Anexo 1

Proyecto 1: Tokenization, Stimming and Lemmatization

Mg. Luis Felipe Bustamante Narváez

Tokenization

In [...
```
# Instalamos skitlearn
!pip install -U scikit-learn
```

In [...
```
import pandas as pd
```

In [...
```
# Creamos el dataframe para mostrar la base de datos
df = pd.read_csv('df_total.csv', encoding='UTF-8')
```

In [...
```
df
```

In [...
```
#Mostramos la estructura de los datos
df['news'][2]
```

In [...
```
# Separamos los datos en variable independiente y variable dependiente
# variable independiente
X = df['news']
#variable dependiente ( A predecir)
Y = df['Type']
```

In [...
```
#Mostramos la variable dependiente
df['Type'].value_counts()
```

In [...
```
from sklearn.model_selection import train_test_split
```

In [...
```
# Asignamos los datos de entrenamiento (80%) y prueba (20%)
X_train, X_test, Y_train, Y_test = train_test_split(X, Y, test_size=0.2)
```

In [...
```
#Mostramos los datos de entrenamiento  (1217datos*0.8 = 973)
X_train
```

In [...
```
#Mostramos los datos de prueba (1217datos*0.2 = 244)
X_test
```

In [...
```
#Mostramos los datos a predecir de entrenamiento  (1217datos*0.8 = 973)
Y_train
```

In [...
```
#Mostramos los datos a predecir de prueba  (1217datos*0.2 = 244)
Y_test
```

In [...
```
# Vectorizamos los datos
from sklearn.feature_extraction.text import CountVectorizer
vectorizer = CountVectorizer()
```

In [...
```
# vectorizamos los datos de entramiento
X_train_transformed = vectorizer.fit_transform(X_train)
# vectorizamos los datos de prueba
X_test_transformed = vectorizer.transform(X_test)
```

In [...
```
#Mostramos los datos transformados entrenamiento (de forma nativa)
X_train_transformed
```

In [...
```
#Mostramos los datos transformados entrenamiento
X_train_transformed_dense = X_train_transformed.toarray()
X_train_transformed_dense
```

In [...
```
#Mostramos los datos transformados prueba (de forma nativa)
X_test_transformed
```

In [...
```
#Mostramos los datos transformados prueba
X_test_transformed_dense = X_test_transformed.toarray()
X_test_transformed_dense
```

In [...
```
from sklearn.naive_bayes import MultinomialNB
from sklearn import metrics
```

In [...
```
# Creamos la variable para el modelo
model = MultinomialNB()
# Entrenamos el modelo con los datos de entrenamiento vectorizados y los valores reales dependientes
model.fit(X_train_transformed, Y_train)
#Creamos la predicción con los datos de prueba vectorizados
Y_predict = model.predict(X_test_transformed)
#Revisamos el porcentaje de predicción
accuracy = metrics.accuracy_score(Y_test, Y_predict)
print(accuracy)
```

Stimming

In [...
```
import nltk
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import SnowballStemmer
```

In [...
```
nltk.download('punkt')
nltk.download('stopwords')
```

In [...
```
stemmer = SnowballStemmer('spanish')
```

In [...
```
# Función para tokenizar y stimmar
def tokenize_and_stem(text):
    tokens = word_tokenize(text.lower())
    stems = [stemmer.stem(token) for token in tokens if token.isalpha()]
    return ' '.join(stems)

# Otra manera de mostrar esta función
"""
def tokenize_and_stem(text):
    # Tokenizar el texto y convertirlo a minúsculas
    tokens = word_tokenize(text.lower())

    # Aplicar stemming con un bucle for
    stems = []
    for token in tokens:
        if token.isalpha():  # Filtrar solo palabras (sin números ni símbolos)
            stem = stemmer.stem(token)  # Aplicar stemming
            stems.append(stem)  # Agregar a la lista

    # Unir los stems en un solo string y retornarlo
    return ' '.join(stems)
"""
```

In [...
```
# Agregamos una nueva columna al dataframe para mostrar los tokens
df['news_stemmer'] = df['news'].apply(tokenize_and_stem)
```

In [...
```
df
```

In [...
```
# Separamos los datos en variables de entrada y etiquetas
Xs = df['news_stemmer']
Ys = df['Type']
# Creamos el entrenamiento
Xs_train, Xs_test, Ys_train, Ys_test = train_test_split(Xs, Ys, test_size=0.2)
vectorizerS = CountVectorizer()
# Transformamos los documentos en vectores
Xs_train_transformed = vectorizerS.fit_transform(Xs_train)
Xs_test_transformed = vectorizerS.transform(Xs_test)
# Creamos el modelo
modelS = MultinomialNB()
# Entrenamos el modelo
modelS.fit(Xs_train_transformed, Ys_train)
# Medimos el rendimiento del modelo
Ys_predict = modelS.predict(Xs_test_transformed)
# Mostramos su eficiencia
acurracyS = metrics.accuracy_score(Ys_test, Ys_predict)
print(acurracyS)
```

Comparamos las dimensiones del texto original y los stimming

In [...
```
origin = X_train_transformed.shape
stimming = Xs_train_transformed.shape

# Opcional --- creamos un DataFrame para mejorar la visual y repasar
dfCompare = pd.DataFrame({
    'DataSet': ['X_train_transformed', 'Xs_train_transformed'],
    'Vectores': [origin[0], stimming[0]],
    'Dimensiones': [origin[1], stimming[1]]
})
dfCompare
```

Lemmatization

In [...
```
import spacy
```

In [...
```
nlp = spacy.load('es_core_news_sm')
```

In [...
```
# función de lemmatization
def lemmatize_text(text):
    doc = nlp(text.lower())
    lemmas = [token.lemma_ for token in doc if token.is_alpha]
    return ' '.join(lemmas)
```

In [...
```
# Agregamos una nueva columna al dataframe para mostrar los lemmas
df['news_lemma'] = df['news'].apply(lemmatize_text)
```

In [...
```
df['news_lemma'][5]
```

In [...
```
# Separamos los datos en variables de entrada y etiquetas
Xl = df['news_lemma']
Yl = df['Type']
# Creamos el entrenamiento
Xl_train, Xl_test, Yl_train, Yl_test = train_test_split(Xl, Yl, test_size=0.2)
vectorizerL = CountVectorizer()
```

```
# Transformamos los documentos en vectores
Xl_train_transformed = vectorizerL.fit_transform(Xl_train)
Xl_test_transformed = vectorizerL.transform(Xl_test)
# Creamos el modelo
modelL = MultinomialNB()
# Entrenamos el modelo
modelL.fit(Xl_train_transformed, Yl_train)
# Medimos el rendimiento del modelo
Yl_predict = modelL.predict(Xl_test_transformed)
# Mostramos su eficiencia
acurracyL = metrics.accuracy_score(Yl_test, Yl_predict)
print(acurracyL)
```

Comparamos las dimensiones del texto original y los stimming

In […

```
origin = X_train_transformed.shape
stimming = Xs_train_transformed.shape
lemmatizing = Xl_train_transformed.shape

# Opcional --- creamos un DataFrame para mejorar la visual y repasar
dfCompare = pd.DataFrame({
    'DataSet': ['X_train_transformed', 'Xs_train_transformed', 'Xl_train_transformed'],
    'Vectores': [origin[0], stimming[0], lemmatizing[0]],
    'Dimensiones': [origin[1], stimming[1], lemmatizing[1]]
})
dfCompare
```

Conclusiones

Se puede tomar la decisión de seleccionar el mejor modelo, aunque los datos son pocos, se puede observar una buena predicción, aunque en el texto se ve mejor la lemmatización, este requiere un gasto computacional más alto, que no se ve reflejado, para este ejemplo, en la eficacia del modelo.

Mg. Luis Felipe Bustamante Narváez