In [1]:
```
!pip install nltk
```

```
Requirement already satisfied: nltk in c:\users\smain\appdata\local\programs\pyth
on\python310\lib\site-packages (3.9.1)
Requirement already satisfied: joblib in c:\users\smain\appdata\local\programs\py
thon\python310\lib\site-packages (from nltk) (1.4.2)
Requirement already satisfied: regex>=2021.8.3 in c:\users\smain\appdata\local\pr
ograms\python\python310\lib\site-packages (from nltk) (2024.11.6)
Requirement already satisfied: click in c:\users\smain\appdata\local\programs\pyt
hon\python310\lib\site-packages (from nltk) (8.1.8)
Requirement already satisfied: tqdm in c:\users\smain\appdata\local\programs\pyth
on\python310\lib\site-packages (from nltk) (4.67.1)
Requirement already satisfied: colorama in c:\users\smain\appdata\roaming\python
\python310\site-packages (from click->nltk) (0.4.6)
WARNING: You are using pip version 21.2.3; however, version 25.0.1 is available.
You should consider upgrading via the 'C:\Users\smain\AppData\Local\Programs\Pyth
on\Python310\python.exe -m pip install --upgrade pip' command.
```

In [1]:
```
import nltk
nltk.download()
```

```
showing info https://raw.githubusercontent.com/nltk/nltk_data/gh-pages/index.xml
```

Out[1]:
```
True
```

In [4]:
```
!pip install gensim
```

```
Requirement already satisfied: gensim in c:\users\smain\appdata\local\programs\py
thon\python310\lib\site-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in c:\users\smain\appdata\local
\programs\python\python310\lib\site-packages (from gensim) (1.26.4)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in c:\users\smain\appdata\loc
al\programs\python\python310\lib\site-packages (from gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in c:\users\smain\appdata\local
\programs\python\python310\lib\site-packages (from gensim) (7.1.0)
Requirement already satisfied: wrapt in c:\users\smain\appdata\local\programs\pyt
hon\python310\lib\site-packages (from smart-open>=1.8.1->gensim) (1.17.2)
WARNING: You are using pip version 21.2.3; however, version 25.0.1 is available.
You should consider upgrading via the 'C:\Users\smain\AppData\Local\Programs\Pyth
on\Python310\python.exe -m pip install --upgrade pip' command.
```

In [5]:
```
!pip install pattern
```

```
Requirement already satisfied: pattern in c:\users\smain\appdata\local\programs\p
ython\python310\lib\site-packages (0.0.1a0)
Requirement already satisfied: seaborn>=0.10.0 in c:\users\smain\appdata\local\pr
ograms\python\python310\lib\site-packages (from pattern) (0.13.2)
Requirement already satisfied: scikit-learn>=1.5.0 in c:\users\smain\appdata\loca
l\programs\python\python310\lib\site-packages (from pattern) (1.6.1)
Requirement already satisfied: numpy>=1.26.4 in c:\users\smain\appdata\local\prog
rams\python\python310\lib\site-packages (from pattern) (1.26.4)
Requirement already satisfied: matplotlib>=3.9.0 in c:\users\smain\appdata\local
\programs\python\python310\lib\site-packages (from pattern) (3.10.0)
Requirement already satisfied: kiwisolver>=1.3.1 in c:\users\smain\appdata\local
\programs\python\python310\lib\site-packages (from matplotlib>=3.9.0->pattern)
(1.4.8)
Requirement already satisfied: pyparsing>=2.3.1 in c:\users\smain\appdata\local\p
rograms\python\python310\lib\site-packages (from matplotlib>=3.9.0->pattern) (3.
2.1)
Requirement already satisfied: packaging>=20.0 in c:\users\smain\appdata\roaming
\python\python310\site-packages (from matplotlib>=3.9.0->pattern) (24.2)
Requirement already satisfied: contourpy>=1.0.1 in c:\users\smain\appdata\local\p
rograms\python\python310\lib\site-packages (from matplotlib>=3.9.0->pattern) (1.
3.1)
Requirement already satisfied: cycler>=0.10 in c:\users\smain\appdata\local\progr
ams\python\python310\lib\site-packages (from matplotlib>=3.9.0->pattern) (0.12.1)
Requirement already satisfied: pillow>=8 in c:\users\smain\appdata\local\programs
\python\python310\lib\site-packages (from matplotlib>=3.9.0->pattern) (11.1.0)
Requirement already satisfied: fonttools>=4.22.0 in c:\users\smain\appdata\local
\programs\python\python310\lib\site-packages (from matplotlib>=3.9.0->pattern)
(4.56.0)
Requirement already satisfied: python-dateutil>=2.7 in c:\users\smain\appdata\roa
ming\python\python310\site-packages (from matplotlib>=3.9.0->pattern) (2.9.0.post
0)
Requirement already satisfied: six>=1.5 in c:\users\smain\appdata\roaming\python
\python310\site-packages (from python-dateutil>=2.7->matplotlib>=3.9.0->pattern)
(1.17.0)
Requirement already satisfied: joblib>=1.2.0 in c:\users\smain\appdata\local\prog
rams\python\python310\lib\site-packages (from scikit-learn>=1.5.0->pattern) (1.4.
2)
Requirement already satisfied: threadpoolctl>=3.1.0 in c:\users\smain\appdata\loc
al\programs\python\python310\lib\site-packages (from scikit-learn>=1.5.0->patter
n) (3.5.0)
Requirement already satisfied: scipy>=1.6.0 in c:\users\smain\appdata\local\progr
ams\python\python310\lib\site-packages (from scikit-learn>=1.5.0->pattern) (1.13.
1)
Requirement already satisfied: pandas>=1.2 in c:\users\smain\appdata\local\progra
ms\python\python310\lib\site-packages (from seaborn>=0.10.0->pattern) (2.2.3)
Requirement already satisfied: tzdata>=2022.7 in c:\users\smain\appdata\local\pro
grams\python\python310\lib\site-packages (from pandas>=1.2->seaborn>=0.10.0->patt
ern) (2025.1)
Requirement already satisfied: pytz>=2020.1 in c:\users\smain\appdata\local\progr
ams\python\python310\lib\site-packages (from pandas>=1.2->seaborn>=0.10.0->patter
n) (2025.1)
```

```
WARNING: You are using pip version 21.2.3; however, version 25.0.1 is available.
You should consider upgrading via the 'C:\Users\smain\AppData\Local\Programs\Pyth
on\Python310\python.exe -m pip install --upgrade pip' command.
```

In [ ]:

```python
from nltk.tokenize import sent_tokenize, \
        word_tokenize, WordPunctTokenizer

# Define input text
input_text = "Do you know how tokenization works? It's actually quite interestin
```

```
# Sentence tokenizer
print("\nSentence tokenizer:")
print(sent_tokenize(input_text))

# Word tokenizer
print("\nWord tokenizer:")
print(word_tokenize(input_text))

# WordPunct tokenizer
print("\nWord punct tokenizer:")
print(WordPunctTokenizer().tokenize(input_text))
```

```
Sentence tokenizer:
['Do you know how tokenization works?', "It's actually quite interesting!", "Le
t's analyze a couple of sentences and figure it out."]

Word tokenizer:
['Do', 'you', 'know', 'how', 'tokenization', 'works', '?', 'It', "'s", 'actuall
y', 'quite', 'interesting', '!', 'Let', "'s", 'analyze', 'a', 'couple', 'of', 'se
ntences', 'and', 'figure', 'it', 'out', '.']

Word punct tokenizer:
['Do', 'you', 'know', 'how', 'tokenization', 'works', '?', 'It', "'", 's', 'actua
lly', 'quite', 'interesting', '!', 'Let', "'", 's', 'analyze', 'a', 'couple', 'o
f', 'sentences', 'and', 'figure', 'it', 'out', '.']
```

Ce code utilise NLTK pour découper un texte en phrases et en mots. Il commence par segmenter le texte en phrases avec sent_tokenize. Ensuite, il divise le texte en mots en tenant compte des apostrophes et de la ponctuation avec word_tokenize. Enfin, il applique WordPunctTokenizer, qui sépare les mots et la ponctuation de manière plus stricte. Ce processus est utilisé en traitement automatique du langage pour analyser et structurer du texte.

In [7]:
```
from nltk.stem.porter import PorterStemmer
from nltk.stem.lancaster import LancasterStemmer
from nltk.stem.snowball import SnowballStemmer

input_words = ['writing', 'calves', 'be', 'branded', 'horse', 'randomize',
        'possibly', 'provision', 'hospital', 'kept', 'scratchy', 'code']

# Create various stemmer objects
porter = PorterStemmer()
lancaster = LancasterStemmer()
snowball = SnowballStemmer('english')

# Create a list of stemmer names for display
stemmer_names = ['PORTER', 'LANCASTER', 'SNOWBALL']
formatted_text = '{:>16}' * (len(stemmer_names) + 1)
print('\n', formatted_text.format('INPUT WORD', *stemmer_names),
        '\n', '='*68)

# Stem each word and display the output
for word in input_words:
    output = [word, porter.stem(word),
            lancaster.stem(word), snowball.stem(word)]
    print(formatted_text.format(*output))
```

```
           INPUT WORD            PORTER       LANCASTER          SNOWBALL
===========================================================================
             writing              write            writ             write
              calves               calv            calv              calv
                  be                 be              be                be
             branded              brand           brand             brand
               horse               hors            hors              hors
           randomize             random          random            random
            possibly            possibl            poss           possibl
           provision             provis          provid            provis
            hospital             hospit          hospit            hospit
                kept               kept            kept              kept
            scratchy           scratchi        scratchy          scratchi
                code               code             cod              code
```

Ce code applique trois algorithmes de stemming (Porter, Lancaster et Snowball) à une liste de mots en anglais. Il commence par créer des objets pour chaque algorithme de stemming, puis définit une liste de mots à traiter. Ensuite, il génère un tableau affichant chaque mot original et ses versions réduites selon chaque algorithme. Le stemming permet de réduire les mots à leur racine pour normaliser le texte en traitement automatique du langage.

In [2]:
```python
from nltk.stem import WordNetLemmatizer

input_words = ['writing', 'calves', 'be', 'branded', 'horse', 'randomize',
        'possibly', 'provision', 'hospital', 'kept', 'scratchy', 'code']

# Create lemmatizer object
lemmatizer = WordNetLemmatizer()

# Create a list of lemmatizer names for display
lemmatizer_names = ['NOUN LEMMATIZER', 'VERB LEMMATIZER']
formatted_text = '{:>24}' * (len(lemmatizer_names) + 1)
print('\n', formatted_text.format('INPUT WORD', *lemmatizer_names),
        '\n', '='*75)

# Lemmatize each word and display the output
for word in input_words:
    output = [word, lemmatizer.lemmatize(word, pos='n'),
           lemmatizer.lemmatize(word, pos='v')]
    print(formatted_text.format(*output))
```

```
                      INPUT WORD         NOUN LEMMATIZER         VERB LEMMATIZER
 ===========================================================================
                   writing                 writing                   write
                    calves                    calf                   calve
                        be                      be                      be
                   branded                 branded                   brand
                     horse                   horse                   horse
                 randomize               randomize               randomize
                  possibly                possibly                possibly
                 provision               provision               provision
                  hospital                hospital                hospital
                      kept                    kept                    keep
                  scratchy                scratchy                scratchy
                      code                    code                    code
```

Ce code applique la lemmatisation à une liste de mots en utilisant WordNetLemmatizer de NLTK. Contrairement au stemming, la lemmatisation ramène un mot à sa forme lexicale correcte en fonction de sa catégorie grammaticale. Le script traite chaque mot sous deux formes : en tant que nom (pos='n') et en tant que verbe (pos='v'). Ensuite, il affiche un tableau comparant chaque mot avec ses versions lemmatisées. Ce processus est essentiel en traitement automatique du langage pour améliorer la précision de l'analyse sémantique.

In [3]:
```
import numpy as np
from nltk.corpus import brown

# Split the input text into chunks, where
# each chunk contains N words
def chunker(input_data, N):
    input_words = input_data.split(' ')
    output = []

    cur_chunk = []
    count = 0
    for word in input_words:
        cur_chunk.append(word)
        count += 1
        if count == N:
            output.append(' '.join(cur_chunk))
            count, cur_chunk = 0, []

    output.append(' '.join(cur_chunk))

    return output

if __name__=='__main__':
    # Read the first 12000 words from the Brown corpus
    input_data = ' '.join(brown.words()[:12000])

    # Define the number of words in each chunk
    chunk_size = 700

    chunks = chunker(input_data, chunk_size)
    print('\nNumber of text chunks =', len(chunks), '\n')
    for i, chunk in enumerate(chunks):
        print('Chunk', i+1, '==>', chunk[:50])
```

```
Number of text chunks = 18

Chunk 1 ==> The Fulton County Grand Jury said Friday an invest
Chunk 2 ==> '' . ( 2 ) Fulton legislators `` work with city of
Chunk 3 ==> . Construction bonds Meanwhile , it was learned th
Chunk 4 ==> , anonymous midnight phone calls and veiled threat
Chunk 5 ==> Harris , Bexar , Tarrant and El Paso would be $451
Chunk 6 ==> set it for public hearing on Feb. 22 . The proposa
Chunk 7 ==> College . He has served as a border patrolman and
Chunk 8 ==> of his staff were doing on the address involved co
Chunk 9 ==> plan alone would boost the base to $5,000 a year a
Chunk 10 ==> nursing homes In the area of `` community health s
Chunk 11 ==> of its Angola policy prove harsh , there has been
Chunk 12 ==> system which will prevent Laos from being used as
Chunk 13 ==> reform in recipient nations . In Laos , the admini
Chunk 14 ==> . He is not interested in being named a full-time
Chunk 15 ==> said , `` to obtain the views of the general publi
Chunk 16 ==> '' . Mr. Reama , far from really being retired , i
Chunk 17 ==> making enforcement of minor offenses more effectiv
Chunk 18 ==> to tell the people where he stands on the tax issu
```

Ce code divise un texte en segments de taille fixe en utilisant un chunking basé sur le nombre de mots. Il commence par extraire les 12 000 premiers mots du corpus Brown de NLTK, puis les segmente en morceaux contenant 700 mots chacun. Chaque chunk est ensuite stocké dans une liste et affiché avec ses 50 premiers caractères pour un aperçu. Ce type de segmentation est utile pour le traitement de texte en lots, notamment en NLP et apprentissage automatique.

In [5]:
```python
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer

# Fonction pour lire le fichier
def read_file(filepath):
    with open(filepath, 'r', encoding='utf-8') as file:
        return file.read()

# Fonction pour diviser le texte en segments
def chunker(text, chunk_size):
    return [text[i:i + chunk_size] for i in range(0, len(text), chunk_size)]

# Charger le texte depuis `data.txt`
file_path = r"C:\Users\smain\OneDrive\Documents\data.txt"  # Ajouter le `r` pour
input_data = read_file(file_path)

# Nombre de mots dans chaque chunk
chunk_size = 800
text_chunks = chunker(input_data, chunk_size)

# Convertir les segments en dictionnaire
chunks = [{'index': i, 'text': chunk} for i, chunk in enumerate(text_chunks)]

# Extraction de la matrice terme-document
count_vectorizer = CountVectorizer(min_df=2, max_df=20)  # Ajustez min_df si bes
document_term_matrix = count_vectorizer.fit_transform([chunk['text'] for chunk i

# Extraction du vocabulaire
vocabulary = np.array(count_vectorizer.get_feature_names_out())
```

```
print("\nVocabulary:\n", vocabulary)

# Générer les noms de chunks
chunk_names = [f'Chunk-{i+1}' for i in range(len(text_chunks))]

# Affichage de la matrice terme-document
print("\nDocument term matrix:")
formatted_text = '{:>12}' * (len(chunk_names) + 1)
print('\n', formatted_text.format('Word', *chunk_names), '\n')

for word, item in zip(vocabulary, document_term_matrix.T):
    output = [word] + [str(freq) for freq in item.data]
    print(formatted_text.format(*output))
```

```
Vocabulary:
 ['and' 'between' 'europe' 'formulate' 'have' 'in' 'mathematics' 'of'
 'that' 'the']

Document term matrix:

         Word     Chunk-1     Chunk-2

         and           4           1
     between           1           1
      europe           1           1
   formulate           1           1
        have           1           1
          in           6           1
 mathematics           1           1
          of           5           3
        that           2           1
         the           9           1
```

Ce code lit un fichier texte (data.txt), le divise en segments de 800 caractères, puis crée une matrice terme-document en utilisant CountVectorizer de sklearn. Il commence par charger le texte, le segmente en chunks, et construit un dictionnaire contenant ces segments. Ensuite, il extrait les termes fréquents (présents dans au moins 2 et au plus 20 segments) et génère un vocabulaire. Enfin, il affiche la matrice terme-document qui représente la fréquence des mots dans chaque chunk. Ce procédé est utile en analyse de texte et NLP pour identifier les termes les plus significatifs d'un document volumineux.

In [8]:

```
import numpy as np
from sklearn.datasets import fetch_20newsgroups
from sklearn.naive_bayes import MultinomialNB
from sklearn.feature_extraction.text import TfidfTransformer, CountVectorizer

# Définition des catégories
category_map = {
    'talk.politics.misc': 'Politics',
    'rec.autos': 'Autos',
    'rec.sport.hockey': 'Hockey',
    'sci.electronics': 'Electronics',
    'sci.med': 'Medicine'
}

print("🚀 Fetching training data...")
training_data = fetch_20newsgroups(subset='train', categories=category_map.keys(
```

```python
print(f"✅ Training data loaded. Number of documents: {len(training_data.data)}

# Vérification si les données sont bien récupérées
if len(training_data.data) == 0:
    print("❌ Aucune donnée récupérée. Vérifiez les catégories !")
    exit()

# Vectorisation avec suppression des stopwords
print("🛠 Vectorizing text data...")
count_vectorizer = CountVectorizer(stop_words='english')
train_tc = count_vectorizer.fit_transform(training_data.data)
print(f"✅ Training data vectorized. Shape: {train_tc.shape}")

# Vérification si la vectorisation fonctionne
if train_tc.shape[0] == 0 or train_tc.shape[1] == 0:
    print("❌ La matrice de compte est vide. Problème de vectorisation !")
    exit()

# Transformation TF-IDF
print("🔄 Transforming with TF-IDF...")
tfidf = TfidfTransformer()
train_tfidf = tfidf.fit_transform(train_tc)
print(f"✅ TF-IDF transformation done. Shape: {train_tfidf.shape}")

# Entraînement du modèle Naïve Bayes
print("🤖 Training Naïve Bayes model...")
classifier = MultinomialNB().fit(train_tfidf, training_data.target)
print("✅ Model trained successfully!")

# Données de test personnalisées
input_data = [
    'You need to be careful with cars when you are driving on slippery roads',
    'A lot of devices can be operated wirelessly',
    'Players need to be careful when they are close to goal posts',
    'Political debates help us understand the perspectives of both sides'
]

# Transformation et prédiction
print("📡 Transforming input data...")
input_tc = count_vectorizer.transform(input_data)
input_tfidf = tfidf.transform(input_tc)

print("🔮 Predicting categories...")
predictions = classifier.predict(input_tfidf)

# Affichage des résultats
print("\n🎯 Predictions:")
for sent, category in zip(input_data, predictions):
    print(f"🔹 Input: {sent}\n   Predicted category: {category_map[training_dat
🚀 Fetching training data...
✅ Training data loaded. Number of documents: 2844
🛠 Vectorizing text data...
✅ Training data vectorized. Shape: (2844, 40018)
🔁 Transforming with TF-IDF...
✅ TF-IDF transformation done. Shape: (2844, 40018)
🤖 Training Naïve Bayes model...
✅ Model trained successfully!
📡 Transforming input data...
🔮 Predicting categories...

🎯 Predictions:
🔹 Input: You need to be careful with cars when you are driving on slippery road
s
   Predicted category: Autos

🔹 Input: A lot of devices can be operated wirelessly
   Predicted category: Electronics

🔹 Input: Players need to be careful when they are close to goal posts
   Predicted category: Hockey

🔹 Input: Political debates help us understand the perspectives of both sides
   Predicted category: Medicine
```

Ce code entraîne un modèle Naïve Bayes multinomial pour classer des textes en fonction de leur contenu, en utilisant le dataset 20 Newsgroups. Il commence par récupérer des articles liés à cinq catégories (politique, automobile, hockey, électronique et médecine). Ensuite, il vectorise les textes en appliquant une transformation TF-IDF pour pondérer les mots les plus significatifs. Une fois le modèle entraîné, il est utilisé pour prédire la catégorie de nouveaux textes donnés en entrée. L'affichage final montre chaque phrase test avec sa catégorie prédite, ce qui est utile pour l'analyse de texte et la classification automatique.

In [9]:
```python
import random

from nltk import NaiveBayesClassifier
from nltk.classify import accuracy as nltk_accuracy
from nltk.corpus import names

# Extract last N letters from the input word
# and that will act as our "feature"
def extract_features(word, N=2):
    last_n_letters = word[-N:]
    return {'feature': last_n_letters.lower()}

if __name__=='__main__':
    # Create training data using labeled names available in NLTK
    male_list = [(name, 'male') for name in names.words('male.txt')]
    female_list = [(name, 'female') for name in names.words('female.txt')]
    data = (male_list + female_list)

    # Seed the random number generator
    random.seed(5)
    # Shuffle the data
    random.shuffle(data)

    # Create test data
    input_names = ['Alexander', 'Danielle', 'David', 'Cheryl']

    # Define the number of samples used for train and test
    num_train = int(0.8 * len(data))

    # Iterate through different lengths to compare the accuracy
    for i in range(1, 6):
        print('\nNumber of end letters:', i)
        features = [(extract_features(n, i), gender) for (n, gender) in data]
        train_data, test_data = features[:num_train], features[num_train:]
        classifier = NaiveBayesClassifier.train(train_data)

        # Compute the accuracy of the classifier
        accuracy = round(100 * nltk_accuracy(classifier, test_data), 2)
        print('Accuracy = ' + str(accuracy) + '%')

        # Predict outputs for input names using the trained classifier model
        for name in input_names:
            print(name, '==>', classifier.classify(extract_features(name, i)))
```

```
Number of end letters: 1
Accuracy = 74.7%
Alexander ==> male
Danielle ==> female
David ==> male
Cheryl ==> male

Number of end letters: 2
Accuracy = 78.79%
Alexander ==> male
Danielle ==> female
David ==> male
Cheryl ==> female

Number of end letters: 3
Accuracy = 77.22%
Alexander ==> male
Danielle ==> female
David ==> male
Cheryl ==> female

Number of end letters: 4
Accuracy = 69.98%
Alexander ==> male
Danielle ==> female
David ==> male
Cheryl ==> female

Number of end letters: 5
Accuracy = 64.63%
Alexander ==> male
Danielle ==> female
David ==> male
Cheryl ==> female
```

Ce code utilise un classificateur Naïve Bayes pour prédire le genre d'un prénom en fonction de ses dernières lettres. Il commence par extraire les prénoms masculins et féminins du corpus names de NLTK, puis les mélange de manière aléatoire. Ensuite, il entraîne un modèle sur 80 % des données et teste sa précision sur les 20 % restants. Il répète ce processus pour des longueurs de suffixes allant de 1 à 5 lettres afin de comparer l'impact sur la précision. Enfin, il utilise le modèle entraîné pour prédire le genre de nouveaux prénoms, comme Alexander ou Danielle. Ce type de classification est souvent utilisé en NLP et analyse de données linguistiques.

In [10]:

```python
from nltk.corpus import movie_reviews
from nltk.classify import NaiveBayesClassifier
from nltk.classify.util import accuracy as nltk_accuracy

# Extract features from the input list of words
def extract_features(words):
    return dict([(word, True) for word in words])

if __name__=='__main__':
    # Load the reviews from the corpus
    fileids_pos = movie_reviews.fileids('pos')
    fileids_neg = movie_reviews.fileids('neg')

    # Extract the features from the reviews
    features_pos = [(extract_features(movie_reviews.words(
            fileids=[f])), 'Positive') for f in fileids_pos]
    features_neg = [(extract_features(movie_reviews.words(
            fileids=[f])), 'Negative') for f in fileids_neg]

    # Define the train and test split (80% and 20%)
    threshold = 0.8
    num_pos = int(threshold * len(features_pos))
    num_neg = int(threshold * len(features_neg))

     # Create training and training datasets
    features_train = features_pos[:num_pos] + features_neg[:num_neg]
    features_test = features_pos[num_pos:] + features_neg[num_neg:]

    # Print the number of datapoints used
    print('\nNumber of training datapoints:', len(features_train))
    print('Number of test datapoints:', len(features_test))

    # Train a Naive Bayes classifier
    classifier = NaiveBayesClassifier.train(features_train)
    print('\nAccuracy of the classifier:', nltk_accuracy(
            classifier, features_test))

    N = 15
    print('\nTop ' + str(N) + ' most informative words:')
    for i, item in enumerate(classifier.most_informative_features()):
        print(str(i+1) + '. ' + item[0])
        if i == N - 1:
            break

    # Test input movie reviews
    input_reviews = [
        'The costumes in this movie were great',
        'I think the story was terrible and the characters were very weak',
        'People say that the director of the movie is amazing',
        'This is such an idiotic movie. I will not recommend it to anyone.'
    ]

    print("\nMovie review predictions:")
    for review in input_reviews:
        print("\nReview:", review)

        # Compute the probabilities
        probabilities = classifier.prob_classify(extract_features(review.split()

        # Pick the maximum value
        predicted_sentiment = probabilities.max()

        # Print outputs
        print("Predicted sentiment:", predicted_sentiment)
        print("Probability:", round(probabilities.prob(predicted_sentiment), 2))
```

```
Number of training datapoints: 1600
Number of test datapoints: 400

Accuracy of the classifier: 0.735

Top 15 most informative words:
1. outstanding
2. insulting
3. vulnerable
4. ludicrous
5. uninvolving
6. astounding
7. avoids
8. fascination
9. affecting
10. animators
11. anna
12. darker
13. seagal
14. symbol
15. idiotic

Movie review predictions:

Review: The costumes in this movie were great
Predicted sentiment: Positive
Probability: 0.59

Review: I think the story was terrible and the characters were very weak
Predicted sentiment: Negative
Probability: 0.8

Review: People say that the director of the movie is amazing
Predicted sentiment: Positive
Probability: 0.6

Review: This is such an idiotic movie. I will not recommend it to anyone.
Predicted sentiment: Negative
Probability: 0.87
```

Ce code entraîne un classificateur Naïve Bayes pour effectuer une analyse de sentiment sur des critiques de films. Il commence par charger les critiques positives et négatives du corpus movie_reviews de NLTK, puis extrait leurs caractéristiques en représentant chaque mot comme une caractéristique binaire (présent ou non). Ensuite, il divise les données en 80 % pour l'entraînement et 20 % pour le test. Une fois le modèle entraîné, il affiche sa précision et identifie les 15 mots les plus informatifs pour la classification. Enfin, il teste le modèle sur de nouvelles critiques de films et prédit leur sentiment positif ou négatif avec un score de probabilité. Ce type d'analyse est utilisé en NLP et opinion mining pour détecter les émotions dans les textes.

In [1]:
```
import os
import nltk
from nltk.tokenize import RegexpTokenizer
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from gensim import models, corpora

# Télécharger les stopwords si nécessaire
nltk.download('stopwords')

# Définition du fichier
input_file = r"C:\Users\smain\OneDrive\Documents\data.txt"  # Chemin absolu Wind

# Vérifier si le fichier existe
if not os.path.exists(input_file):
    print(f"❌ Erreur : Le fichier '{input_file}' est introuvable.")
    exit()

# Charger les données
def load_data(input_file):
    with open(input_file, 'r', encoding='utf-8') as f:
        data = [line.strip() for line in f.readlines()]

    print(f"✅ {len(data)} lignes chargées depuis '{input_file}'")
    return data

# Fonction de prétraitement (tokenisation, stopwords, stemming)
def process(input_text):
    tokenizer = RegexpTokenizer(r'\w+')
    stemmer = SnowballStemmer('english')
    stop_words = set(stopwords.words('english'))  # Utilisation d'un set pour re

    tokens = tokenizer.tokenize(input_text.lower())
    tokens = [word for word in tokens if word not in stop_words]
    tokens_stemmed = [stemmer.stem(word) for word in tokens]

    return tokens_stemmed

if __name__ == '__main__':
    # Charger et traiter les données
    data = load_data(input_file)
    tokens = [process(text) for text in data]

    # Création du dictionnaire et de la matrice terme-document
    dict_tokens = corpora.Dictionary(tokens)
    doc_term_mat = [dict_tokens.doc2bow(token) for token in tokens]

    # Nombre de sujets à identifier
    num_topics = 2

    # Création du modèle LDA
    print("🔄 Entraînement du modèle LDA...")
    ldamodel = models.LdaModel(doc_term_mat, num_topics=num_topics, id2word=dict
    print("✅ Modèle LDA entraîné !")

    # Affichage des sujets avec leurs mots-clés
    num_words = 5
    print(f"\n📌 Top {num_words} mots clés pour chaque sujet :")
    topics = ldamodel.show_topics(num_topics=num_topics, num_words=num_words, fc

    for topic_num, word_weights in topics:
        print(f"\n🟢 Sujet {topic_num + 1}:")
        for word, weight in word_weights:
            print(f"  {word} ==> {round(weight * 100, 2)}%")
```

```
✅ 10 lignes chargées depuis 'C:\Users\smain\OneDrive\Documents\data.txt'
🔄 Entraînement du modèle LDA...
✅ Modèle LDA entraîné !

📌 Top 5 mots clés pour chaque sujet :

🟢 Sujet 1:
  empir ==> 3.89%
  mathemat ==> 3.89%
  time ==> 2.78%
  histor ==> 2.78%
  peopl ==> 2.78%

🟢 Sujet 2:
  europ ==> 3.12%
  cultur ==> 3.12%
  formul ==> 3.12%
  set ==> 1.88%
  structur ==> 1.88%
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\smain\AppData\Roaming\nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Ce code effectue une analyse thématique sur un fichier texte (data.txt) en utilisant le modèle LDA (Latent Dirichlet Allocation) de Gensim. Il commence par charger le fichier et applique un prétraitement du texte, incluant la tokenisation, la suppression des stopwords et le stemming. Ensuite, il construit un dictionnaire de mots et une matrice terme-document pour préparer les données. Il entraîne ensuite un modèle LDA avec deux sujets, puis affiche les cinq mots les plus représentatifs pour chaque sujet. Ce type d'analyse est couramment utilisé en NLP pour extraire des thèmes cachés dans de grands corpus de texte.