Aprendizaje Automático para la Gestión de Datos Masivos

Dr. Paulo López Meyer

Análisis de texto

Itzel Rubi Alcala Gil 2930147

A continuación, voy a cargar y analizar un archivo que contiene una base de datos de encabezados de noticias y generar un modelo para clasificar estas noticias en 4 categorías: Negocios, Ciencia y Tecnología, Entretenimiento y Salud.

Voy a comenzar por instalar las bibliotecas necesarias para este procesamiento de texto, modelado de documentos y aprendizaje automático.

```
✓ 2s [1]  pip install gensim sklearn pandas nltk
```

Se importan las bibliotecas necesarias. pandas para manipulación de datos, nltk para procesamiento de lenguaje natural, gensim para modelado de documentos y string para operaciones con cadenas.

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from gensim.models.doc2vec import TaggedDocument
import string
```

Después, se descargan los recursos necesarios de NLTK para la tokenización y las stopwords, cargo el conjunto de datos y se establecen los nombres de las columnas y filtro solo las categorías de interés: Negocios, Ciencia y Tecnología, Entretenimiento y Salud.

```
import pandas as pd
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from gensim.models.doc2vec import TaggedDocument
import string

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

# Cargar el conjunto de datos
data = pd.read_csv(r'/content/newsCorpora-trimmed.csv')
data.columns = ['category', 'text']

# Filtrar solo las categorías de interés
categories = ['b', 't', 'e', 'm']  # Business, Science and Technology, Entertainment, Health
data = data[data['category'].isin(categories)]
```

Ahora, vamos a definir una función para preprocesar el texto, que incluye convertir a minúsculas, tokenizar, eliminar stopwords y palabras no alfabéticas y a crear documentos etiquetados necesarios para entrenar el modelo Doc2Vec.

Las stopwords (palabras vacías) en inglés, que son palabras comunes que normalmente no aportan mucho significado en el análisis (por ejemplo, "and", "the", "is").

```
# Preprocesamiento de texto
stop_words = set(stopwords.words('english'))
def preprocess_text(text):
    tokens = word_tokenize(text.lower())
    tokens = [word for word in tokens if word.isalpha() and word not in stop_words]
    return tokens

data['tokens'] = data['text'].apply(preprocess_text)

# Crear documentos etiquetados
tagged_data = [TaggedDocument(words=row['tokens'], tags=[row['category']]) for index, row in data.iterrows()]
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
[nltk_data] Downloading package stopwords to /root/nltk_data...
[nltk_data]   Unzipping corpora/stopwords.zip.
```

Entrenamos un modelo Doc2Vec con los documentos etiquetados y se guarda el modelo entrenado.

El modelo Doc2Vec requiere que los documentos sean etiquetados. Esto se realiza utilizando TaggedDocument de Gensim, donde cada documento es una lista de palabras (tokens) y se le asigna una etiqueta (categoría). Este formato permite que el modelo aprenda las representaciones vectoriales de los documentos en función de sus palabras y etiquetas.

La lista tagged_data contiene todos los documentos etiquetados que se usarán para entrenar el modelo, como podemos ver en el siguiente print.

```
from gensim.models import Doc2Vec

print(tagged_data)

# Entrenar el modelo Doc2Vec
model = Doc2Vec(tagged_data, vector_size=100, window=5, min_count=5, workers=4, epochs=20)

# Guardar el modelo
model.save("doc2vec_model")
```

```
[TaggedDocument(words=['fed', 'official', 'says', 'weak', 'data', 'caused', 'weather', 'slow', 'taper'], tags=['b']), TaggedDocument(words=['fed
```

Ya que tenemos este paso concluido, se crea una función para vectorizar los documentos utilizando el modelo Doc2Vec entrenado.

Nuestro modelo cuenta con los siguientes parámetros

- Vector: Especifica el tamaño del vector que representará cada documento. Un vector más grande puede capturar más información, pero también requiere más recursos computacionales.
- Window: Define el tamaño de la ventana de contexto para el modelo. En este caso, el modelo considera un contexto de 5 palabras a cada lado de la palabra objetivo durante el entrenamiento.
- Min Count: Ignora todas las palabras con una frecuencia total menor que este valor. Ayuda a eliminar palabras raras que no son útiles para el aprendizaje del modelo.
- Workers: Especifica el número de hilos a usar durante el entrenamiento. Más hilos pueden acelerar el entrenamiento.
- Epochs: El número de iteraciones (épocas) sobre el conjunto de entrenamiento. Más épocas pueden permitir que el modelo aprenda mejor, pero también pueden llevar a sobreajuste.

```
from sklearn.model_selection import train_test_split
from sklearn.svm import SVC
from sklearn.metrics import accuracy_score

# Convertir los documentos etiquetados a vectores
def vectorize_doc(doc):
    return model.infer_vector(doc.words)

data['vector'] = data['tokens'].apply(lambda x: vectorize_doc(TaggedDocument(x, [0])))

# Crear el conjunto de entrenamiento y prueba
X = list(data['vector'])
y = data['category']
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)

# Entrenar el clasificador SVM
classifier = SVC(kernel='linear')
classifier.fit(X_train, y_train)
```

Se define una función vectorize_doc para inferir el vector de un nuevo documento utilizando el modelo Doc2Vec entrenado. Este vector captura la representación semántica del documento.

Se aplica la función vectorize_doc a cada documento en el conjunto de datos para convertirlos en vectores. Esto facilita su uso en modelos de aprendizaje automático.

Y ahora, vamos a dividir nuestros datos en entrenamiento y prueba para entrenar un modelo de clasificación SVM utilizando los vectores de documentos.

El porcentaje de accuracy que nos arrojó este modelo es el siguiente.

```
# Evaluar el clasificador
y_pred = classifier.predict(X_test)
print(f"Accuracy: {accuracy_score(y_test, y_pred)}")
```

```
Accuracy: 0.6636992257049398
```

Ahora, voy a definir una función para clasificar nuevos documentos y a probarla con un ejemplo.

```
def classify_new_document(text):
    tokens = preprocess_text(text)
    vector = vectorize_doc(TaggedDocument(tokens, [0]))
    return classifier.predict([vector])[0]

# Ejemplo de clasificación
new_document = "New breakthrough in cancer research"
predicted_category = classify_new_document(new_document)
print(f"Predicted category: {predicted_category}")
```

```
Predicted category: m
```

Podemos ver, que el encabezado que le di “New breakthrough in cancer research”, fue clasificado en la categoría “m”, la cuál corresponde a la salud. Por lo cual nuestro modelo SVM funcionó correctamente en este ejemplo.

En conclusión, el modelo Doc2Vec es una herramienta poderosa para representar documentos como vectores en un espacio semántico, este código realiza un flujo completo de preprocesamiento de texto, entrenamiento de un modelo de representación de documentos Doc2Vec y entrena un clasificador SVM para categorizar nuevos documentos con precisión.