ACTIVIDAD #1:
Ejercicio: estructura del tema de clasificación con Python por pasos:

A continuación se detalla la estructura del tema de clasificación con Python:
Importación de bibliotecas:

En esta parte, se importan las bibliotecas necesarias para el desarrollo del sistema de clasificación. Estas incluyen funciones para obtener conjuntos de datos de textos, herramientas para el procesamiento de texto, algoritmos de clasificación y métricas para evaluar el rendimiento del modelo.

```
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report
```

tObtención de datos:

En esta sección, se obtienen los datos de entrenamiento y prueba. En este ejemplo, se utiliza el conjunto de datos "20 Newsgroups" de scikit-learn, que contiene documentos de diferentes categorías.

```
categories = ['sci.space', 'comp.graphics', 'rec.sport.baseball']
newsgroups_train = fetch_20newsgroups(subset='train', categories=categories)
newsgroups_test = fetch_20newsgroups(subset='test', categories=categories)
```

Preprocesamiento de datos:

Aquí se lleva a cabo el preprocesamiento de los datos de texto. Se utiliza un vectorizador TF-IDF para convertir los documentos de texto en vectores numéricos que pueden ser entendidos por el algoritmo de clasificación.

```
vectorizer = TfidfVectorizer()
X_train = vectorizer.fit_transform(newsgroups_train.data)
X_test = vectorizer.transform(newsgroups_test.data)
```

```
y_train = newsgroups_train.target
y_test = newsgroups_test.target
```

Entrenamiento del clasificador:
En esta parte, se entrena el modelo de clasificación utilizando el algoritmo Naive Bayes Multinomial, que es adecuado para datos de conteo como los vectores TF-IDF.

```
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
```

Predicción y evaluación:
Se utilizan los datos de prueba para hacer predicciones utilizando el modelo entrenado y se evalúa el rendimiento del clasificador utilizando métricas como la exactitud y un informe de clasificación detallado.

```
y_pred = classifier.predict(X_test)

accuracy = accuracy_score(y_test, y_pred)
print("Exactitud del clasificador: {:.2f}%".format(accuracy * 100))
print("\nInforme de clasificación:")
print(classification_report(y_test, y_pred, target_names=newsgroups_test.target_names))
```

Lo anterior deberá imprimir en pantalla lo siguiente:

Exactitud del clasificador: 95.59%
Informe de clasificación:

	precision	recall	f1-score	support
comp.graphics	0.98	0.90	0.94	389
rec.sport.baseball	0.96	0.99	0.98	397
sci.space	0.93	0.98	0.95	394
accuracy			0.96	1180
macro avg	0.96	0.96	0.96	1180
weighted avg	0.96	0.96	0.96	1180

El fragmento de código proporciona una evaluación del rendimiento de un sistema de clasificación basado en textos. Sistemas de clasificación basados en textos, la exactitud del clasificador representa la capacidad del sistema para realizar predicciones correctas sobre la categorización de documentos o textos en diferentes clases. En este caso, el sistema alcanza una exactitud del 95.59%, lo que sugiere una alta eficacia en la clasificación de documentos.

El informe de clasificación proporciona métricas detalladas sobre cómo el sistema clasifica cada clase de la tarea. Estas métricas, como precisión, recall y F1-score, son para comprender el rendimiento del sistema en la identificación precisa de cada categoría de texto. Por ejemplo, una alta precisión indica que el sistema clasifica correctamente la mayoría de los documentos como pertenecientes a una clase específica, mientras que un alto recall sugiere que el sistema identifica la mayoría de los documentos relevantes de esa clase.

Los clasificadores basados en reglas son sistemas simples que utilizan reglas predefinidas o patrones de texto para asignar documentos a categorías específicas. Aunque son fáciles de implementar, pueden carecer de flexibilidad y precisión en comparación con otros enfoques más avanzados. Por otro lado, los clasificadores de aprendizaje automático utilizan algoritmos de aprendizaje automático, como Naive Bayes, Máquinas de Vectores de Soporte (SVM), y árboles de decisión, para aprender patrones en los datos y realizar clasificaciones automáticas. Estos sistemas suelen ofrecer una mayor precisión y flexibilidad.

Los clasificadores basados en modelos de lenguaje pre-entrenados, como BERT o Word2Vec, utilizan representaciones semánticas ricas del texto previamente aprendidas en grandes cantidades de datos para clasificar documentos. Estos modelos pueden ser finamente ajustados o utilizados como características para la clasificación. Por último, los clasificadores basados en técnicas de aprendizaje profundo emplean arquitecturas avanzadas, como redes neuronales convolucionales (CNN) o recurrentes (RNN), para aprender representaciones de texto altamente abstractas y complejas, lo que puede resultar en clasificaciones más precisas y robustas.

```
# Paso 1: Importar las bibliotecas necesarias
print("Paso 1: Importar las bibliotecas necesarias")
import numpy as np
import nltk
from nltk.corpus import movie_reviews
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.model_selection import train_test_split
from sklearn.svm import LinearSVC
from sklearn.metrics import accuracy_score, classification_report
print("Bibliotecas importadas correctamente.\n")
```

```
# Paso 2: Cargar el conjunto de datos
print("Paso 2: Cargar el conjunto de datos")
nltk.download('movie_reviews')
documents = [(movie_reviews.raw(fileid), category)
        for category in movie_reviews.categories()
        for fileid in movie_reviews.fileids(category)]
X, y = zip(*documents)
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=42)
print("Conjunto de datos cargado correctamente.\n")

# Paso 3: Preprocesamiento de texto con NLTK
print("Paso 3: Preprocesamiento de texto con NLTK")
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

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

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
 tokens = word_tokenize(text)
```

```
tokens = [word.lower() for word in tokens if word.isalpha()]
 tokens = [word for word in tokens if word not in stop_words]
 tokens = [lemmatizer.lemmatize(word) for word in tokens]
 return ' '.join(tokens)

X_train = [preprocess_text(text) for text in X_train]
X_test = [preprocess_text(text) for text in X_test]
print("Texto preprocesado correctamente.\n")

# Paso 4: Vectorizar los textos
print("Paso 4: Vectorizar los textos")
vectorizer = TfidfVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
print("Textos vectorizados correctamente.\n")

# Paso 5: Entrenar el clasificador
print("Paso 5: Entrenar el clasificador")
classifier = LinearSVC()
classifier.fit(X_train, y_train)
print("Clasificador entrenado correctamente.\n")
```

```
# Paso 6: Realizar predicciones
print("Paso 6: Realizar predicciones")
y_pred = classifier.predict(X_test)
print("Predicciones realizadas correctamente.\n")

# Paso 7: Evaluar el rendimiento del clasificador
print("Paso 7: Evaluar el rendimiento del clasificador")
accuracy = accuracy_score(y_test, y_pred)
print("Exactitud del clasificador:", accuracy)

print("\nInforme de clasificación:")
print(classification_report(y_test, y_pred,
target_names=movie_reviews.categories()))
```

Este script utiliza NLTK para realizar el preprocesamiento de texto, que incluye tokenización, eliminación de palabras vacías (stopwords), y lematización. Luego, se vectorizan los textos utilizando TF-IDF y se entrena un clasificador SVM. Se evalúa el rendimiento del clasificador en el conjunto de datos de prueba. Cada paso se imprime en la consola antes de ejecutarse para seguir el flujo del proceso paso a paso.

Otro caso puede ser el de Bayes (Naive Bayes) para clasificar reseñas como positivas o negativas

```
# Paso 1: Importar las bibliotecas necesarias
print("Paso 1: Importar las bibliotecas necesarias")
import numpy as np
import nltk
from nltk.corpus import movie_reviews
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import MultinomialNB
from sklearn.metrics import accuracy_score, classification_report, confusion_matrix
import matplotlib.pyplot as plt
import seaborn as sns
print("Bibliotecas importadas correctamente.\n")

# Paso 2: Cargar el conjunto de datos
print("Paso 2: Cargar el conjunto de datos")
nltk.download('movie_reviews')
documents = [(movie_reviews.raw(fileid), category)
             for category in movie_reviews.categories()
             for fileid in movie_reviews.fileids(category)]
X, y = zip(*documents)
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=42)
print("Conjunto de datos cargado correctamente.\n")
```

```
# Paso 3: Preprocesamiento de texto con NLTK
print("Paso 3: Preprocesamiento de texto con NLTK")
from nltk.tokenize import word_tokenize
from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer

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

stop_words = set(stopwords.words('english'))
lemmatizer = WordNetLemmatizer()

def preprocess_text(text):
 tokens = word_tokenize(text)
  tokens = [word.lower() for word in tokens if word.isalpha()]
  tokens = [word for word in tokens if word not in stop_words]
  tokens = [lemmatizer.lemmatize(word) for word in tokens]
 return ' '.join(tokens)
```

```
tX_train = [preprocess_text(text) for text in X_train]
X_test = [preprocess_text(text) for text in X_test]
print("Texto preprocesado correctamente.\n")

# Paso 4: Vectorizar los textos
print("Paso 4: Vectorizar los textos")
vectorizer = CountVectorizer(max_features=5000)
X_train = vectorizer.fit_transform(X_train)
X_test = vectorizer.transform(X_test)
print("Textos vectorizados correctamente.\n")


# Paso 5: Entrenar el clasificador de Bayes Ingenuo
print("Paso 5: Entrenar el clasificador de Bayes Ingenuo")
classifier = MultinomialNB()
classifier.fit(X_train, y_train)
print("Clasificador entrenado correctamente.\n")

# Paso 6: Realizar predicciones
print("Paso 6: Realizar predicciones")
y_pred = classifier.predict(X_test)
print("Predicciones realizadas correctamente.\n")
```

```
# Paso 7: Evaluar el rendimiento del clasificador
print("Paso 7: Evaluar el rendimiento del clasificador")
accuracy = accuracy_score(y_test, y_pred)
print("Exactitud del clasificador:", accuracy)

print("\nInforme de clasificación:")
print(classification_report(y_test, y_pred,
target_names=movie_reviews.categories()))

# Gráfico de barras para la distribución de clases
plt.figure(figsize=(8, 6))
sns.countplot(y_test)
plt.title('Distribución de clases en el conjunto de prueba')
plt.xlabel('Clases')
plt.ylabel('Cantidad')

plt.show()

# Matriz de confusión
plt.figure(figsize=(8, 6))
cm = confusion_matrix(y_test, y_pred)
sns.heatmap(cm, annot=True, fmt='d', cmap='Blues',
xticklabels=movie_reviews.categories(),
yticklabels=movie_reviews.categories())
```

```
plt.title('Matriz de Confusión')
plt.xlabel('Predicciones')
plt.ylabel('Valores reales')
plt.show()
```

Esto arroja en pantalla lo siguiente:

Paso 1: Importar las bibliotecas necesarias
Bibliotecas importadas correctamente.

Paso 2: Cargar el conjunto de datos
[nltk_data] Downloading package movie_reviews to /root/nltk_data...
[nltk_data] Package movie_reviews is already up-to-date!

Conjunto de datos cargado correctamente.

Paso 3: Preprocesamiento de texto con NLTK

[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data] Package punkt is already up-to-date!
[nltk_data] Downloading package stopwords to /root/nltk_data...

[nltk_data] Package stopwords is already up-to-date!
[nltk_data] Downloading package wordnet to /root/nltk_data...
[nltk_data] Package wordnet is already up-to-date!

Texto preprocesado correctamente.

Paso 4: Vectorizar los textos
Textos vectorizados correctamente.

Paso 5: Entrenar el clasificador de Bayes Ingenuo
Clasificador entrenado correctamente.

Paso 6: Realizar predicciones
Predicciones realizadas correctamente.

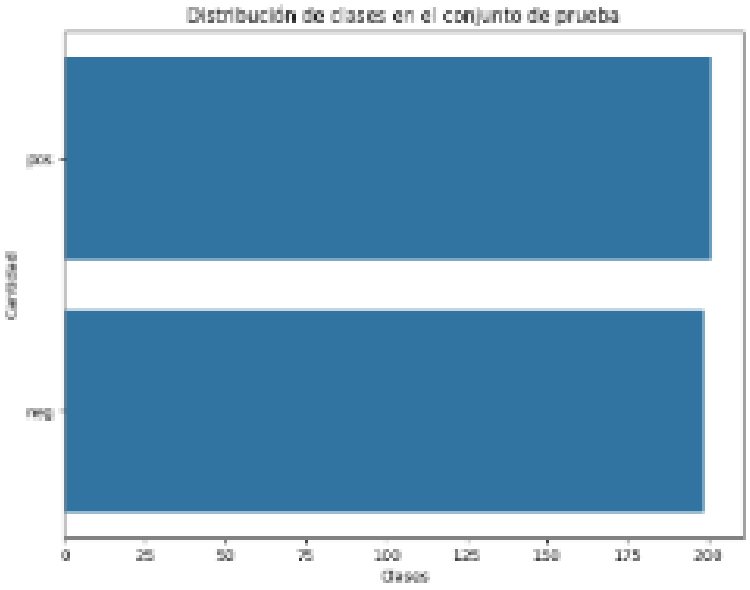
Paso 7: Evaluar el rendimiento del clasificador
Exactitud del clasificador: 0.8

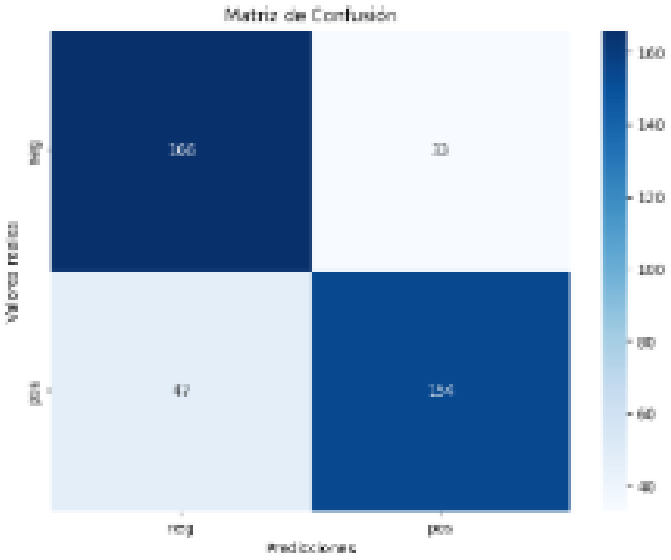
Informe de clasificación:

	precision	recall	f1-score	support
neg	0.78	0.83	0.81	199
pos	0.82	0.77	0.79	201
accuracy			0.80	400
macro avg	0.80	0.80	0.80	400

weighted avg 0.80 0.80 0.80 400

El gráfico que ilustra la distribución de estas clases se presenta a continuación, junto con su correspondiente matriz de confusión.





Esta matriz organiza las predicciones del modelo en función de las clases verdaderas, lo que permite identificar con claridad los aciertos y los errores del clasificador.

En el contexto de este análisis de sentimientos en reseñas de películas, la matriz de confusión se presenta en forma de una cuadrícula donde las filas representan las clases verdaderas y las columnas representan las clases predichas por el modelo. La diagonal principal de la matriz muestra los valores donde la clase predicha coincide con la clase verdadera, mientras que los valores fuera de la diagonal principal representan las predicciones incorrectas.

En este caso específico, se observa que en la diagonal principal de la matriz de confusión se encuentran los siguientes valores:

166: Correspondiente a las reseñas clasificadas correctamente como positivas.
154: Correspondiente a las reseñas clasificadas correctamente como negativas.
47: Correspondiente a las reseñas negativas que fueron clasificadas incorrectamente como positivas.
33: Correspondiente a las reseñas positivas que fueron clasificadas incorrectamente como negativas.