Procesamiento de Información 2023

Unidad 8 - Tarea

Desempeño de los clasificadores

David Aarón Ramírez Olmeda

Introducción

En esta tarea evaluaremos el desempeño de dos clasificadores. Los clasificadores seleccionados son Naive Bayes y Support Vector Machine (lineal). Utilizaremos dos técnicas de pesado diferentes, CountVectorizer y TfidfVectorizer, para preprocesar los datos. Realizaremos la evaluación en dos conjuntos de datos diferentes: datos de agresividad (2 clases) y datos de celulares (4 clases).

Desarrollo

A continuación se presenta el código utilizado para crear los modelos y evaluar su desempeño:

In [1]:
```
import os
import re
import json
import unicodedata
import numpy as np
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.stem.snowball import SnowballStemmer
from sklearn.feature_extraction.text import CountVectorizer, TfidfVectori
from sklearn import svm
from sklearn.metrics import classification_report, confusion_matrix
import matplotlib.pyplot as plt
from sklearn.naive_bayes import MultinomialNB
from sklearn.svm import LinearSVC
```

In [2]:
```python
# Preprocesamiento
stop_words = set(stopwords.words("spanish"))
stemmer = SnowballStemmer("spanish")

def preprocess(text):
    text = re.sub(r'[^\w\s]', '', text)
    # Eliminar símbolos duplicados (2 consecutivos)
    text = re.sub(r'(\w)\1+', r'\1\1', text)
    text = text.lower()
    tokens = nltk.word_tokenize(text)
    tokens = [token for token in tokens if token not in stop_words]
    tokens = [stemmer.stem(token) for token in tokens]
    return " ".join(tokens)
```

In [3]:
```python
# Carga de datos
def cargar_datos(archivo):
    datos = []
    with open(archivo) as f:
        for line in f:
            datos.append(json.loads(line))
    return pd.DataFrame(datos)
```

In [4]:
```python
# Cargar datos de agresividad
agresividad_train = cargar_datos("AggressivenessDetection_train.json")
agresividad_test = cargar_datos("AggressivenessDetection_test.json")

# Cargar datos de celulares
celulares_train = cargar_datos("cellphones_train.json")
celulares_test = cargar_datos("cellphones_test.json")
```

In [5]:
```python
def preprocess_data(train_data, test_data, vectorizer_type):
    # Obtener las columnas de texto
    text_train = train_data['text']
    text_test = test_data['text']

    # Crear el vectorizador y ajustarlo dependiendo del elegido
    if vectorizer_type == 'count':
        vectorizer = CountVectorizer()
    elif vectorizer_type == 'tfidf':
        vectorizer = TfidfVectorizer()
    else:
        raise ValueError("'count' o 'tfidf'")

    vectorizer.fit(text_train)

    # Transformar los datos de entrenamiento y prueba
    X_train = vectorizer.transform(text_train)
    X_test = vectorizer.transform(text_test)

    # Obtener las etiquetas
    y_train = train_data['klass']
    y_test = test_data['klass']

    return X_train, X_test, y_train, y_test
```

Hasta este punto es importante recalcar que la mayoría de lo anterior presentado ya lo habíamos desarrollado en tareas anteriores

Ahora podemos utilizar la función anterior para preprocesar los datos de agresividad y celulares utilizando ambos vectorizadores.

In [6]:
```python
# Preprocesar datos de agresividad
agresividad_X_train_count, agresividad_X_test_count, agresividad_y_train,
agresividad_X_train_tfidf, agresividad_X_test_tfidf, _, _ = preprocess_da

# Preprocesar datos de celulares
celulares_X_train_count, celulares_X_test_count, celulares_y_train, celul
celulares_X_train_tfidf, celulares_X_test_tfidf, _, _ = preprocess_data(c
```

Después vamos a crear y entrenar los modelos de Naive Bayes y Support Vector Machine (lineal) utilizando las configuraciones preprocesadas. También podemos hacer predicciones en los conjuntos de prueba y evaluar su rendimiento.

Agresividad

In [7]:
```
# Crear modelos
nb_count = MultinomialNB()
nb_tfidf = MultinomialNB()
svm_count = LinearSVC()
svm_tfidf = LinearSVC()

# Entrenar modelos de Naive Bayes
nb_count.fit(agresividad_X_train_count, agresividad_y_train)
nb_tfidf.fit(agresividad_X_train_tfidf, agresividad_y_train)

# Entrenar modelos de Support Vector Machine
svm_count.fit(agresividad_X_train_count, agresividad_y_train)
svm_tfidf.fit(agresividad_X_train_tfidf, agresividad_y_train)

# Hacer predicciones para los conjuntos de prueba con Naive Bayes y Count
nb_count_predictions = nb_count.predict(agresividad_X_test_count)

# Hacer predicciones para los conjuntos de prueba con Naive Bayes y Tfidf
nb_tfidf_predictions = nb_tfidf.predict(agresividad_X_test_tfidf)

# Hacer predicciones para los conjuntos de prueba con SVM y CounterVector
svm_count_predictions = svm_count.predict(agresividad_X_test_count)

# Hacer predicciones para los conjuntos de prueba con SVM y TfidfVectoriz
svm_tfidf_predictions = svm_tfidf.predict(agresividad_X_test_tfidf)
```

Ahora, vamos a imprimir la matriz de confusión y calcular la precisión, recall y F-score para cada configuración.

In [8]:
```
# Matriz de confusión y métricas para Naive Bayes y CounterVectorizer
print("Naive Bayes con CounterVectorizer:")
print(confusion_matrix(agresividad_y_test, nb_count_predictions))
print(classification_report(agresividad_y_test, nb_count_predictions))

# Matriz de confusión y métricas para Naive Bayes y TfidfVectorizer
print("Naive Bayes con TfidfVectorizer:")
print(confusion_matrix(agresividad_y_test, nb_tfidf_predictions))
print(classification_report(agresividad_y_test, nb_tfidf_predictions))

# Matriz de confusión y métricas para SVM y CounterVectorizer
print("SVM con CounterVectorizer:")
print(confusion_matrix(agresividad_y_test, svm_count_predictions))
print(classification_report(agresividad_y_test, svm_count_predictions))

# Matriz de confusión y métricas para SVM y TfidfVectorizer
print("SVM con TfidfVectorizer:")
print(confusion_matrix(agresividad_y_test, svm_tfidf_predictions))
print(classification_report(agresividad_y_test, svm_tfidf_predictions))
```

```
Naive Bayes con CounterVectorizer:
[[1485   80]
 [ 316  315]]
              precision    recall  f1-score   support

           0       0.82      0.95      0.88      1565
           1       0.80      0.50      0.61       631

    accuracy                           0.82      2196
   macro avg       0.81      0.72      0.75      2196
weighted avg       0.82      0.82      0.81      2196

Naive Bayes con TfidfVectorizer:
[[1562    3]
 [ 565   66]]
              precision    recall  f1-score   support

           0       0.73      1.00      0.85      1565
           1       0.96      0.10      0.19       631

    accuracy                           0.74      2196
   macro avg       0.85      0.55      0.52      2196
weighted avg       0.80      0.74      0.66      2196

SVM con CounterVectorizer:
[[1401  164]
 [ 230  401]]
              precision    recall  f1-score   support

           0       0.86      0.90      0.88      1565
           1       0.71      0.64      0.67       631

    accuracy                           0.82      2196
   macro avg       0.78      0.77      0.77      2196
weighted avg       0.82      0.82      0.82      2196

SVM con TfidfVectorizer:
[[1410  155]
 [ 218  413]]
              precision    recall  f1-score   support

           0       0.87      0.90      0.88      1565
           1       0.73      0.65      0.69       631

    accuracy                           0.83      2196
   macro avg       0.80      0.78      0.79      2196
weighted avg       0.83      0.83      0.83      2196
```

Analogamente, para el otro conjunto de datos...

Celulares

In [9]:
```
# Crear modelos
nb_count = MultinomialNB()
nb_tfidf = MultinomialNB()
svm_count = LinearSVC()
svm_tfidf = LinearSVC()

# Entrenar modelos de Naive Bayes
nb_count.fit(celulares_X_train_count, celulares_y_train)
nb_tfidf.fit(celulares_X_train_tfidf, celulares_y_train)

# Entrenar modelos de Support Vector Machine
svm_count.fit(celulares_X_train_count, celulares_y_train)
svm_tfidf.fit(celulares_X_train_tfidf, celulares_y_train)

# Hacer predicciones para los conjuntos de prueba con Naive Bayes y Count
nb_count_predictions = nb_count.predict(celulares_X_test_count)

# Hacer predicciones para los conjuntos de prueba con Naive Bayes y Tfidf
nb_tfidf_predictions = nb_tfidf.predict(celulares_X_test_tfidf)

# Hacer predicciones para los conjuntos de prueba con SVM y CounterVector
svm_count_predictions = svm_count.predict(celulares_X_test_count)

# Hacer predicciones para los conjuntos de prueba con SVM y TfidfVectoriz
svm_tfidf_predictions = svm_tfidf.predict(celulares_X_test_tfidf)
```

In [10]:
```
# Matriz de confusión y métricas para Naive Bayes y CounterVectorizer
print("Naive Bayes con CounterVectorizer:")
print(confusion_matrix(celulares_y_test, nb_count_predictions))
print(classification_report(celulares_y_test, nb_count_predictions))

# Matriz de confusión y métricas para Naive Bayes y TfidfVectorizer
print("Naive Bayes con TfidfVectorizer:")
print(confusion_matrix(celulares_y_test, nb_tfidf_predictions))
print(classification_report(celulares_y_test, nb_tfidf_predictions))

# Matriz de confusión y métricas para SVM y CounterVectorizer
print("SVM con CounterVectorizer:")
print(confusion_matrix(celulares_y_test, svm_count_predictions))
print(classification_report(celulares_y_test, svm_count_predictions))

# Matriz de confusión y métricas para SVM y TfidfVectorizer
print("SVM con TfidfVectorizer:")
print(confusion_matrix(celulares_y_test, svm_tfidf_predictions))
print(classification_report(celulares_y_test, svm_tfidf_predictions))
```

```
Naive Bayes con CounterVectorizer:
[[3121    7   75   19]
 [ 277   48   37    6]
 [ 482    1  305    8]
 [ 355    1   38  125]]
              precision    recall  f1-score   support

 information       0.74      0.97      0.84      3222
    negative       0.84      0.13      0.23       368
     neutral       0.67      0.38      0.49       796
    positive       0.79      0.24      0.37       519

    accuracy                           0.73      4905
   macro avg       0.76      0.43      0.48      4905
weighted avg       0.74      0.73      0.69      4905

Naive Bayes con TfidfVectorizer:
[[3218    0    4    0]
 [ 364    3    1    0]
 [ 624    0  172    0]
 [ 462    0    0   57]]
              precision    recall  f1-score   support

 information       0.69      1.00      0.82      3222
    negative       1.00      0.01      0.02       368
     neutral       0.97      0.22      0.35       796
    positive       1.00      0.11      0.20       519

    accuracy                           0.70      4905
   macro avg       0.92      0.33      0.35      4905
weighted avg       0.79      0.70      0.62      4905

SVM con CounterVectorizer:
[[2790   95  212  125]
 [ 134  143   53   38]
 [ 279   47  408   62]
 [ 148   17   74  280]]
              precision    recall  f1-score   support

 information       0.83      0.87      0.85      3222
    negative       0.47      0.39      0.43       368
     neutral       0.55      0.51      0.53       796
    positive       0.55      0.54      0.55       519

    accuracy                           0.74      4905
   macro avg       0.60      0.58      0.59      4905
weighted avg       0.73      0.74      0.73      4905

SVM con TfidfVectorizer:
[[2901   84  156   81]
 [ 146  137   54   31]
 [ 322   34  400   40]
 [ 195   20   54  250]]
              precision    recall  f1-score   support

 information       0.81      0.90      0.85      3222
    negative       0.50      0.37      0.43       368
```

neutral	0.60	0.50	0.55	796
positive	0.62	0.48	0.54	519
accuracy			0.75	4905
macro avg	0.63	0.56	0.59	4905
weighted avg	0.74	0.75	0.74	4905

Conclusiones

Observamos que los resultados de la evaluación varían dependiendo del clasificador y del tipo de vectorizador utilizado. En el caso de los datos de agresividad, el clasificador Naive Bayes obtuvo buenos resultados tanto con CountVectorizer como con TfidfVectorizer, con altos valores de precisión, recall y F-score en ambas configuraciones. Por otro lado, el clasificador Support Vector Machine también mostró buen desempeño, aunque inferior al de Naive Bayes.

En cuanto a los datos de celulares, se observó un desempeño similar entre los clasificadores Naive Bayes y Support Vector Machine. Ambos clasificadores lograron buenos resultados, aunque el clasificador Naive Bayes tuvo un rendimiento ligeramente mejor en términos de precisión y F-score.

Los resultados demuestran la importancia de seleccionar adecuadamente el clasificador y el vectorizador según el conjunto de datos y el problema en cuestión ya que la evaluación de desempeño proporciona información valiosa para tomar decisiones sobre qué configuración utilizar en futuros análisis de clasificación de texto.