Universidad Autónoma de Aguascalientes

Centro de Ciencias Básicas

Departamento de Ciencias de la Computación

Optativa Profesionalizante II: Machine Learning y Deep Learning

10° "A"

Segunda Evaluación Parcial

Docente: Dr. Francisco Javier Luna Rosas

Alumno: Joel Alejandro Espinoza Sánchez (211800)

Fecha de Entrega: Aguascalientes, Ags., 1 de mayo del 2023.

El análisis de sentimientos, a veces también denominado minería de opiniones, es una conocida sub-disciplina del amplio campo del PLN (Procesamiento del Lenguaje Natural); está relacionado con el análisis de la polaridad de documentos. Una tarea popular en el análisis de sentimiento es la clasificación de documentos basados en las emociones u opiniones expresadas de los autores respecto a un tema en particular. El conjunto de datos de críticas de cine consiste en 50,000 críticas de cine polarizadas etiquetadas como negativas y como positivas. Aquí, positiva significa que una película ha sido clasificada con más de seis estrellas, mientras que negativa significa que una película ha sido clasificada con menos de cinco estrellas.

El alumno deberá elaborar un documento (`*.pdf`) y un archivo auto-reproducible (`*.html`) que analice, implemente y evalúe algoritmos de Deep Learning y Machine Learning para clasificar las críticas de cine. El documento deberá contener:

- Portada
- Evidencias del examen
- Conclusiones
- Referencias (formato APA)

---

a) Una explicación del preprocesamiento de datos para generar un formato adecuado de los datos

Primeramente se cargan los datos que se usarán:

In [1]:
```
import pandas as pd
import numpy as np
import tensorflow as tf
from sklearn.model_selection import train_test_split

data = pd.read_csv("movie_data.csv")
```

En este paso se suelen aplicar las siguientes estrategias para darle una mejor forma al conjunto de datos:

1. **Eliminar los datos irrelevantes:** Eliminar los datos que no son útiles para el análisis de opiniones, como las fechas, las direcciones, etc.
2. **Limpiar el texto:** Eliminar caracteres especiales, signos de puntuación, números y otros caracteres no alfabéticos que no aportan información útil para el análisis.
3. **Convertir el texto a minúsculas:** Convertir todo el texto a minúsculas para facilitar el procesamiento y para que no se distingan las palabras en mayúsculas y minúsculas.
4. **Eliminar las palabras comunes:** Eliminar las palabras comunes que no aportan información útil para el análisis, como artículos, preposiciones, conjunciones, etc.
5. **Lematización:** Lematizar el texto para reducir las palabras a su forma base, lo que puede ayudar a reducir la complejidad del análisis y aumentar la precisión.
6. **Eliminar palabras clave irrelevantes:** Eliminar palabras clave irrelevantes que no aportan información útil para el análisis.
7. **Tokenización:** Separar el texto en palabras individuales para su posterior análisis.
8. **Normalización:** Normalizar el texto para que todas las palabras estén en la misma forma, como eliminar los acentos o caracteres especiales.
9. **Análisis de sentimientos:** Realizar un análisis de sentimientos para asignar a cada opinión un valor de positividad, negatividad o neutralidad.

Se realiza un preprocesamiento con la librería NLTK para limpiar el texto, tokenizar, normalizar, eliminar las stopwords y pos_tagging con el siguiente código:

In [2]:
```
import nltk
from nltk.corpus import stopwords
from nltk.tokenize import word_tokenize
from nltk.stem import SnowballStemmer, WordNetLemmatizer
from nltk.tag import pos_tag
import re
nltk.download('averaged_perceptron_tagger')

def clean_text(text):
    # Eliminar caracteres especiales y números
    text = re.sub(r'[^A-Za-z\s]', '', text)
```

```
    # Convertir a minúsculas
    text = text.lower()
    return text

def tokenize(text):
    tokens = word_tokenize(text)
    return tokens

def normalize(tokens):
    # Stemming
    stemmer = SnowballStemmer('english')
    stemmed_tokens = [stemmer.stem(token) for token in tokens]
    # Lemmatization
    lemmatizer = WordNetLemmatizer()
    lemmatized_tokens = [lemmatizer.lemmatize(token) for token in stemmed_tokens]
    return lemmatized_tokens

def remove_stopwords(tokens):
    stop_words = set(stopwords.words('english'))
    filtered_tokens = [token for token in tokens if not token in stop_words]
    return filtered_tokens

def pos_tagging(tokens):
    pos_tokens = pos_tag(tokens)
    return pos_tokens

data['review'] = data['review'].apply(clean_text)
data['review'] = data['review'].apply(tokenize)
data['review'] = data['review'].apply(normalize)
data['review'] = data['review'].apply(remove_stopwords)
data['review'] = data['review'].apply(pos_tagging)

X = np.asarray(data['review'])
X = [' '.join([word[0] for word in sublist]) for sublist in X]
```

```
[nltk_data] Downloading package averaged_perceptron_tagger to
[nltk_data]     C:\Users\alexe\AppData\Roaming\nltk_data...
[nltk_data]   Package averaged_perceptron_tagger is already up-to-
[nltk_data]       date!
```

b) Una explicación del modelo bolsa de palabras o cualquier otro analizador ligústico aplicado al Dataset de críticas de cine

La librería Keras de TensorFlow proporciona la clase Tokenizer y la función pad_sequences para procesar texto y convertirlo en una representación numérica que puede ser utilizada como entrada para modelos de aprendizaje automático.

El modelo de bolsa de palabras que se usó fue dado por TensorFlow de su apartado de Keras siendo el método `Tokenizer` que se utiliza para convertir el texto en una secuencia de enteros (tokens) asignando un número único a cada palabra en el texto. Esto es útil porque los modelos de aprendizaje automático sólo pueden trabajar con datos numéricos, por lo que se debe convertir el texto en una forma numérica para poder ser procesado.

Su implementación se presenta a continuación:

In [3]:
```
from tensorflow.keras.preprocessing.text import Tokenizer
from tensorflow.keras.preprocessing.sequence import pad_sequences

tokenizer = Tokenizer(num_words=5000)
tokenizer.fit_on_texts(X)
```

c) Una explicación de la transformación de las palabras en vectores de características (utilice la frecuencia de termino - frecuencia inversa de documento "tf-idf" o cualquier otra técnica que permita transformar palabras a vectores de características)

Por otro lado, la transformación de palabras en vectores de características se usa nuevamente TensorFlow con el uso de Keras usando `pad_sequences` que se utiliza para igualar la longitud de las secuencias numéricas resultantes de Tokenizer. Dado que las secuencias de palabras en el texto pueden tener diferentes longitudes, es necesario igualarlas para poder procesarlas en el modelo de aprendizaje automático. pad_sequences se encarga de añadir ceros (o cualquier otro valor definido) al principio o al final de las secuencias para que todas tengan la misma longitud.

La implementación y su uso se presenta en el siguiente segmento:

In [4]:
```
X = tokenizer.texts_to_sequences(X)
X = pad_sequences(X, maxlen=200)
```

d) Una explicación del modelo CNN (Convoluciones 1D) para clasificar las críticas de cine. La precisión del modelo debe ser del 95% o mayor

A continuación se muestra la implementación del modelo CNN preparando los conjuntos de datos de aprendizaje supervisado y declarando la arquitectura de la red como se muestra en el siguiente segmento de código:

In [5]:
```
y = np.array(data['sentiment'])
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Embedding, Conv1D, GlobalMaxPooling1D, Dense

model = Sequential()
model.add(Embedding(5000, 32, input_length=200))
model.add(Conv1D(32, 7, activation='relu'))
model.add(GlobalMaxPooling1D())
model.add(Dense(1, activation='sigmoid'))

model.compile(optimizer='adam', loss='binary_crossentropy', metrics=['accuracy'])
model.fit(X_train, y_train, epochs=5, batch_size=32, validation_data=(X_test, y_test))

loss, accuracy = model.evaluate(X_test, y_test)
print("Loss: ", loss)
print("Accuracy: ", accuracy)
```

```
Epoch 1/5
1250/1250 [==============================] - 20s 15ms/step - loss: 0.4005 - accuracy:
0.8098 - val_loss: 0.3168 - val_accuracy: 0.8642
Epoch 2/5
1250/1250 [==============================] - 19s 15ms/step - loss: 0.2532 - accuracy:
0.8967 - val_loss: 0.3022 - val_accuracy: 0.8726
Epoch 3/5
1250/1250 [==============================] - 21s 16ms/step - loss: 0.1903 - accuracy:
0.9282 - val_loss: 0.3218 - val_accuracy: 0.8682
Epoch 4/5
1250/1250 [==============================] - 20s 16ms/step - loss: 0.1358 - accuracy:
0.9534 - val_loss: 0.3396 - val_accuracy: 0.8739
Epoch 5/5
1250/1250 [==============================] - 20s 16ms/step - loss: 0.0851 - accuracy:
0.9750 - val_loss: 0.3804 - val_accuracy: 0.8709
313/313 [==============================] - 1s 5ms/step - loss: 0.3804 - accuracy: 0.8
709
Loss:  0.380359947681427
Accuracy:  0.8708999752998352
```

e) Una explicación del análisis comparativo de modelos de Deep Learning (Redes Neuronales, Redes CNN (Convoluciones 1D)) Vs Machine Learning (KNN, Bayes, Arboles de Decisión, SVM, Random Forest, Potenciación, etc.), compare al menos dos clasificadores de cada uno con una precisión del 95% o mayor, el análisis deberá comparar: la precisión del modelo, el error del modelo, precisión negativa (especificidad), precisión positiva (sensibilidad), falsos positivos, falsos negativos, asertividad positiva, asertividad negativa

A continuación se realiza la comparación del modelo implementando en código un modelo Naive Bayes como se aprecia a continuación con las implementaciones de la librería Naive Bayes:

In [6]:
```
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.metrics import precision_score
from sklearn.metrics import confusion_matrix

X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=

classifier = GaussianNB()
classifier.fit(X_train, y_train)

y_pred = classifier.predict(X_test)

matriz = confusion_matrix(y_test, y_pred)
print('Matriz de Confusión')
print(matriz)

precision = precision_score(y_test, y_pred, pos_label="a", average=None)
print('Precisión del modelo')
print(precision)
```

```
Matriz de Confusión
[[3827 1207]
 [3675 1291]]
Precisión del modelo
[0.51013063 0.51681345]
```

```
C:\Users\alexe\Anaconda3\envs\ici-thesis\lib\site-packages\sklearn\metrics\_classific
ation.py:1375: UserWarning: Note that pos_label (set to 'a') is ignored when average
!= 'binary' (got None). You may use labels=[pos_label] to specify a single positive c
lass.
  UserWarning,
```

Podemos observar que la precisión positiva y negativa del modelo utilizado el cual es Naive Bayes, es inferior a la precisión el modelo de la red neuronal convolucional presentado en el presente documento con precisiones cercanas al 50% lo cual no demuestra una gran fiabilidad para usar este modelo en contraste con la red neuronal convolucoinal que se acerca a un sólido 90%.

Conclusiones

Es interesante e importante poder implementar las bases de estos temas para entenderlos en un futuro, pues, posteriormente no basta con sólo importar librerías que realicen el trabajo pesado, ya que, implementar manualmente estos algoritmos nos enseña a qué hay detrás del algoritmo, cómo funciona y poder comprender realmente qué está ocurriendo como la base de una red neuronal convolucional y la forma en la que ésta aprende. Es muy útil la implementación de estos algoritmos en estas tareas para las futuras tareas de la materia y aplicaciones de Machine Learning en la vida personal.

Referencias

- Anónimo (s.f.) “Red neuronal artificial”. Obtenido de Wikipedia: https://es.wikipedia.org/wiki/Red_neuronal_artificial.
- Data Scientest (2021) “Perceptrón. ¿Qué es y para qué sirve?”. Obtenido de Data Scientest: https://datascientest.com/es/perceptron-que-es-y-para-que-sirve.
- Luna, F. (2023) “El Modelo de McCulloch – Pitts”. Apuntes de ICI 10°.

In [ ]: