TP2 - Feature Engineering

In [2]:
```
import os
import pandas as pd
import numpy as np
import seaborn as sns
from scipy.stats import chi2_contingency
import matplotlib.pyplot as plt
import nltk
from nltk.corpus import stopwords
import time

import re
import string
from sklearn.model_selection import train_test_split
from sklearn.feature_extraction.text import TfidfVectorizer, CountVectorizer
from sklearn.linear_model import LogisticRegression
from sklearn.ensemble import RandomForestClassifier
from sklearn.svm import SVC
from sklearn.metrics import f1_score, accuracy_score, classification_report
from sklearn.neighbors import KNeighborsClassifier
```

Étape 1 : On considère les fichiers du répertoire bbcsport

- Indiquer les points marquants l'exploration.
- Pour chaque observation, indiquer l’opération à effectuer qui serait la plus appropriée.

In [4]:
```
start_time = time.time()

# Définir le chemin du répertoire principal
base_dir = "c:/tmp/bbcsport"

# Initialiser une liste pour stocker les données
data = []

# Parcourir chaque sous-répertoire
```

```
for category in os.listdir(base_dir):
    category_path = os.path.join(base_dir, category)

    # Vérifier si c'est bien un répertoire
    if os.path.isdir(category_path):
        # Lire tous les fichiers dans le sous-répertoire
        for file_name in os.listdir(category_path):
            file_path = os.path.join(category_path, file_name)

            # Lire le contenu du fichier texte
            with open(file_path, "r", encoding="utf-8", errors="ignore") as file:
                text = file.read()

            # Ajouter aux données
            data.append((text, category))


# Créer le DataFrame
df = pd.DataFrame(data, columns=["texte", "categorie"])

#from sklearn.utils import resample

df.reset_index(drop=True, inplace=True)
#df.rename(columns={"index": "ID"}, inplace=True)

print('Dimensions:',df.shape, '\n')

maxCharacters = 100
pd.set_option("display.max_colwidth", maxCharacters)  # Affiche jusqu'à maxCharacters caractères

# Afficher les 20 premières lignes du DataFrame avec retours à la ligne
print('Dataframe original:')
display(df.head(20))

# Mélanger les lignes du DataFrame de façon aléatoire
df = df.sample(frac=1, random_state=42).reset_index(drop=True)
print(f"Nombre de doublons : {df.duplicated().sum()}")
df = df.drop_duplicates() # Suppression des doublons
print(df['categorie'].value_counts())
print(df.shape)
```

```
Dimensions: (737, 2)

Dataframe original:
```

	texte	categorie
0	Claxton hunting first major medal\n\nBritish hurdler Sarah Claxton is confident she can win her ...	athletics
1	O'Sullivan could run in Worlds\n\nSonia O'Sullivan has indicated that she would like to particip...	athletics
2	Greene sets sights on world title\n\nMaurice Greene aims to wipe out the pain of losing his Olym...	athletics
3	IAAF launches fight against drugs\n\nThe IAAF - athletics' world governing body - has met anti-d...	athletics
4	Dibaba breaks 5,000m world record\n\nEthiopia's Tirunesh Dibaba set a new world record in winnin...	athletics
5	Isinbayeva claims new world best\n\nPole vaulter Yelena Isinbayeva broke her own indoor world re...	athletics
6	O'Sullivan commits to Dublin race\n\nSonia O'Sullivan will seek to regain her title at the Bupa ...	athletics
7	Hansen 'delays return until 2006'\n\nBritish triple jumper Ashia Hansen has ruled out a comeback...	athletics
8	Off-colour Gardener storms to win\n\nBritain's Jason Gardener shook off an upset stomach to win ...	athletics
9	Collins to compete in Birmingham\n\nWorld and Commonwealth 100m champion Kim Collins will compet...	athletics
10	Radcliffe yet to answer GB call\n\nPaula Radcliffe has been granted extra time to decide whether...	athletics
11	Edwards tips Idowu for Euro gold\n\nWorld outdoor triple jump record holder and BBC pundit Jonat...	athletics
12	Kenya lift Chepkemei's suspension\n\nKenya's athletics body has reversed a ban on marathon runne...	athletics
13	McIlroy aiming for Madrid title\n\nNorthern Ireland man James McIlroy is confident he can win hi...	athletics
14	UK Athletics agrees new kit deal\n\nUK Athletics has agreed a new deal with adidas to supply Gre...	athletics
15	Verdict delay for Greek sprinters\n\nGreek athletics' governing body has postponed by two weeks ...	athletics
16	Call for Kenteris to be cleared\n\nKostas Kenteris' lawyer has called for the doping charges aga...	athletics
17	Merritt close to indoor 400m mark\n\nTeenager LaShawn Merritt ran the third fastest indoor 400m ...	athletics
18	London hope over Chepkemei\n\nLondon Marathon organisers are hoping that banned athlete Susan Ch...	athletics
19	Edwards tips Idowu for Euro gold\n\nWorld outdoor triple jump record holder and BBC pundit Jonat...	athletics

```
Nombre de doublons : 10
categorie
football     262
rugby        146
cricket      121
tennis        99
athletics     99
Name: count, dtype: int64
(727, 2)
```

Rééchantillonnage du dataset

In [6]:

```
from sklearn.utils import resample

# Taille cible : on va utiliser la taille de la catégorie majoritaire
target_size = int(df['categorie'].value_counts().max()*4/7)
print('target_size:', target_size)

# Sous-échantillonner 'football'
df_football_resampled = resample(df[df['categorie'] == 'football'],
                                  replace=False,  # Sous-échantillonnage
                                  n_samples=target_size,
                                  random_state=42)

# Conserver les autres catégories sans modification
df_other = df[df['categorie'] != 'football']

# Concaténer les deux ensembles pour obtenir le DataFrame final
df_balanced = pd.concat([df_football_resampled, df_other])

print("\nDataframe après équilibrage:")
# Éliminer les doublons
df = df_balanced.copy()
df = df.drop_duplicates()
#df.dropna(inplace=True) # supprime toutes les lignes où au moins une colonne a une valeur NaN.

print(df['categorie'].value_counts())  # Vérifier le nouvel équilibre des classes
print('catégories:',df['categorie'].unique())  # Vérifier les categoris uniques
print(f"Nombre de lignes dupliquées : {df.duplicated().sum()}") # Verifier s'il y'a des doublons
print('Dimensions dataset rebalancé:', df.shape)
```

```
target_size: 149

Dataframe après équilibrage:
categorie
football     149
rugby        146
cricket      121
tennis        99
athletics     99
Name: count, dtype: int64
catégories: ['football' 'tennis' 'athletics' 'rugby' 'cricket']
Nombre de lignes dupliquées : 0
Dimensions dataset rebalancé: (614, 2)
```

Extraction des mots-clés par catégorie et appliquer des prédicteurs de base

In [8]:

```
#!pip install spacy
#!python -m spacy download en_core_web_sm
import spacy

# Charger le modèle NLP anglais pour l'analyse linguistique (verbes, noms, etc.)
nlp = spacy.load("en_core_web_sm")

# Définir les catégories sportives
categories = ['tennis', 'athletics', 'rugby', 'cricket', 'football']

# Définir les mots-clés spécifiques à chaque sport
keywords = {
    "tennis": ["serve", "racket", "wimbledon", "grand slam", "ace", "match", "court"],
    "athletics": ["100m", "hurdles", "relay", "track", "marathon", "jump", "sprint"],
    "rugby": ["scrum", "try", "tackle", "ruck", "conversion", "union", "league"],
    "cricket": ["innings", "bowler", "batsman", "wicket", "yorker", "over", "stump"],
    "football": ["goal", "penalty", "striker", "midfield", "defender", "league", "cup"]
}

# Fusionner tous les mots-clés
all_keywords = list(set([word for sublist in keywords.values() for word in sublist]))

def extract_features(text):
    doc = nlp(text.lower())  # Convertir en minuscule pour l'uniformité
```

```
    # Prédicteurs structurels
    num_chars = len(text)
    num_words = len(text.split())
    num_sentences = text.count('.') + text.count('!') + text.count('?')
    avg_word_length = np.mean([len(word) for word in text.split()]) if num_words > 0 else 0
    avg_sentence_length = num_words / num_sentences if num_sentences > 0 else 0
    num_short_words = sum(1 for word in text.split() if len(word) <= 3)
    num_long_words = sum(1 for word in text.split() if len(word) >= 7)

    # Charger la liste des stopwords en anglais
    stop_words = set(stopwords.words("english"))

    # Fonction pour compter les stopwords dans un texte
    words = text.split()  # Découper en mots
    num_stop_words = sum(1 for word in words if word.lower() in stop_words)

    # Mots-clés spécifiques aux sports
    keyword_counts = [text.lower().count(word) for word in all_keywords]

    # Fréquences lexicales et linguistiques
    num_unique_words = len(set(text.split()))
    lexical_richness = num_unique_words / num_words if num_words > 0 else 0
    num_verbs = sum(1 for token in doc if token.pos_ == "VERB")
    num_nouns = sum(1 for token in doc if token.pos_ == "NOUN")
    num_digits = sum(1 for char in text if char.isdigit())
    num_uppercase = sum(1 for char in text if char.isupper())

    # NER: Compter les entités pertinentes
    num_athletes = sum(1 for ent in doc.ents if ent.label_ in ["PERSON"])  # Compte les noms d'athlètes (ex
    num_competitions = sum(1 for ent in doc.ents if ent.label_ in ["EVENT"]) # Compte les compétitions (ex
    num_locations = sum(1 for ent in doc.ents if ent.label_ in ["GPE", "LOC"]) # Compte les lieux géograph
    num_proper_nouns = sum(1 for token in doc if token.pos_ == "PROPN") # Compte les noms propres (ex: "Mes

    # Caractères spécifiques et ponctuation
    num_punctuation = sum(1 for char in text if char in ".,!?")
    num_hashtags = text.count("#")
    num_mentions = text.count("@")

    return [
        num_chars, num_words, num_sentences, num_stop_words, avg_word_length, avg_sentence_length,
        num_short_words, num_long_words, num_unique_words, lexical_richness,
        num_verbs, num_nouns, num_digits, num_uppercase, num_punctuation,
```

```
        num_hashtags, num_mentions
    ] + keyword_counts

# Appliquer la fonction à la colonne 'texte'
df_features = df['texte'].apply(lambda x: extract_features(str(x)))
feature_names = [
    "num_chars", "num_words", "num_sentences", "num_stop_words", "avg_word_length", "avg_sentence_length",
    "num_short_words", "num_long_words", "num_unique_words", "lexical_richness",
    "num_verbs", "num_nouns", "num_digits", "num_uppercase", "num_punctuation",
    "num_hashtags", "num_mentions"
] + all_keywords  # Ajouter les mots-clés aux colonnes

df_features_init = pd.DataFrame(df_features.tolist(), columns=feature_names)

# Ajouter TF-IDF des mots les plus discriminants
vectorizer = TfidfVectorizer(stop_words='english', max_features=10)  # Garder les 10 meilleurs mots
tfidf_matrix = vectorizer.fit_transform(df['texte'])
df_tfidf = pd.DataFrame(tfidf_matrix.toarray(), columns=vectorizer.get_feature_names_out())

# Concaténer toutes les features
df = df.reset_index(drop=True)
df_features_init = df_features_init.reset_index(drop=True)
df_features_base = pd.concat([df_features_init, df_tfidf], axis=1)
print(df_features_base.shape)

df = df.reset_index(drop=True)
df_features_base = df_features_base.reset_index(drop=True)
df = pd.concat([df, df_features_base], axis=1)
display(df.head())  # Afficher les premières lignes du DataFrame final
print(df['categorie'].shape)
print(df.shape)
```

```
(614, 61)
```

	texte	categorie	num_chars	num_words	num_sentences	num_stop_words	avg_word_length	avg_sente
0	Mourinho sends out warning shot\n\nChelsea boss Jose Mourinho believes his team's Carling Cup wi...	football	1308	247	12	123	4.283401	
1	Kenyon denies Robben Barca return\n\nChelsea chief executive Peter Kenyon has played down report...	football	1218	221	12	101	4.502262	
2	Strachan turns down Pompey\n\nFormer Southampton manager Gordon Strachan has rejected the chance...	football	1463	241	12	91	5.058091	
3	Dundee Utd 4-1 Aberdeen\n\nDundee United eased into the semi-final of the Scottish Cup with an e...	football	2611	451	27	160	4.760532	
4	Mansfield 0-1 Leyton Orient\n\nAn second-half goal from Andy Scott condemned Mansfield to a nint...	football	820	126	10	32	5.452381	

5 rows × 63 columns

```
(614,)
(614, 63)
```

Encodage de la cible

In [10]:
```
from sklearn.preprocessing import LabelEncoder

# Définir les labels de sport (catégories)
categories = ['tennis', 'athletics', 'rugby', 'cricket', 'football']
le = LabelEncoder()  # Pour encoder les catégories de sport sous forme numérique
df['category_encoded'] = le.fit_transform(df['categorie'])  # 'category' est la colonne avec les labels de

# Séparer les features et les labels
X = df_features_base  # Les features extraites
y = df['category_encoded']  # Les labels encodés

print(X.shape)
print(y.shape)
print(df['categorie'].shape)
```

```
(614, 61)
(614,)
(614,)
```

Profilage

In [12]:
```
import sweetviz as sv
df_profiled = pd.concat([df['category_encoded'], df_features_base], axis=1)
print(df_profiled.columns.tolist())  # Vérifie les noms exacts des colonnes
#df_profiled.columns = df_profiled.columns.str.strip()  # Supprime les espaces invisibles
#df_profiled = df_profiled.loc[:, ~df_profiled.columns.duplicated()]  # Supprime les colonnes en double
report = sv.analyze(df_profiled)
report.show_html("profile_sortie.html")
print(df_profiled.shape)
```

```
['category_encoded', 'num_chars', 'num_words', 'num_sentences', 'num_stop_words', 'avg_word_length', 'avg_s
entence_length', 'num_short_words', 'num_long_words', 'num_unique_words', 'lexical_richness', 'num_verbs',
'num_nouns', 'num_digits', 'num_uppercase', 'num_punctuation', 'num_hashtags', 'num_mentions', 'scrum', 'de
fender', 'wicket', 'bowler', 'league', 'stump', 'ace', '100m', 'conversion', 'match', 'over', 'jump', 'rela
y', 'ruck', 'yorker', 'penalty', 'striker', 'wimbledon', 'track', 'serve', 'racket', 'goal', 'midfield', 'u
nion', 'cup', 'sprint', 'marathon', 'try', 'court', 'innings', 'batsman', 'tackle', 'grand slam', 'hurdle
s', 'england', 'game', 'new', 'play', 'said', 'team', 'time', 'win', 'world', 'year']
                                                   |                                                   | [  0%]   00:00
->…
```

```
Report profile_sortie.html was generated! NOTEBOOK/COLAB USERS: the web browser MAY not pop up, regardless, the report IS saved in your notebook/colab files.
(614, 62)
```

Sélection de Prédicteurs par Random Forest (RF)

In [14]:
```python
from sklearn.ensemble import RandomForestClassifier

# Entraîner le modèle Random Forest
rf = RandomForestClassifier(n_estimators=100, random_state=42)
rf.fit(X, y)

# Récupérer l'importance des features
importances = rf.feature_importances_
sorted_indices = np.argsort(importances)[::-1]  # Tri décroissant
print(len(sorted_indices))

# Sélectionner les 20 meilleures features
feature_names = df_features_base.columns
top_k = 20
best_features = [feature_names[i] for i in sorted_indices[:top_k]]

# Créer un nouveau DataFrame X_rfe avec ces features
X_rfe = X[best_features]

# Affichage des features sélectionnées
print("Top 20 features sélectionnées:", best_features)
```

```
61
Top 20 features sélectionnées: ['england', 'wicket', 'world', 'bowler', 'avg_word_length', 'num_digits', 'striker', 'goal', 'play', 'batsman', 'sprint', 'said', 'avg_sentence_length', 'num_long_words', 'game', 'num_uppercase', 'match', 'wimbledon', 'league', 'year']
```

Etape 3: Nettoyage des données

In [16]:
```python
# Prétraitement du texte
def clean_text(text):
    text = text.lower()
    text = re.sub(f"[{string.punctuation}]", "", text)  # Suppression des ponctuations
    text = re.sub("\d+", "", text)  # Suppression des chiffres
```

```
    return text

df["cleaned_text"] = df["texte"].apply(clean_text)

# Séparation des données X et y
X = df["cleaned_text"]
y = df["categorie"]

# Séparation train/test
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

df_metrics = pd.DataFrame(columns=["Technique", "Modele", "Score F1", "Accuracy"])
```

Étape 4: Utilisation d'une technique d’extraction (par ex: TF–IDF Vectors) et choisir au moins 3 algorithmes de classification. De ce fait, on obtient au moins 3 modèles. On fera la comparaison en se basant la technique d’extraction.

In [18]:

```
# Modèles de classification
models = {
    "Logistic Regression": LogisticRegression(max_iter=10000, solver="lbfgs"),
    "Random Forest": RandomForestClassifier(),
    "SVM": SVC(),
    "KNN": KNeighborsClassifier()
}

# Entraînement et évaluation
def appliquer_model_classification(technique, train_x, test_x):
    results = {}
    for name, model in models.items():
        model.fit(train_x, y_train)
        y_pred = model.predict(test_x)
        f1 = f1_score(y_test, y_pred, average='weighted')
        accuracy = accuracy_score(y_test, y_pred)
        results[name] = {"F1-score": f1, "Accuracy": accuracy}
        df_metrics.loc[len(df_metrics)] = [technique, name, f1, accuracy]
        #print(technique, ',  ', name)
        #print(classification_report(y_test, y_pred))
```

```
    # Affichage des résultats
    print("\nComparaison des scores F1 et Accuracy:")
    for model, scores in results.items():
        print(f"{model}: F1-score = {scores['F1-score']:.4f}, Accuracy = {scores['Accuracy']:.4f}")
```

In [19]:
```
from collections import Counter
from scipy.sparse import hstack

all_words = " ".join(df["cleaned_text"]).split()
word_freq = Counter(all_words)
most_common_words = [word for word, _ in word_freq.most_common(2000)]  # Top 2000 mots
print('most common words:', most_common_words[:100]) #  Affiche les 100 premiers éléments de la liste.
techniques = {
    "TfidfVectorizer": TfidfVectorizer(),
    "Bag of Words": CountVectorizer(),
    "Word Embeddings": None,
    "NLP based features": None,
    "Prédicteurs de base": CountVectorizer(vocabulary=most_common_words),
    "Prédicteurs de base sélectionnés": None
}
```

```
most common words: ['the', 'to', 'a', 'in', 'and', 'of', 'for', 'on', 'he', 'was', 'i', 'but', 'is', 'wit
h', 'his', 'it', 'at', 'that', 'have', 'be', 'said', 'has', 'will', 'we', 'as', 'from', 'not', 'after', 'b
y', 'had', 'their', 'an', 'are', 'they', 'who', 'been', 'first', 'out', 'this', 'england', 'when', 'over',
'against', 'game', 'were', 'two', 'win', 'all', 'there', 'm', 'last', 'up', 'world', 'would', 'one', 'if',
'its', 'you', 'play', 'new', 'players', 'time', 'before', 'back', 'team', 'just', 'my', 'also', 'can', 'sec
ond', 'three', 'she', 'which', 'her', 'off', 'match', 'only', 'him', 'now', 'cup', 'very', 'side', 'six',
'test', 'into', 'more', 'so', 'good', 'set', 'about', 'well', 'could', 'then', 'year', 'final', 'me', 'wale
s', 'coach', 'four', 'them']
```

In [20]:
```
from gensim.models import Word2Vec
from gensim.utils import simple_preprocess
from sklearn.preprocessing import StandardScaler

# Fonction pour convertir les textes en vecteurs de mots
def document_vector(model, doc):
    return np.mean([model.wv[word] for word in doc if word in model.wv] or [np.zeros(embedding_dim)], axis=


for technique, model in techniques.items():
    if technique == "Word Embeddings":
        X_train_tokens = [simple_preprocess(text) for text in X_train]
```

```
        X_test_tokens = [simple_preprocess(text) for text in X_test]

        # Entraînement du modèle Word2Vec
        embedding_dim = 100
        word2vec_model = Word2Vec(sentences=X_train_tokens, vector_size=embedding_dim, window=5, min_count=

        train_x = np.array([document_vector(word2vec_model, doc) for doc in X_train_tokens])
        test_x = np.array([document_vector(word2vec_model, doc) for doc in X_test_tokens])


    elif technique == "NLP based features":
        continue


    elif technique == "Prédicteurs de base sélectionnés":
        technique == "Prédicteurs de base sélectionnés"
        print(50*"=")
        print(f"Technique: {technique}")

        X_base = X_rfe

        print(f"Shape:",X_base.shape)

        scaler = StandardScaler()
        X_scaled = scaler.fit_transform(X_base)
        X = X_scaled
        y = df['category_encoded']  # Les labels encodés

        train_x, test_x, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

    elif model is None:
        continue

    else:
        print(50*"=")
        print(f"Technique: {technique}")
        train_x = model.fit_transform(X_train)
        test_x = model.transform(X_test)

    appliquer_model_classification(technique, train_x, test_x)
    print("\n")
```

```
==================================================
Technique: TfidfVectorizer

Comparaison des scores F1 et Accuracy:
Logistic Regression: F1-score = 0.9838, Accuracy = 0.9837
Random Forest: F1-score = 0.9838, Accuracy = 0.9837
SVM: F1-score = 0.9758, Accuracy = 0.9756
KNN: F1-score = 0.9360, Accuracy = 0.9350


==================================================
Technique: Bag of Words

Comparaison des scores F1 et Accuracy:
Logistic Regression: F1-score = 0.9755, Accuracy = 0.9756
Random Forest: F1-score = 0.9594, Accuracy = 0.9593
SVM: F1-score = 0.8724, Accuracy = 0.8699
KNN: F1-score = 0.6497, Accuracy = 0.6504



Comparaison des scores F1 et Accuracy:
Logistic Regression: F1-score = 0.3408, Accuracy = 0.3740
Random Forest: F1-score = 0.5007, Accuracy = 0.5041
SVM: F1-score = 0.1065, Accuracy = 0.2439
KNN: F1-score = 0.3584, Accuracy = 0.3577


==================================================
Technique: Prédicteurs de base

Comparaison des scores F1 et Accuracy:
Logistic Regression: F1-score = 0.9757, Accuracy = 0.9756
Random Forest: F1-score = 0.9837, Accuracy = 0.9837
SVM: F1-score = 0.8724, Accuracy = 0.8699
KNN: F1-score = 0.6818, Accuracy = 0.6829


==================================================
Technique: Prédicteurs de base sélectionnés
Shape: (614, 20)
```

```
Comparaison des scores F1 et Accuracy:
Logistic Regression: F1-score = 0.7531, Accuracy = 0.7561
Random Forest: F1-score = 0.7612, Accuracy = 0.7642
SVM: F1-score = 0.7316, Accuracy = 0.7317
KNN: F1-score = 0.6723, Accuracy = 0.6748
```

In [21]:
```
df_pivot = df_metrics.pivot_table(index='Technique', columns='Modele', values=['Score F1', 'Accuracy'])
df_pivot
```

Out[21]:

	Accuracy				Score F1			
Modele	**KNN**	**Logistic Regression**	**Random Forest**	**SVM**	**KNN**	**Logistic Regression**	**Random Forest**	**SVM**
Technique								
Bag of Words	0.650407	0.975610	0.959350	0.869919	0.649696	0.975526	0.959421	0.872388
Prédicteurs de base	0.682927	0.975610	0.983740	0.869919	0.681818	0.975719	0.983682	0.872388
Prédicteurs de base sélectionnés	0.674797	0.756098	0.764228	0.731707	0.672327	0.753115	0.761154	0.731589
TfidfVectorizer	0.934959	0.983740	0.983740	0.975610	0.935956	0.983752	0.983785	0.975833
Word Embeddings	0.357724	0.373984	0.504065	0.243902	0.358414	0.340771	0.500697	0.106522

In [22]:
```
end_time = time.time()

# Calcul du temps écoulé
execution_time = end_time - start_time

print(f"Temps d'exécution: {execution_time:.6f} secondes")
```

```
Temps d'exécution: 95.571648 secondes
```

In [ ]:

In [ ]: