Laboratorio 5 - Representaciones Vectoriales de Texto

Francisco Castillo - 21562

1. Preprocesamiento del corpus

In [1]:
```
from sklearn.datasets import fetch_20newsgroups
import re
import nltk
nltk.download('punkt', quiet=True)
nltk.download('stopwords', quiet=True)
nltk.download('wordnet', quiet=True)
nltk.download('punkt_tab', quiet=True)
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from sklearn.model_selection import train_test_split
```

In [2]:
```
categories=['talk.politics.guns', 'talk.politics.mideast', 'talk.politics.misc', 'rec.autos']
news = fetch_20newsgroups(subset='all', categories=categories, remove=('headers', 'footers', 'quotes'), shuffle=True,
```

In [3]:
```
def preprocess_text(text):
    text = text.lower()
    text = re.sub(r'[^a-z\s]', '', text)
    # Remove sequences of repeated characters or potential artifacts like 'outofcontrolgif' and the subsequent random
    text = re.sub(r'\b\w*outofcontrolgif\w*\b', '', text) # Remove the specific 'outofcontrolgif' word and any attach
    text = re.sub(r'\b[a-z]{15,}\b', '', text) # Remove words that are 15 or more characters long and consist only of
    return text
```

In [4]:
```
preprocessed_news = [preprocess_text(doc) for doc in news.data]
tokenized_news = [nltk.word_tokenize(doc) for doc in preprocessed_news]

# Remove tokens starting with "begin"
tokenized_news = [[word for word in doc if not word.startswith('begin')] for doc in tokenized_news]
```

In [5]:
```
for i in range(10):
    print(tokenized_news[i])
```

['what', 'say', 'you', 'and', 'nick', 'go', 'somewhere', 'else', 'with', 'this', 'shool', 'yard', 'crap']
['in', 'europe', 'you', 'can', 'buy', 'a', 'ix', 'with', 'computer', 'controlled', 'diffs', 'rather', 'than', 'the',
'horrid', 'viscous', 'coupled', 'ones', 'of', 'the', 'outgoing', 'ix']
['i', 'dont', 'think', 'weve', 'got', 'a', 'conspiracy', 'on', 'our', 'hands', 'or', 'anything', 'vaugely', 'simila
r', 'i', 'do', 'think', 'that', 'the', 'feds', 'showed', 'a', 'distinct', 'lack', 'of', 'both', 'intelligence', 'an
d', 'disregard', 'for', 'others', 'safety', 'throughout', 'this', 'whole', 'mess', 'i', 'do', 'think', 'the', 'fbi',
'and', 'the', 'batf', 'screwed', 'up', 'big', 'what', 'made', 'me', 'really', 'concerned', 'was', 'fbi', 'director',
'william', 'sessions', 'being', 'on', 'cnn', 'engaging', 'in', 'what', 'could', 'only', 'be', 'called', 'spin', 'cont
rol', 'before', 'the', 'place', 'had', 'even', 'cooled', 'down', 'evertyhing', 'had', 'literally', 'blown', 'up', 'i
n', 'their', 'faces', 'and', 'i', 'felt', 'there', 'had', 'to', 'be', 'something', 'more', 'important', 'he', 'shoul
d', 'have', 'been', 'doing']
['mfehaeqkkl', 'mgfqqloozfemdj', 'lzylluke', 'end']
['for', 'starters', 'they', 'could', 'have', 'gone', 'on', 'waiting', 'and', 'negotiating', 'the', 'davidians', 'were
nt', 'going', 'anywhere', 'and', 'their', 'supplies', 'had', 'to', 'be', 'limited', 'large', 'perhaps', 'but', 'limit
ed', 'if', 'they', 'had', 'simply', 'fired', 'the', 'compound', 'by', 'themselves', 'without', 'govt', 'tanks', 'smas
hing', 'down', 'their', 'walls', 'then', 'at', 'least', 'the', 'govt', 'would', 'not', 'be', 'guilty', 'of', 'havin
g', 'again', 'used', 'an', 'inappropriate', 'level', 'of', 'force', 'and', 'would', 'have', 'been', 'able', 'to', 'us
e', 'the', 'meantime', 'to', 'continue', 'to', 'pressure', 'and', 'negotiate', 'no', 'they', 'would', 'not', 'have',
'looked', 'good', 'on', 'the', 'news', 'in', 'six', 'months', 'or', 'a', 'year', 'but', 'they', 'sure', 'as', 'hell',
'dont', 'look', 'very', 'good', 'now', 'larry', 'smith', 'smithctroncom', 'no', 'i', 'dont', 'speak', 'for', 'cabletr
on', 'need', 'you', 'ask']
['is', 'this', 'a', 'joke']
['hi', 'maybe', 'someone', 'can', 'help', 'me', 'here', 'i', 'am', 'looking', 'to', 'buy', 'this', 'nissan', 'maxim
a', 'gxe', 'for', 'cdn', 'right', 'now', 'the', 'car', 'has', 'km', 'or', 'about', 'miles', 'on', 'it', 'a', 'typica
l', 'mileage', 'for', 'cars', 'seem', 'to', 'be', 'about', 'km', 'or', 'about', 'k', 'mi', 'the', 'seller', 'just',
'informed', 'me', 'that', 'when', 'he', 'brought', 'the', 'car', 'in', 'for', 'certification', 'he', 'was', 'told',
'that', 'the', 'front', 'break', 'pads', 'and', 'the', 'exhausts', 'had', 'to', 'be', 'replaced', 'to', 'meet', 'th
e', 'legal', 'standards', 'he', 'said', 'he', 'will', 'replace', 'the', 'components', 'before', 'selling', 'the', 'ca
r', 'to', 'me', 'being', 'copmletely', 'ignorant', 'to', 'the', 'technical', 'stuff', 'on', 'cars', 'i', 'dont', 'kno
w', 'what', 'this', 'could', 'mean', 'is', 'k', 'km', 'about', 'the', 'time', 'typical', 'for', 'replacing', 'the',
'above', 'mentioned', 'items', 'or', 'is', 'this', 'an', 'indication', 'that', 'the', 'car', 'was', 'abused', 'woul
d', 'other', 'things', 'break', 'down', 'or', 'have', 'to', 'be', 'replaced', 'soon', 'the', 'seller', 'told', 'me',
'that', 'he', 'used', 'the', 'car', 'on', 'the', 'highway', 'a', 'lot', 'but', 'i', 'dont', 'know', 'how', 'to', 'ver
ify', 'this', 'ive', 'seen', 'the', 'paint', 'chipped', 'away', 'in', 'tiny', 'dots', 'in', 'the', 'front', 'edge',
'of', 'the', 'hood', 'though', 'although', 'the', 'maxima', 'is', 'an', 'excellent', 'car', 'and', 'the', 'car', 'i
s', 'very', 'clean', 'and', 'well', 'kept', 'its', 'currently', 'out', 'of', 'warranty', 'a', 'similarly', 'priced',
'accord', 'with', 'k', 'km', 'will', 'have', 'years', 'or', 'k', 'km', 'worth', 'of', 'warranty', 'left', 'and', 'i',
'dont', 'want', 'to', 'worry', 'about', 'paying', 'for', 'any', 'repair', 'bills', 'but', 'i', 'also', 'need', 'a',
'car', 'for', 'people', 'when', 'will', 'the', 'new', 'maxima', 'come', 'out', 'by', 'the', 'way', 'i', 'would', 'ver
y', 'much', 'appreciate', 'your', 'input', 'in', 'this', 'please', 'reply', 'by', 'email', 'preferred', 'or', 'post',
'in', 'this', 'newsgroup', 'thanks', 'ryan']
['watch', 'the', 'videotape', 'carefully', 'the', 'cnn', 'coverage', 'was', 'fairly', 'decisive', 'the', 'first', 'fi

```
re', 'starts', 'in', 'the', 'tower', 'this', 'is', 'three', 'storeys', 'high', 'and', 'there', 'is', 'a', 'flag', 't
o', 'the', 'right', 'of', 'it', 'on', 'the', 'picture', 'the', 'second', 'fire', 'starts', 'in', 'another', 'tower',
'which', 'is', 'similar', 'to', 'the', 'first', 'only', 'two', 'storeys', 'high', 'the', 'flag', 'is', 'on', 'the',
'left', 'in', 'the', 'camera', 'picture', 'that', 'shows', 'this', 'fire', 'starting', 'thus', 'the', 'camera', 'pict
ures', 'cleraly', 'show', 'the', 'fire', 'starting', 'at', 'two', 'separate', 'locations', 'the', 'fbi', 'report',
'a', 'third', 'i', 'was', 'not', 'able', 'to', 'verify', 'it', 'from', 'the', 'videotape', 'however', 'someone', 'els
e', 'identified', 'a', 'fire', 'shown', 'to', 'be', 'starting', 'behind', 'the', 'small', 'tower', 'in', 'the', 'seco
nd', 'flag', 'on', 'left', 'camera', 'angle', 'the', 'flames', 'coming', 'out', 'of', 'the', 'building', 'are', 'yell
oworange', 'this', 'is', 'the', 'normal', 'colour', 'for', 'carbon', 'compounds', 'burning', 'the', 'flames', 'were',
'those', 'of', 'a', 'solid', 'or', 'confined', 'liquid', 'burning', 'not', 'of', 'a', 'gas', 'exploding', 'the', 'exp
losion', 'that', 'occurs', 'mid', 'way', 'along', 'the', 'building', 'is', 'certainly', 'not', 'an', 'explosive', 'th
ough', 'the', 'cloud', 'itself', 'is', 'on', 'fire', 'this', 'would', 'seem', 'to', 'be', 'most', 'likely', 'to', 'b
e', 'some', 'sort', 'of', 'fuel', 'oil', 'store', 'exploding', 'rather', 'than', 'the', 'explosion', 'of', 'a', 'maga
zine', 'depends', 'entirely', 'on', 'how', 'they', 'were', 'distributed', 'you', 'would', 'not', 'be', 'able', 'to',
'identify', 'ammunition', 'rounds', 'going', 'off', 'from', 'video', 'camera', 'coverage', 'from', 'a', 'mile', 'awa
y', 'if', 'and', 'when', 'the', 'fbi', 'release', 'pictures', 'from', 'cmeras', 'on', 'the', 'armoured', 'vehicles',
'which', 'presumably', 'exist', 'it', 'might', 'be', 'possible', 'to', 'get', 'a', 'clearer', 'picture', 'if', 'anyon
e', 'expects', 'to', 'see', 'explosions', 'hollywood', 'style', 'aka', 'rambo', 'movies', 'then', 'remember', 'that',
'in', 'real', 'life', 'cars', 'do', 'not', 'burst', 'into', 'flames', 'when', 'going', 'over', 'cliffs', 'just', 'abo
ut', 'the', 'most', 'you', 'could', 'expect', 'would', 'be', 'to', 'see', 'the', 'grenades', 'going', 'off', 'since',
'the', 'building', 'was', 'designed', 'to', 'be', 'blast', 'proof', 'to', 'some', 'extent', 'it', 'would', 'be', 'dif
ficult', 'to', 'distinguish', 'the', 'grenades', 'going', 'off', 'from', 'the', 'collapse', 'of', 'the', 'building',
'due', 'to', 'the', 'fire', 'paranoia', 'you', 'wouldnt', 'beleive', 'the', 'fbi', 'if', 'they', 'showed', 'you',
'a', 'picture', 'of', 'koresh', 'himself', 'setting', 'light', 'to', 'the', 'place', 'your', 'mindset', 'is', 'such',
'that', 'you', 'are', 'simply', 'unable', 'to', 'accept', 'as', 'true', 'anything', 'that', 'might', 'suggest', 'tha
t', 'a', 'group', 'of', 'heavily', 'armed', 'weapons', 'fanatics', 'might', 'indeed', 'be', 'in', 'the', 'wrong', 'th
e', 'gun', 'lobby', 'cant', 'accept', 'that', 'the', 'bd', 'set', 'light', 'to', 'the', 'place', 'because', 'that',
'would', 'mean', 'that', 'koreh', 'had', 'murdered', 'children', 'that', 'would', 'mean', 'that', 'their', 'taking',
'his', 'account', 'of', 'the', 'murder', 'of', 'batf', 'agents', 'would', 'be', 'even', 'less', 'credible', 'than',
'it', 'was', 'to', 'start', 'with', 'koresh', 'had', 'days', 'to', 'come', 'out', 'with', 'his', 'hands', 'up', 'an
d', 'face', 'a', 'fair', 'trial', 'instead', 'he', 'ordered', 'the', 'murder', 'of', 'everyone', 'in', 'the', 'plac
e']
[]
['my', 'wife', 'rarely', 'carries', 'a', 'purse', 'so', 'all', 'of', 'her', 'crap', 'ends', 'up', 'in', 'my', 'pocket
s']
```

En el preprocesamiento eliminamos lo que parecen ser diferentes gifs e imagenes que contienen cadenas de texto "aleatorias" y pueden alterar el desempeño de la vectorización.

In [6]:
```
stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()
```

```
def remove_stopwords_and_lemmatize(tokens):
    return [lemmatizer.lemmatize(word) for word in tokens if word not in stop_words]

tokenized_news_cleaned = [remove_stopwords_and_lemmatize(doc) for doc in tokenized_news]

# Remove empty documents and their corresponding labels
non_empty_indices = [i for i, doc in enumerate(tokenized_news_cleaned) if doc]
tokenized_news_cleaned = [tokenized_news_cleaned[i] for i in non_empty_indices]
cleaned_target = [news.target[i] for i in non_empty_indices]
```

In [7]:
```
for i in range(10):
    print(tokenized_news_cleaned[i])
```

['say', 'nick', 'go', 'somewhere', 'else', 'shool', 'yard', 'crap']
['europe', 'buy', 'ix', 'computer', 'controlled', 'diffs', 'rather', 'horrid', 'viscous', 'coupled', 'one', 'outgoin
g', 'ix']
['dont', 'think', 'weve', 'got', 'conspiracy', 'hand', 'anything', 'vaugely', 'similar', 'think', 'fed', 'showed', 'd
istinct', 'lack', 'intelligence', 'disregard', 'others', 'safety', 'throughout', 'whole', 'mess', 'think', 'fbi', 'ba
tf', 'screwed', 'big', 'made', 'really', 'concerned', 'fbi', 'director', 'william', 'session', 'cnn', 'engaging', 'co
uld', 'called', 'spin', 'control', 'place', 'even', 'cooled', 'evertyhing', 'literally', 'blown', 'face', 'felt', 'so
mething', 'important']
['mfehaeqkkl', 'mgfqqloozfemdj', 'lzylluke', 'end']
['starter', 'could', 'gone', 'waiting', 'negotiating', 'davidians', 'werent', 'going', 'anywhere', 'supply', 'limite
d', 'large', 'perhaps', 'limited', 'simply', 'fired', 'compound', 'without', 'govt', 'tank', 'smashing', 'wall', 'lea
st', 'govt', 'would', 'guilty', 'used', 'inappropriate', 'level', 'force', 'would', 'able', 'use', 'meantime', 'conti
nue', 'pressure', 'negotiate', 'would', 'looked', 'good', 'news', 'six', 'month', 'year', 'sure', 'hell', 'dont', 'lo
ok', 'good', 'larry', 'smith', 'smithctroncom', 'dont', 'speak', 'cabletron', 'need', 'ask']
['joke']
['hi', 'maybe', 'someone', 'help', 'looking', 'buy', 'nissan', 'maximum', 'gxe', 'cdn', 'right', 'car', 'km', 'mile',
'typical', 'mileage', 'car', 'seem', 'km', 'k', 'mi', 'seller', 'informed', 'brought', 'car', 'certification', 'tol
d', 'front', 'break', 'pad', 'exhaust', 'replaced', 'meet', 'legal', 'standard', 'said', 'replace', 'component', 'sel
ling', 'car', 'copmletely', 'ignorant', 'technical', 'stuff', 'car', 'dont', 'know', 'could', 'mean', 'k', 'km', 'tim
e', 'typical', 'replacing', 'mentioned', 'item', 'indication', 'car', 'abused', 'would', 'thing', 'break', 'replace
d', 'soon', 'seller', 'told', 'used', 'car', 'highway', 'lot', 'dont', 'know', 'verify', 'ive', 'seen', 'paint', 'chi
pped', 'away', 'tiny', 'dot', 'front', 'edge', 'hood', 'though', 'although', 'maximum', 'excellent', 'car', 'car', 'c
lean', 'well', 'kept', 'currently', 'warranty', 'similarly', 'priced', 'accord', 'k', 'km', 'year', 'k', 'km', 'wort
h', 'warranty', 'left', 'dont', 'want', 'worry', 'paying', 'repair', 'bill', 'also', 'need', 'car', 'people', 'new',
'maximum', 'come', 'way', 'would', 'much', 'appreciate', 'input', 'please', 'reply', 'email', 'preferred', 'post', 'n
ewsgroup', 'thanks', 'ryan']
['watch', 'videotape', 'carefully', 'cnn', 'coverage', 'fairly', 'decisive', 'first', 'fire', 'start', 'tower', 'thre
e', 'storey', 'high', 'flag', 'right', 'picture', 'second', 'fire', 'start', 'another', 'tower', 'similar', 'first',
'two', 'storey', 'high', 'flag', 'left', 'camera', 'picture', 'show', 'fire', 'starting', 'thus', 'camera', 'pictur
e', 'cleraly', 'show', 'fire', 'starting', 'two', 'separate', 'location', 'fbi', 'report', 'third', 'able', 'verify',
'videotape', 'however', 'someone', 'else', 'identified', 'fire', 'shown', 'starting', 'behind', 'small', 'tower', 'se
cond', 'flag', 'left', 'camera', 'angle', 'flame', 'coming', 'building', 'yelloworange', 'normal', 'colour', 'carbo
n', 'compound', 'burning', 'flame', 'solid', 'confined', 'liquid', 'burning', 'gas', 'exploding', 'explosion', 'occur
s', 'mid', 'way', 'along', 'building', 'certainly', 'explosive', 'though', 'cloud', 'fire', 'would', 'seem', 'likel
y', 'sort', 'fuel', 'oil', 'store', 'exploding', 'rather', 'explosion', 'magazine', 'depends', 'entirely', 'distribut
ed', 'would', 'able', 'identify', 'ammunition', 'round', 'going', 'video', 'camera', 'coverage', 'mile', 'away', 'fb
i', 'release', 'picture', 'cmeras', 'armoured', 'vehicle', 'presumably', 'exist', 'might', 'possible', 'get', 'cleare
r', 'picture', 'anyone', 'expects', 'see', 'explosion', 'hollywood', 'style', 'aka', 'rambo', 'movie', 'remember', 'r
eal', 'life', 'car', 'burst', 'flame', 'going', 'cliff', 'could', 'expect', 'would', 'see', 'grenade', 'going', 'sinc
e', 'building', 'designed', 'blast', 'proof', 'extent', 'would', 'difficult', 'distinguish', 'grenade', 'going', 'col
lapse', 'building', 'due', 'fire', 'paranoia', 'wouldnt', 'beleive', 'fbi', 'showed', 'picture', 'koresh', 'setting',
'light', 'place', 'mindset', 'simply', 'unable', 'accept', 'true', 'anything', 'might', 'suggest', 'group', 'heavil

```
y', 'armed', 'weapon', 'fanatic', 'might', 'indeed', 'wrong', 'gun', 'lobby', 'cant', 'accept', 'bd', 'set', 'light',
'place', 'would', 'mean', 'koreh', 'murdered', 'child', 'would', 'mean', 'taking', 'account', 'murder', 'batf', 'agen
t', 'would', 'even', 'less', 'credible', 'start', 'koresh', 'day', 'come', 'hand', 'face', 'fair', 'trial', 'instea
d', 'ordered', 'murder', 'everyone', 'place']
['wife', 'rarely', 'carry', 'purse', 'crap', 'end', 'pocket']
['oh', 'ok', 'wondering', 'real', 'expert', 'weapon', 'wondering', 'would', 'job']
```

También, dado que no generaremos texto procedemos a lematizar y eliminar las stopwords para obtener mejor precisión.

In [8]:
```
X_train, X_test, y_train, y_test = train_test_split(tokenized_news_cleaned, cleaned_target, test_size=0.25, random_st

print(f"Training set size: {len(X_train)}")
print(f"Testing set size: {len(X_test)}")
```

```
Training set size: 2616
Testing set size: 873
```

2. Construcción de representación TF-IDF

In [9]:
```
from sklearn.feature_extraction.text import TfidfVectorizer
tfidf_vectorizer = TfidfVectorizer()
```

In [10]:
```
X_train_tfidf = tfidf_vectorizer.fit_transform([" ".join(doc) for doc in X_train])
X_test_tfidf = tfidf_vectorizer.transform([" ".join(doc) for doc in X_test])

print("TTF-IDF Train: ", X_train_tfidf.shape)
print("TF-IDF Test: ", X_test_tfidf.shape)
```

```
TTF-IDF Train:  (2616, 25760)
TF-IDF Test:  (873, 25760)
```

In [11]:
```
def get_top_tfidf_words(tfidf_matrix_row, feature_names, n=5):
    # Get the indices of the top n TF-IDF scores
    top_n_indices = tfidf_matrix_row.argsort()[-n:][::-1]
    # Get the corresponding words and scores
    top_n_words = [(feature_names[i], tfidf_matrix_row[i]) for i in top_n_indices]
    return top_n_words
```

In [12]:
```
feature_names = tfidf_vectorizer.get_feature_names_out()
```

```python
for i in range(5):
    print(f"\n--- Document {i+1} ---")
    tfidf_row = X_train_tfidf[i].toarray()[0]
    top_words = get_top_tfidf_words(tfidf_row, feature_names)
    for word, score in top_words:
        print(f"{word}: {score:.4f}")
```

```
--- Document 1 ---
ohio: 0.2662
frank: 0.2662
bad: 0.2527
regardless: 0.2220
trial: 0.2107

--- Document 2 ---
serb: 0.6182
muslim: 0.3360
bosnian: 0.2662
serbia: 0.2124
refused: 0.1876

--- Document 3 ---
being: 0.2249
automobile: 0.2039
honest: 0.2028
value: 0.1680
criminal: 0.1600

--- Document 4 ---
gun: 0.3145
shoot: 0.2573
sw: 0.2199
training: 0.2163
revolver: 0.1899

--- Document 5 ---
slip: 0.4499
tranny: 0.2771
gear: 0.2504
usually: 0.2110
speed: 0.2104
```

Palabras con mayor peso

Las palabras con más peso (los scores TF-IDF más altos) en un documento particular son aquellas que son importantes dentro de ese documento y, al mismo tiempo, no son muy comunes en el resto del corpus. Esto significa que esas palabras son las que mejor representan o distinguen el tema o contenido específico de ese documento en comparación con otros documentos. Por ejemplo, en el "Documento 2", palabras como "serb", "muslim", "bosnian" y "serbia" tienen altos pesos, lo que sugiere que este documento trata sobre el conflicto en los Balcanes. De manera similar, en el "Documento 4", términos como "gun", "shoot", "training" y "revolver" indican claramente que el documento está relacionado con el tema de las armas. En resumen, estos altos pesos nos señalan las palabras clave que son muy relevantes y distintivas para cada documento particular.

Limitaciones semánticas de TF-IDF

TF-IDF se basa únicamente en la frecuencia y rareza de las palabras, sin considerar su significado o el contexto en el que aparecen. Al tratar los documentos como una "bolsa de palabras", ignora el orden y las relaciones entre ellas, lo que impide capturar sinónimos, polisemia o relaciones semánticas más profundas. Esto significa que, aunque identifica palabras clave relevantes, no comprende el sentido completo ni las conexiones entre las ideas en un texto.

3. Construcción de Representación PPMI

In [13]:
```
from collections import defaultdict
import pandas as pd
from scipy.sparse import lil_matrix

def build_cooccurrence_matrix(tokenized_docs, window_size):
    word_counts = defaultdict(int)
    cooccurrence_counts = defaultdict(lambda: defaultdict(int))

    for doc in tokenized_docs:
        for i, target_word in enumerate(doc):
            word_counts[target_word] += 1
            start_index = max(0, i - window_size)
            end_index = min(len(doc), i + window_size + 1)

            for j in range(start_index, end_index):
                if i != j:
```

```
                    context_word = doc[j]
                    cooccurrence_counts[target_word][context_word] += 1

    vocabulary = list(word_counts.keys())
    word_to_index = {word: i for i, word in enumerate(vocabulary)}

    # Create a sparse matrix for co-occurrence counts
    cooccurrence_matrix = lil_matrix((len(vocabulary), len(vocabulary)), dtype=int)

    for target_word, context_data in cooccurrence_counts.items():
        target_index = word_to_index[target_word]
        for context_word, count in context_data.items():
            context_index = word_to_index[context_word]
            cooccurrence_matrix[target_index, context_index] = count

    return cooccurrence_matrix, vocabulary
```

In [14]:

```
import numpy as np

def calculate_ppmi(cooccurrence_matrix, vocabulary):
    total_pairs = cooccurrence_matrix.sum()
    word_sums = cooccurrence_matrix.sum(axis=1).A1  # Sum of each row (target words)
    context_sums = cooccurrence_matrix.sum(axis=0).A1 # Sum of each column (context words)

    ppmi_matrix = lil_matrix(cooccurrence_matrix.shape, dtype=float)

    rows, cols = cooccurrence_matrix.nonzero() # Get indices of non-zero elements

    for row, col in zip(rows, cols):
        cooc_count = cooccurrence_matrix[row, col]
        p_target_context = cooc_count / total_pairs
        p_target = word_sums[row] / total_pairs
        p_context = context_sums[col] / total_pairs

        # Handle cases to avoid division by zero or log of zero
        if p_target > 0 and p_context > 0:
            pmi = np.log2(p_target_context / (p_target * p_context))
            ppmi = max(0, pmi)
            ppmi_matrix[row, col] = ppmi
        else:
            ppmi_matrix[row, col] = 0
```

```
    return ppmi_matrix
```

In [15]:
```
cooccurrence_matrix, vocabulary = build_cooccurrence_matrix(tokenized_news_cleaned, window_size=4)
ppmi_matrix = calculate_ppmi(cooccurrence_matrix, vocabulary)
print("PPMI Matrix Shape:", ppmi_matrix.shape)
```

```
PPMI Matrix Shape: (29942, 29942)
```

4. Construcción de representación Word2Vec

from gensim.models import Word2Vec

In [16]:
```
vector_size = 100  # Dimension of the word vectors
window = 5       # Context window size
min_count = 5    # Ignore words with frequency lower than this
workers = 4      # Number of CPU cores to use
epochs = 10
```

In [17]:
```
!pip install gensim
```

```
Requirement already satisfied: gensim in /usr/local/lib/python3.11/dist-packages (4.3.3)
Requirement already satisfied: numpy<2.0,>=1.18.5 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.26.4)
Requirement already satisfied: scipy<1.14.0,>=1.7.0 in /usr/local/lib/python3.11/dist-packages (from gensim) (1.13.1)
Requirement already satisfied: smart-open>=1.8.1 in /usr/local/lib/python3.11/dist-packages (from gensim) (7.3.0.post
1)
Requirement already satisfied: wrapt in /usr/local/lib/python3.11/dist-packages (from smart-open>=1.8.1->gensim) (1.1
7.3)
```

In [18]:
```
from gensim.models import Word2Vec
model = Word2Vec(vector_size=vector_size, window=window, min_count=min_count, workers=workers)
model.build_vocab(tokenized_news_cleaned)

total_examples = model.corpus_count
model.train(tokenized_news_cleaned, total_examples=total_examples, epochs=epochs)
```

Out[18]:
```
(3818409, 4240950)
```

In [19]:
```
import numpy as np
```

```
def document_vector(doc, model):
    word_vectors = []
    for word in doc:
        # Check if the word exists in the Word2Vec model's vocabulary
        if word in model.wv.key_to_index:
            # If the word is in the vocabulary, retrieve its vector using model.wv[word] and append it to the list of
            word_vectors.append(model.wv[word])

    if word_vectors:
        return np.mean(word_vectors, axis=0)
    else:
        return np.zeros(model.vector_size)
```

In [20]:
```
document_embeddings = [document_vector(doc, model) for doc in tokenized_news_cleaned]
document_embeddings = np.array(document_embeddings)

print(document_embeddings.shape)
```

```
(3489, 100)
```

In [21]:
```
words_to_explore = ['jew', 'christian', 'muslim', 'police']

print("--- Words Most Similar ---")
for word in words_to_explore:
    if word in model.wv:
        print(f"\nWords similar to '{word}':")
        similar_words = model.wv.most_similar(word, topn=5)
        for sim_word, score in similar_words:
            print(f"  {sim_word}: {score:.4f}")
    else:
        print(f"'{word}' not found in vocabulary.")


# Calculate similarity between word pairs
word_pairs = [('gun', 'shoot'), ('car', 'wheel'), ('politics', 'mideast')]

print("\n--- Word Pair Similarities ---")
for word1, word2 in word_pairs:
    if word1 in model.wv and word2 in model.wv:
        similarity_score = model.wv.similarity(word1, word2)
        print(f"Similarity between '{word1}' and '{word2}': {similarity_score:.4f}")
```

```
    else:
        print(f"One or both words ('{word1}', '{word2}') not found in vocabulary.")
```

```
--- Words Most Similar ---

Words similar to 'jew':
  christian: 0.8579
  wwii: 0.8252
  jewish: 0.8184
  holocaust: 0.7992
  gaza: 0.7969

Words similar to 'christian':
  wwii: 0.9240
  extermination: 0.9200
  croatia: 0.9177
  islam: 0.9113
  hatred: 0.9106

Words similar to 'muslim':
  croat: 0.9609
  bosnian: 0.9478
  serb: 0.9012
  ethnically: 0.8858
  exterminated: 0.8799

Words similar to 'police':
  officer: 0.8381
  agent: 0.8106
  charge: 0.7711
  local: 0.7474
  sheriff: 0.7438

--- Word Pair Similarities ---
Similarity between 'gun' and 'shoot': 0.5484
Similarity between 'car' and 'wheel': 0.8818
Similarity between 'politics' and 'mideast': 0.8180
```

5. Evaluación Comparativa

In [22]:
```python
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
```

In [23]:
```python
param_grid = {
    'C': [0.01, 0.1, 1, 10],
    'penalty': ['l2']
}
```

In [24]:
```python
def train_model(X_train, y_train, X_test, y_test, param_grid):
    logistic_regression = LogisticRegression(max_iter=1000)
    grid_search = GridSearchCV(estimator=logistic_regression, param_grid=param_grid, scoring='precision_weighted', cv

    grid_search.fit(X_train, y_train)
    print("Best parameters found:")
    print(grid_search.best_params_)

    precision_score = grid_search.score(X_test, y_test)
    print(f"Precision: {precision_score:.4f}")

    return grid_search
```

In [25]:
```python
print("---  TF-IDF ---")
grid_search_tfidf = train_model(X_train_tfidf, y_train, X_test_tfidf, y_test, param_grid)
```

```
---  TF-IDF ---
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
/usr/local/lib/python3.11/dist-packages/sklearn/metrics/_classification.py:1565: UndefinedMetricWarning: Precision is
ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this b
ehavior.
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
Best parameters found:
{'C': 10, 'penalty': 'l2'}
Precision: 0.8470
```

In [26]:
```
word_to_index = {word: i for i, word in enumerate(vocabulary)}

def document_vector_ppmi(doc, ppmi_matrix, word_to_index):
    word_vectors = []
    for word in doc:
        if word in word_to_index:
            word_index = word_to_index[word]
            word_vector = ppmi_matrix[word_index, :].toarray().flatten()
            word_vectors.append(word_vector)

    if word_vectors:
        return np.mean(word_vectors, axis=0)
    else:
        return np.zeros(ppmi_matrix.shape[1])
```

In [27]:
```
ppmi_document_representations = [document_vector_ppmi(doc, ppmi_matrix, word_to_index) for doc in tokenized_news_clea
ppmi_document_representations = np.array(ppmi_document_representations)
X_train_ppmi, X_test_ppmi, y_train_check, y_test_check = train_test_split(
    ppmi_document_representations,
    cleaned_target,
    test_size=0.25,
    random_state=21562
)
```

In [28]:
```
print("\n--- Training Logistic Regression with PPMI ---")
grid_search_ppmi = train_model(X_train_ppmi, y_train, X_test_ppmi, y_test, param_grid)
```

```
--- Training Logistic Regression with PPMI ---
Best parameters found:
{'C': 0.01, 'penalty': 'l2'}
Precision: 0.8344
```

In [29]:
```
X_train_w2v, X_test_w2v, y_train_check_w2v, y_test_check_w2v = train_test_split(
    document_embeddings,
    cleaned_target,
    test_size=0.25,
    random_state=21562
)

print("\n--- Word2Vec ---")
grid_search_w2v = train_model(X_train_w2v, y_train, X_test_w2v, y_test, param_grid)
```

```
--- Word2Vec ---
Best parameters found:
{'C': 10, 'penalty': 'l2'}
Precision: 0.7884
```

In [30]:
```python
best_precision_tfidf = grid_search_tfidf.best_score_
best_precision_ppmi = grid_search_ppmi.best_score_
best_precision_w2v = grid_search_w2v.best_score_

print("\n--- Model Precision Comparison ---")
print(f"TF-IDF Model Best Precision: {best_precision_tfidf:.4f}")
print(f"PPMI Model Best Precision: {best_precision_ppmi:.4f}")
print(f"Word2Vec Model Best Precision: {best_precision_w2v:.4f}")
```

```
--- Model Precision Comparison ---
TF-IDF Model Best Precision: 0.8397
PPMI Model Best Precision: 0.8186
Word2Vec Model Best Precision: 0.7766
```

La representación TF-IDF logró el mejor rendimiento (0.8397), seguida por PPMI (0.8186) y finalmente Word2Vec (0.7766). Este orden podría parecer inicialmente sorprendente, ya que Word2Vec y PPMI tienen la capacidad de capturar relaciones semánticas que TF-IDF no. Sin embargo, para una tarea de clasificación de noticias como esta, la importancia distintiva de las palabras clave (capturada efectivamente por TF-IDF) puede ser un factor más determinante que las relaciones semánticas finas. Es posible que las categorías de noticias seleccionadas estén bien diferenciadas por la presencia de términos específicos que aparecen con alta frecuencia en una categoría y baja en otras. Además, el rendimiento de Word2Vec puede depender mucho de la calidad de los embeddings entrenados con un corpus particular y del método utilizado para agregarlos a nivel de documento (como el promedio de vectores).

6. Discusión Final

Cómo cada representación captura (o no) relaciones semánticas

TF-IDF no captura directamente relaciones semánticas, ya que se enfoca en la importancia de una palabra dentro de un documento en relación con su frecuencia en todo el corpus, sin considerar el contexto o el significado de las palabras. PPMI captura relaciones semánticas de coexistencia o asociación entre palabras basándose en la frecuencia con la que aparecen juntas dentro de una ventana de contexto; un PPMI alto indica una fuerte asociación. Por otro lado, Word2Vec captura relaciones semánticas más ricas al representar palabras en un espacio vectorial continuo donde palabras con significados similares o que aparecen en contextos

similares tienen vectores cercanos; permite capturar relaciones como analogías ("rey" - "hombre" + "mujer" ≈ "reina"). Aunque en este caso particular TF-IDF tuvo la mejor precisión, la capacidad de PPMI y Word2Vec para capturar asociaciones y significados a menudo es crucial para tareas que requieren una comprensión semántica más profunda.

¿En que escenarios es más útil cada técnica?

TF-IDF es especialmente útil en escenarios donde la relevancia de un documento se basa en la presencia de palabras clave distintivas y no tanto en las relaciones semánticas complejas, como en sistemas de recuperación de información, motores de búsqueda simples o filtrado de spam. PPMI es valioso para tareas que analizan la co-ocurrencia de palabras para entender asociaciones, construir diccionarios de sinónimos o realizar análisis de sentimientos basados en palabras adyacentes. Word2Vec es muy efectivo en aplicaciones donde es fundamental capturar el significado y las relaciones semánticas entre palabras y documentos, como en la traducción automática, recomendación de contenido, análisis de similitud semántica de documentos, o como capa de entrada para modelos de deep learning.

¿Cuáles son las limitaciones prácticas (memoria, tiempo de cómputo, interoperabilidad)?

Las representaciones TF-IDF y PPMI pueden resultar en matrices muy grandes y dispersas (con muchos ceros) para vocabularios extensos, lo que puede consumir mucha memoria y ser ineficiente computacionalmente para ciertas operaciones. El cálculo de la matriz de co-ocurrencia para PPMI puede ser costoso en tiempo, especialmente con ventanas de contexto amplias. Word2Vec produce vectores densos de menor dimensión, lo que generalmente reduce los requisitos de memoria y hace las operaciones posteriores más rápidas; sin embargo, el entrenamiento del modelo Word2Vec en grandes corpus puede ser computacionalmente intensivo. En cuanto a la interoperabilidad, las matrices TF-IDF y PPMI están ligadas al corpus con el que fueron creadas, mientras que los embeddings de Word2Vec pueden ser pre-entrenados en corpus masivos y reutilizados en diferentes tareas y dominios, ofreciendo mayor interoperabilidad.