C3U3ACT005. STEAMMER AND LEMATIZER

December 2, 2023

[1]:
```
import os
import re
import nltk
import pandas as pd
import matplotlib.pyplot as plt
from nltk.corpus import stopwords
from nltk.stem import PorterStemmer, WordNetLemmatizer
from sklearn.feature_extraction.text import TfidfVectorizer, TfidfTransformer
from sklearn.model_selection import train_test_split, cross_val_score,␣
 ↪StratifiedKFold
from sklearn.svm import SVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, precision_score, recall_score,␣
 ↪f1_score
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.feature_selection import chi2

nltk.download('punkt')
nltk.download('wordnet')
nltk.download('stopwords')
```

```
[nltk_data] Downloading package punkt to
[nltk_data]     C:\Users\luism\AppData\Roaming\nltk_data…
[nltk_data]   Package punkt is already up-to-date!
[nltk_data] Downloading package wordnet to
[nltk_data]     C:\Users\luism\AppData\Roaming\nltk_data…
[nltk_data]   Package wordnet is already up-to-date!
[nltk_data] Downloading package stopwords to
[nltk_data]     C:\Users\luism\AppData\Roaming\nltk_data…
[nltk_data]   Package stopwords is already up-to-date!
```

[1]:
```
True
```

[2]:
```
def clean_text(text):
    text = text.lower()
    text = re.sub(r'[^\w\s]', '', text)
    tokens = nltk.word_tokenize(text)
```

```
    stop_words = set(stopwords.words('spanish'))
    filtered_tokens = [word for word in tokens if word not in stop_words]
    cleaned_text = " ".join(filtered_tokens)
    return cleaned_text

def stem_and_lemmatize_text(text):
    stemmer = PorterStemmer()
    lemmatizer = WordNetLemmatizer()
    words = nltk.word_tokenize(text)
    stemmed_and_lemmatized_words = [stemmer.stem(lemmatizer.lemmatize(word))␣
 ↪for word in words]
    stemmed_and_lemmatized_text = " ".join(stemmed_and_lemmatized_words)
    return stemmed_and_lemmatized_text

url = "./data"
# Lista para almacenar el texto limpio
dataClean = []
etiquetas = [str(i) for i in range(0, 5)] * 10
# Número máximo de palabras en cada archivo
max_palabras = 2000

# Recorrer todos los archivos en la carpeta
for archivo in os.listdir(url):
    # Leer el contenido del archivo
    with open(os.path.join(url, archivo), "r", encoding="utf-8") as file:
        texto = file.read()

    # Limpiar el texto
    texto_limpio = clean_text(texto)

    # Limitar la cantidad de palabras
    texto_palabras = texto_limpio.split()[:max_palabras]
    texto_limite_palabras = " ".join(texto_palabras)

    # Agregar el texto limpio y limitado a la lista
    dataClean.append(texto_limite_palabras)
```

[3]:
```
# Aplicar lematización y stemming a los textos limpios
textos_stemmed_and_lemmatized = [stem_and_lemmatize_text(texto) for texto in␣
 ↪dataClean]

# Crear un vectorizador TF-IDF para training & testing
vectorizer_tfidf = TfidfVectorizer()

# Obtener la matriz de características para training & testing
tfidf_matrix = vectorizer_tfidf.fit_transform(textos_stemmed_and_lemmatized)
```

```
# Divide los datos en conjuntos de entrenamiento y prueba
X_train, X_test, y_train, y_test = train_test_split(tfidf_matrix, etiquetas,␣
 ↪test_size=0.2, random_state=42)

# Entrenar el algoritmo SVM para training & testing
svm_tfidf = SVC(kernel='linear', C=1.0, random_state=42)
svm_tfidf.fit(X_train, y_train)

# Predecir con el conjunto de prueba
y_pred = svm_tfidf.predict(X_test)

# Evaluar el rendimiento
accuracy = accuracy_score(y_test, y_pred)
precision = precision_score(y_test, y_pred, average='macro',zero_division=1)
recall = recall_score(y_test, y_pred, average='macro',zero_division=1)
f1 = f1_score(y_test, y_pred, average='macro',zero_division=1)

print("SVM Accuracy (Lemmatization & Stemming):", accuracy)
print("SVM Precision (Lemmatization & Stemming):", precision)
print("SVM Recall (Lemmatization & Stemming):", recall)
print("SVM F1-Score (Lemmatization & Stemming):", f1)

vectorizer_cv = TfidfVectorizer()
```

```
SVM Accuracy (Lemmatization & Stemming): 0.1
SVM Precision (Lemmatization & Stemming): 0.82
SVM Recall (Lemmatization & Stemming): 0.2
SVM F1-Score (Lemmatization & Stemming): 0.03636363636363636
```

[4]:
```
# Obtener la matriz de características para cross-validation
tfidf_matrix_cv = vectorizer_cv.fit_transform(textos_stemmed_and_lemmatized)

# Entrenar el algoritmo SVM para cross-validation
svm_tfidf_cv = SVC(kernel='linear', C=1.0, random_state=42)

cv_stratified = StratifiedKFold(n_splits=2, shuffle=True, random_state=10)
scores_svm_cv = cross_val_score(svm_tfidf_cv, tfidf_matrix_cv, etiquetas,␣
 ↪cv=cv_stratified)

# Obtener el rendimiento promedio para cross-validation
mean_accuracy_svm_cv = scores_svm_cv.mean()
print("SVM Mean Accuracy (Cross-validation - Lemmatization & Stemming):",␣
 ↪mean_accuracy_svm_cv)

# Entrenar el algoritmo Naive Bayes para cross-validation
nb_tfidf_cv = MultinomialNB()
```

```
# Realizar cross-validation con 5 folds
scores_nb_cv = cross_val_score(nb_tfidf_cv, tfidf_matrix_cv, etiquetas,␣
 ↪cv=cv_stratified)

# Obtener el rendimiento promedio para cross-validation
mean_accuracy_nb_cv = scores_nb_cv.mean()
print("Naive Bayes Mean Accuracy (Cross-validation - Lemmatization & Stemming):
 ↪", mean_accuracy_nb_cv)
```

```
SVM Mean Accuracy (Cross-validation - Lemmatization & Stemming):
0.30000000000000004
Naive Bayes Mean Accuracy (Cross-validation - Lemmatization & Stemming):
0.24000000000000002
```

[5]:
```
# 2. Identificar la cantidad de palabras de cada texto
word_counts = [len(text.split()) for text in dataClean]
print("Cantidad de palabras por texto:", word_counts)
```

```
Cantidad de palabras por texto: [1874, 425, 772, 551, 395, 547, 698, 687, 464,
562, 322, 1867, 562, 348, 669, 445, 366, 625, 510, 2000, 489, 628, 634, 367,
1366, 250, 369, 338, 1592, 1073, 261, 401, 702, 534, 395, 326, 303, 462, 536,
2000, 762, 530, 430, 783, 938, 281, 1036, 398, 776, 982]
```

[6]:
```
# 3. Identificar los promedios y su desviación estándar
average_word_count = sum(word_counts) / len(word_counts)
std_dev_word_count = pd.Series(word_counts).std()
print("Promedio de palabras por texto:", average_word_count)
print("Desviación estándar de palabras por texto:", std_dev_word_count)
```

```
Promedio de palabras por texto: 692.62
Desviación estándar de palabras por texto: 458.9511507508291
```

[7]:
```
# 4. Identificar las 500 palabras más usadas por cada documento y guardar en un␣
 ↪archivo
vectorizer = CountVectorizer(max_features=500)
word_matrix = vectorizer.fit_transform(dataClean)
feature_names = vectorizer.get_feature_names_out()
df_word_matrix = pd.DataFrame(word_matrix.toarray(), columns=feature_names)
df_word_matrix.to_csv('top500.csv', index=False)
```

[8]:
```
# 5. Imprimir las palabras más usadas y sus frecuencias
top_words = df_word_matrix.sum().nlargest(10)
print("Palabras más usadas:")
for word, frequency in top_words.items():
    print(f"{word}: {frequency}")
```

```
Palabras más usadas:
habilidades: 1020
```

```
trabajo: 367
profesionales: 325
laboral: 244
profesional: 234
ser: 202
capacidad: 186
competencias: 185
empresas: 182
habilidad: 157
```

[9]:
```
# 6. Elaborar un análisis de correlación
cosine_similarities = cosine_similarity(tfidf_matrix)
df_similarity = pd.DataFrame(cosine_similarities, columns=etiquetas,␣
 ↪index=etiquetas)
correlation_threshold = 0.5
related_documents = df_similarity[df_similarity > correlation_threshold].
 ↪stack().reset_index()
related_documents.columns = ['Documento 1', 'Documento 2', 'Correlación']
print("Documentos relacionados:")
print(related_documents)
```

```
Documentos relacionados:
    Documento 1 Documento 2  Correlación
0             0           0     1.000000
1             1           1     1.000000
2             2           2     1.000000
3             3           3     1.000000
4             4           4     1.000000
5             0           0     1.000000
6             1           1     1.000000
7             2           2     1.000000
8             3           3     1.000000
9             4           4     1.000000
10            0           0     1.000000
11            1           1     1.000000
12            2           2     1.000000
13            3           3     1.000000
14            4           4     1.000000
15            0           0     1.000000
16            1           1     1.000000
17            2           2     1.000000
18            3           3     1.000000
19            4           4     1.000000
20            0           0     1.000000
21            1           1     1.000000
22            2           2     1.000000
23            3           3     1.000000
24            4           4     1.000000
```

```
25          4           3     0.628321
26          0           0     1.000000
27          1           1     1.000000
28          2           2     1.000000
29          3           4     0.628321
30          3           3     1.000000
31          4           4     1.000000
32          0           0     1.000000
33          1           1     1.000000
34          2           2     1.000000
35          3           3     1.000000
36          4           4     1.000000
37          0           0     1.000000
38          1           1     1.000000
39          2           2     1.000000
40          3           3     1.000000
41          4           4     1.000000
42          0           0     1.000000
43          1           1     1.000000
44          2           2     1.000000
45          3           3     1.000000
46          4           4     1.000000
47          0           0     1.000000
48          1           1     1.000000
49          2           2     1.000000
50          3           3     1.000000
51          4           4     1.000000
```

[15]:
```python
# 7. Comparar las palabras más comunes entre documentos relacionados
df_similarity_reset = df_similarity.reset_index(drop=True)

# Seleccionamos las palabras mas comunes entre los documentos relacionados
related_word_matrix = df_word_matrix.loc[df_similarity_reset.index, :]
common_words = related_word_matrix.sum(axis=0)
print("Palabras más comunes entre documentos relacionados:")
print(common_words)
```

```
Palabras más comunes entre documentos relacionados:
10         41
2019       22
2020       30
2022       29
abierta    17
           ..
xxi        25
ámbito     15
área       26
áreas      34
```

```
éxito      58
Length: 500, dtype: int64
```

[12]:
```
# 8. Establece el término de frecuencia inverso
transformer = TfidfTransformer()
tfidf_matrix_transformed = transformer.fit_transform(word_matrix)
chi2_stat, p_val = chi2(tfidf_matrix_transformed, etiquetas)

log_odds_ratio = pd.Series(chi2_stat / tfidf_matrix_transformed.sum(axis=0).A1,␣
 ↪index=feature_names)
top_words_log_odds_ratio = log_odds_ratio.nlargest(10)
print("Top 10 palabras no relacionadas entre textos:")
for word, log_odds in top_words_log_odds_ratio.items():
    print(f"{word}: {log_odds}")
```

```
Top 10 palabras no relacionadas entre textos:
inserción: 4.000000000000001
issn: 4.000000000000001
seo: 4.000000000000001
il: 4.0
lector: 4.0
licencia: 4.0
operaciones: 3.4090653494354144
ocupaciones: 2.513061054361769
participantes: 2.3864507481614927
solución: 2.323461109606844
```

[16]:
```
# 9. Establecer el término de frecuencia inverso y encontrar términos con mayor␣
 ↪relevancia
vectorizer_tfidf_relevance = TfidfVectorizer()
tfidf_matrix_relevance = vectorizer_tfidf_relevance.
 ↪fit_transform(textos_stemmed_and_lemmatized)
relevance_scores = pd.DataFrame(tfidf_matrix_relevance.toarray(),␣
 ↪columns=vectorizer_tfidf_relevance.get_feature_names_out())
top_relevant_terms = relevance_scores.mean().nlargest(10)
print("Top 10 términos con mayor relevancia:")
for term, relevance in top_relevant_terms.items():
    print(f"{term}: {relevance}")
```

```
Top 10 términos con mayor relevancia:
habilidad: 0.2189829837193732
profesional: 0.08005903099956324
empresa: 0.07879899395222913
trabajo: 0.07651961370323913
competencia: 0.0582216953623542
capacidad: 0.05681503834733689
labor: 0.054950303587101096
profesion: 0.05169650201986266
```

```
persona: 0.050893040238293
ser: 0.043798745676146805
```