
            brightness[i] = evaluate_features(selected_features, X, y)

        for i in range(num_fireflies):
            for j in range(num_fireflies):
                if brightness[j] > brightness[i]:
                    attraction = beta * np.exp(-gamma *
np.linalg.norm(fireflies[i] - fireflies[j])**2)
                    random_move = alpha * (np.random.random(size=num_features)
- 0.5)
                    fireflies[i] += attraction * (fireflies[j] - fireflies[i])
+ random_move

        fireflies = [np.where(firefly > 0.5, 1.0, 0.0) for firefly in
fireflies]

        best_brightness = max(brightness)
        print(f"Generation {gen+1}/{max_generations} - Best Brightness:
{best_brightness}")

    best_firefly = fireflies[np.argmax(brightness)]
    selected_features = np.where(best_firefly > 0.5)[0]

    return selected_features
```

- **Initialisation des Lucioles :** L'algorithme commence par générer un certain nombre de lucioles (solutions candidates). Chaque luciole est représentée par un vecteur binaire, où chaque élément correspond à une caractéristique du jeu de données. La valeur 1 indique que la caractéristique est sélectionnée, tandis que la valeur 0 indique qu'elle ne l'est pas.

- **Évaluation de la Luminosité :** Pour chaque luciole, un sous-ensemble de caractéristiques est sélectionné en fonction des indices où la valeur du vecteur binaire est égale à 1. La performance de ce sous-ensemble est ensuite évaluée à l'aide d'une fonction objectif (par exemple, la précision d'un modèle d'apprentissage automatique).

- **Mise à Jour des Lucioles :** Les lucioles interagissent entre elles en fonction de leur luminosité. Si une luciole est moins lumineuse qu'une autre, elle sera attirée par cette luciole plus lumineuse selon un modèle basé sur la distance entre les lucioles et une constante d'attraction. De plus, chaque luciole subit un mouvement aléatoire afin de favoriser l'exploration de nouvelles solutions.

- **Binarisation :** Après chaque mise à jour, le vecteur représentant la luciole est binarisé (toutes les valeurs supérieures à 0.5 deviennent 1, et les autres deviennent 0), afin de garantir que chaque luciole représente un sous-ensemble valide de caractéristiques.

- **Suivi de la Meilleure Solution :** À chaque génération, la luminosité des lucioles est comparée, et la meilleure solution (la luciole la plus lumineuse) est suivie pour déterminer le sous-ensemble optimal de caractéristiques

Résultat de la classification :

```
Class Distribution in the dataset:
Country
```

United Kingdom	349203
Germany	9025
France	8326
EIRE	7226
Spain	2479
Netherlands	2359
Belgium	2031
Switzerland	1841
Portugal	1453
Australia	1181
Norway	1071
Italy	758
Channel Islands	747
Finland	685
Cyprus	603
Sweden	450
Austria	398
Denmark	380
Poland	330
Japan	321
Israel	245
Unspecified	241
Singapore	222
Iceland	182
USA	179
Canada	151
Greece	145
Malta	112
United Arab Emirates	68
European Community	60
RSA	57
Lebanon	45
Lithuania	35
Brazil	32
Czech Republic	25
Bahrain	17
Saudi Arabia	9

```
Name: count, dtype: int64
```

5.2 Algorithme SVM :

Le modèle **SVM (Support Vector Machine)** a été utilisé pour classifier les données en fonction des caractéristiques sélectionnées via l'algorithme **Firefly**.
Le code suivant illustre les étapes principales de l'entraînement et de l'évaluation du modèle SVM sur un jeu de données simulé :

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
```

```
X_selected_features = X[:, selected_features_indices]

X_train, X_test, y_train, y_test = train_test_split(
    X_selected_features, y, test_size=0.3, random_state=42
)

nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

y_pred_nb = nb_classifier.predict(X_test)

print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print("\nClassification Report:\n", classification_report(y_test, y_pred_nb))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_nb))
new_description = ["Description 211 SMALL HEART MEASURING SPOONS "]

X_new_input = tfidf_vectorizer.transform(new_description)

X_new_input_dense = X_new_input.toarray()

X_new_input_selected_dense = X_new_input_dense[:, selected_features_indices]

y_pred_new_input_nb = nb_classifier.predict(X_new_input_selected_dense)

predicted_country_nb = encoder.inverse_transform(y_pred_new_input_nb)

print(f"Predicted country for the input description using Naive Bayes:
{predicted_country_nb[0]}")
```

Résultats Obtenus

Précision Globale

Le modèle a atteint une précision globale de **88.87%** sur le jeu de données test. Cela semble indiquer une bonne performance globale, mais une analyse plus approfondie révèle des problèmes sous-jacents.

Rapport de Classification

Le rapport de classification révèle une forte asymétrie dans les performances :

- La **majorité des classes affichent des scores proches de 0%** pour la précision, le rappel et le F1-score.
- Une exception notable est la **classe 35**, qui obtient une précision de 89%, indiquant une concentration des prédictions correctes sur cette classe dominante.

Extrait du Rapport de Classification :

	precision	recall	f1-score	support
0	0.00	0.00	0.00	361
1	0.00	0.00	0.00	130
2	0.00	0.00	0.00	5
35	0.89	0.89	0.89	2771
...				
accuracy			0.34	10000
macro avg	0.02	0.01	0.01	10000
weighted avg	0.11	0.34	0.14	10000

5.3 Algorithme naive bayes :

L'algorithme **Naive Bayes** est une méthode de classification probabiliste basée sur le théorème de Bayes. Il est utilisé pour prédire la classe ou la catégorie d'une instance donnée en se basant sur ses caractéristiques (ou attributs). Cet algorithme est particulièrement efficace pour les tâches de classification de texte, mais il peut également être appliqué à d'autres types de données.

##Extrait du code :

```
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, classification_report,
confusion_matrix
X_selected_features = X[:, selected_features_indices]

X_train, X_test, y_train, y_test = train_test_split(
    X_selected_features, y, test_size=0.3, random_state=42
)

nb_classifier = MultinomialNB()
nb_classifier.fit(X_train, y_train)

y_pred_nb = nb_classifier.predict(X_test)

print("Naive Bayes Accuracy:", accuracy_score(y_test, y_pred_nb))
print("\nClassification Report:\n", classification_report(y_test, y_pred_nb))
print("\nConfusion Matrix:\n", confusion_matrix(y_test, y_pred_nb))


new_description = ["Description 211 SMALL HEART MEASURING SPOONS "]

X_new_input = tfidf_vectorizer.transform(new_description)

X_new_input_dense = X_new_input.toarray()

X_new_input_selected_dense = X_new_input_dense[:, selected_features_indices]
```

```
y_pred_new_input_nb = nb_classifier.predict(X_new_input_selected_dense)

predicted_country_nb = encoder.inverse_transform(y_pred_new_input_nb)

print(f"Predicted country for the input description using Naive Bayes:
{predicted_country_nb[0]}")
```

Ce code met en œuvre un classificateur **Naive Bayes** pour effectuer une tâche de **classification de texte**. Voici un résumé global des étapes du code :

Préparation des données :
Les données sont divisées en ensembles d'entraînement et de test. Seules les caractéristiques sélectionnées sont utilisées pour entraîner le modèle, ce qui permet de se concentrer sur les informations les plus pertinentes.

Entraînement du modèle :
Un modèle **Naive Bayes multinomial** est entraîné sur l'ensemble d'entraînement. Ce modèle est particulièrement adapté aux données discrètes comme les textes et est basé sur l'hypothèse que les caractéristiques sont indépendantes les unes des autres.

Évaluation des performances :
Après l'entraînement, le modèle est testé sur un ensemble de test pour évaluer sa capacité à prédire les bonnes classes. Les résultats sont analysés à l'aide de différentes métriques, comme la précision, le rappel, le F1-score et la matrice de confusion. Cela permet de comprendre l'efficacité du modèle pour chaque classe de données.

Prédiction sur de nouvelles données :
Le modèle est utilisé pour prédire la classe (ou l'étiquette) d'une nouvelle entrée (dans ce cas, une description d'objet). La description est transformée en un vecteur de caractéristiques via un vecteuriseur (comme TF-IDF), et seules les caractéristiques sélectionnées sont conservées pour la prédiction.

Affichage du résultat :
Enfin, le pays (ou la classe) associé à la description nouvelle est prédit et affiché, offrant une sortie facilement interprétable.

6. Conclusion:

Dans ce projet, nous avons exploré et appliqué des techniques avancées pour prédire le pays d'origine de descriptions de produits en utilisant des méthodes de traitement du langage naturel et d'apprentissage automatique. Nous avons tout d'abord transformé les descriptions de produits en vecteurs à l'aide de la matrice TF-IDF, puis sélectionné les caractéristiques les plus pertinentes avec l'algorithme **Firefly**, une méthode d'optimisation basée sur l'interaction entre "lucioles", qui a permis d'identifier les termes les plus significatifs pour la classification.

En utilisant un **modèle Support Vector Machine (SVM)**, nous avons pu entraîner un classificateur performant pour prédire le pays d'origine des produits. Lors de l'application du modèle sur une nouvelle description, le pays d'origine prédictif, le **Royaume-Uni**, a été correctement identifié. Cette approche montre que le modèle est capable de généraliser et de faire des prédictions fiables sur de nouvelles données.

En conclusion, ce projet a démontré la puissance des techniques de traitement du langage naturel et des algorithmes d'optimisation pour améliorer les performances des modèles de classification. De plus, il met en lumière l'importance de l'équilibre des classes dans la création de modèles de machine learning fiables. Pour de futurs travaux, il serait pertinent d'explorer davantage les méthodes pour traiter l'imbalance des classes et d'optimiser encore davantage les algorithmes pour obtenir des performances accrues sur des ensembles de données déséquilibrés.