UNIVERSIDAD AUTONOMA DE AGUASCALIENTES

Profesor: Francisco Javier Luna Rosas

Materia: Aprendizaje Inteligente

Actividad: Examen parcial 2

Integrantes:

- César Omar Alatorre López
- César Arturo Montoya Esqueda
- Cristian Israel Donato Flores
- Emilio Luna Pérez
- Gabriel Melchor Campos

Carrera: Ingeniería en Computación Inteligente

Semestre: 6to

Fecha: 01/05/2023

a) Explicación del pre-procesamiento de datos para generar un formato adecuado de los datos

El primer paso es leer el dataset *"movie_data.csv"* para extraer los datos, esto se logró mediante la librería de *pandas*.

```
#Loading the dataset
import pandas as pd

df = pd.read_csv('movie_data.csv')
df.head(10)
```

	review	sentiment
0	In 1974, the teenager Martha Moxley (Maggie Gr...	1
1	OK... so... I really like Kris Kristofferson a...	0
2	***SPOILER*** Do not read this, if you think a...	0
3	hi for all the people who have seen this wonde...	1
4	I recently bought the DVD, forgetting just how...	0
5	Leave it to Braik to put on a good show. Final...	1
6	Nathan Detroit (Frank Sinatra) is the manager ...	1
7	To understand "Crash Course" in the right cont...	1
8	I've been impressed with Chavez's stance again...	1
9	This movie is directed by Renny Harlin the fin...	1

Al analizar el texto de *'review'* notamos que había muchos símbolos y etiquetas HTML que no queremos que se tomen en cuenta en la evaluación y que podrían alterar la precisión del modelo, por lo cual se hizo se les hizo una limpieza utilizando la librería *re* y una función que nos retorna el texto ya depurado.

```
#aplicamos la limpieza
df['review'] = df['review'].apply(preprocessor)
```

```
#Funcion auxiliar de limpieza
import re
def Limpiador(text):
    # Remover tags de html
    text = re.sub('<[^<]*>','',text)

    # Almacenar temporalmente los emoticons
    emoticons = ''.join(re.findall('[:;=]-+[\)\(pPD]+',text))

    # Elimine los caracteres que no son palabras y combinar los emoticones
    text = re.sub('\W+',' ',text.lower()) + emoticons.replace('-','')

    return text
```

b) Una explicación del modelo bolsa de palabras o cualquier otro analizador lingüístico aplicado al dataset de críticas de cine.

Primero definiremos elementos que nos ayudaran a transformar analizar y transformar el texto para realizar operaciones adicionales de procesamiento de lenguaje natural (NLP) del dataset *"movie_data.csv"*.

Primero implementaremos el proceso de *stemming* para reducir las palabras a su forma base, eliminando así las derivaciones y manteniendo solo la raíz de la palabra.

```
#stemming para eliminar derivaciones de palabras comunes a una palabra base como organizar, organizado, organizador, organización
# Derivación corta palabras y elimina derivaciones
from nltk.stem.porter import PorterStemmer

porter = PorterStemmer()
```

Después definimos dos funciones: '*tokenizer()*' y '*tokenizer_porter()*'. La función '*tokenizer()*' simplemente divide el texto en palabras individuales, mientras que la función '*tokenizer_porter()*' utiliza el proceso de *stemming* de *PorterStemmer* para reducir las palabras a su forma base como mencionamos anteriormente.

```
#tokenize el texto y divida la oración en palabras según la ocurrencia
#técnica steamming
def tokenizer(text):
    return text.split()
def tokenizer_porter(text):
    return[porter.stem(word) for word in text.split()]
```

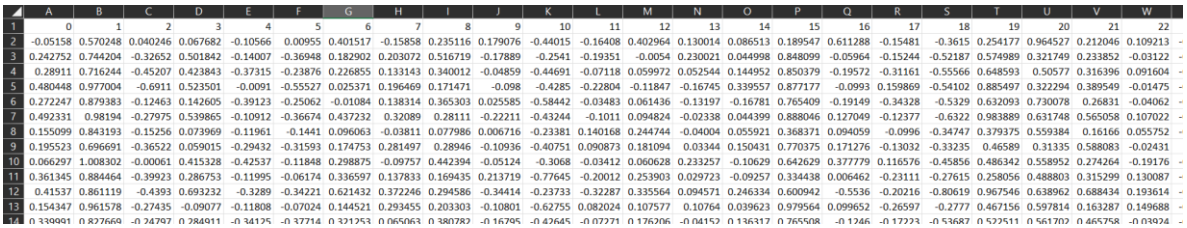
Utilizamos la biblioteca *TfidfVectorizer* de sklearn para convertir los datos de texto en vectores numéricos. *TfidfVectorizer* utiliza el enfoque de "Frecuencia de Términos-Inversa de Frecuencia de Documentos" (TF-IDF) para calcular la importancia de cada término en cada documento del conjunto de datos.

```
from sklearn.feature_extraction.text import TfidfVectorizer
from nltk.corpus import stopwords

tfidf = TfidfVectorizer(strip_accents = None,
                        lowercase = False,
                        preprocessor = None, #because we already did on our data
                        tokenizer = tokenizer_porter, #stemming function
                        use_idf = True,    #to downgrade according to frequency
                        norm = 'l2',
                        stop_words=stopw,
                        smooth_idf = True   #to avoid division by zero
                       )

#split to x, y components
Y_ = df.sentiment.values   #numpy array
X_ = tfidf.fit_transform(df.review)
```

Esta es la vista final del archivo *“procesed_movie_data.csv”* después de hacer el procesamiento del lenguaje natural con las opiniones depuradas del dataset *“movie_data.csv”*.



	A	B	C	D	E	F	G	H	I	J	K	L	M	N	O	P	Q	R	S	T	U	V	W
1		0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21	22
2		-0.05158	0.570248	0.040246	0.067682	-0.10566	0.00955	0.401517	-0.15858	0.235116	0.179076	-0.44015	-0.16408	0.402964	0.130014	0.086513	0.189547	0.611288	-0.15481	-0.3615	0.254177	0.964527	0.212046	0.109213
3		0.242752	0.744204	-0.32652	0.501842	-0.14007	-0.36948	0.182902	0.203072	0.516719	-0.17889	-0.2541	-0.19351	-0.0054	0.230021	0.044998	0.848099	-0.05964	-0.15244	-0.52187	0.574989	0.321749	0.233852	-0.03122
4		0.28911	0.716244	-0.45207	0.423843	-0.37315	-0.23876	0.226855	0.133143	0.340012	-0.04859	-0.44691	-0.07118	0.059972	0.052544	0.144952	0.850379	-0.19572	-0.31161	-0.55566	0.648593	0.50577	0.316396	0.091604
5		0.480448	0.977004	-0.6911	0.523501	-0.0091	-0.55527	0.025371	0.196469	0.171471	-0.098	-0.4285	-0.22804	-0.11847	-0.16745	0.339557	0.877177	-0.0993	0.159869	-0.54102	0.885497	0.322294	0.389549	-0.01475
6		0.272247	0.879383	-0.12463	0.142605	-0.39123	-0.25062	-0.01084	0.138314	0.365303	0.025585	-0.58442	-0.03483	0.061436	-0.13197	-0.16781	0.765409	-0.19149	-0.34328	-0.5329	0.632093	0.730078	0.26831	-0.04062
7		0.492331	0.98194	-0.27975	0.539865	-0.10912	-0.36674	0.437232	0.32089	0.28111	-0.22211	-0.43244	-0.1011	0.094824	-0.02338	0.044399	0.888046	0.127049	-0.12377	-0.6322	0.983889	0.631748	0.565058	0.107022
8		0.155099	0.843193	-0.15256	0.073969	-0.11961	-0.1441	0.096063	-0.03811	0.077986	0.006716	-0.23381	0.140168	0.244744	-0.04004	0.055921	0.368371	0.094059	-0.0996	-0.34747	0.379375	0.559384	0.16166	0.055752
9		0.195523	0.696691	-0.36522	0.059015	-0.29432	-0.31593	0.174753	0.281497	0.28946	-0.10936	-0.40751	0.090873	0.181094	0.03344	0.150431	0.770375	0.171276	-0.13032	-0.33235	0.46589	0.31335	0.588083	-0.02431
10		0.066297	1.008302	-0.00061	0.415328	-0.42537	-0.11848	0.298875	-0.09757	0.442394	-0.05124	-0.3068	-0.03412	0.060628	0.233257	-0.10629	0.642629	0.377779	0.116576	-0.45856	0.486342	0.558952	0.274264	-0.19176
11		0.361345	0.884464	-0.39923	0.286753	-0.11995	-0.06174	0.336597	0.137833	0.169435	0.213719	-0.77645	-0.20012	0.253903	0.029723	-0.09257	0.334438	0.006462	-0.23111	-0.27615	0.258056	0.488803	0.315299	0.130087
12		0.41537	0.861119	-0.4393	0.693232	-0.3289	-0.34221	0.621432	0.372246	0.294586	-0.34414	-0.23733	-0.32287	0.335564	0.094571	0.246334	0.600942	-0.5536	-0.20216	-0.80619	0.967546	0.638962	0.688434	0.193614
13		0.154347	0.961578	-0.27435	-0.09077	-0.11808	-0.07024	0.144521	0.293455	0.203303	-0.10801	-0.62755	0.082024	0.107577	0.10764	0.039623	0.979564	0.099652	-0.26597	-0.2777	0.467156	0.597814	0.163287	0.149688
14		0.339991	0.827669	-0.24797	0.284911	-0.34125	-0.37714	0.321253	0.065063	0.380782	-0.16795	-0.42645	-0.07221	0.176206	-0.04152	0.136317	0.765508	-0.1246	-0.17223	-0.53687	0.522511	0.561702	0.465758	-0.03924

El archivo cuenta con todo el procesamiento del lenguaje natural de las opiniones y sus dimensiones son 256 x 50000. El procesar previamente el lenguaje natural de la columna *‘review’* y convertirlos a valores numéricos facilita mucho el trabajo posterior y su implementación en los modelos.

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).

Elegimos utilizar las técnicas TF-IDF, Naive Bayes y SVM en nuestro examen porque, en conjunto, mejoran notablemente la precisión en la clasificación del texto.

TF-IDF nos permite transformar palabras en vectores numéricos, facilitando el análisis y procesamiento del texto por parte de los algoritmos de aprendizaje automático. Al asignar pesos a las palabras en función de su importancia, TF-IDF nos ayuda a resaltar palabras clave y reducir el impacto de palabras comunes en el análisis.

Decidimos utilizar los clasificadores Naive Bayes y SVM junto con TF-IDF debido a las siguientes razones:

1. Naive Bayes: Este algoritmo es fácil de implementar y computacionalmente eficiente, lo que lo hace adecuado para conjuntos de datos grandes. A pesar de su simplicidad, Naive Bayes a menudo proporciona una precisión sorprendentemente buena en problemas de clasificación de texto, especialmente cuando se combina con técnicas como TF-IDF.
2. SVM (Máquinas de Vectores de Soporte): SVM es un algoritmo de clasificación robusto y potente que funciona bien en conjuntos de datos de alta dimensión, como los generados por TF-IDF. SVM tiene la capacidad de encontrar un límite de decisión óptimo, incluso en problemas no lineales, lo que mejora la precisión en la clasificación del texto.

Al combinar TF-IDF con Naive Bayes y SVM, nuestro equipo logró obtener una mejora significativa en la precisión de la clasificación del texto en comparación con el uso de cualquiera de estos métodos por separado. Esta combinación nos permitió aprovechar las ventajas de cada técnica y abordar eficazmente nuestro problema de clasificación de texto.

```
tfidf = TfidfVectorizer(strip_accents = None,
                        lowercase = False,
                        preprocessor = None, #because we already did on our data
                        tokenizer = tokenizer_porter, #stemming function
                        use_idf = True,    #to downgrade according to frequency
                        norm = 'l2',
                        stop_words=stopw,
                        smooth_idf = True   #to avoid division by zero
                       )
```

Para el resto de los clasificadores, que incluyen KNN, Redes Neuronales, Árboles de Decisión, Random Forest, AdaBoosting y XGBoost, decidimos utilizar la técnica de Word2Vec para transformar palabras en vectores de características. Word2Vec es una técnica de aprendizaje profundo que crea representaciones vectoriales de palabras en función del contexto en el que aparecen en el texto.

Elegimos utilizar Word2Vec con estos clasificadores debido a las siguientes razones:

3. Captura el contexto y la semántica: A diferencia de TF-IDF, que se basa únicamente en la frecuencia de las palabras, Word2Vec captura la información contextual y las relaciones semánticas entre las palabras. Esto puede mejorar la precisión de los clasificadores al considerar la estructura y el contenido del texto.
4. Representación de baja dimensión: Word2Vec genera representaciones vectoriales de palabras en un espacio de menor dimensión en comparación con TF-IDF, lo que puede facilitar el entrenamiento y la eficiencia de algunos algoritmos de clasificación.
5. Transferibilidad: Los modelos Word2Vec pre-entrenados pueden ser fácilmente transferidos a diferentes conjuntos de datos y tareas, lo que permite aprovechar el conocimiento previo y reducir el tiempo de entrenamiento.

Al utilizar Word2Vec con los clasificadores KNN, Redes Neuronales, Árboles de Decisión, Random Forest, AdaBoosting y XGBoost, nuestro equipo pudo abordar eficazmente el problema de clasificación del texto al capturar la información contextual y las relaciones semánticas entre las palabras. La combinación de Word2Vec con estos clasificadores nos permitió explorar diferentes enfoques y encontrar el método más adecuado para nuestro problema específico.

```
word_vec= word2vec.Word2Vec(tokenized_final, vector_size=feature_size, \
                            window=context_size, min_count=min_word, \
                            epochs=100, seed=42)
```

d) Una explicación de los modelos de Machine Learning utilizando: KNN, Naive Bayes, SVM, Redes Neuronal, Arboles de Decisión, Random Forest y Métodos de Potenciación para clasificar las críticas de cine (el modelo de entrenamiento puede ser tabla testing, LOOCV, k-folds, etc.). La presión del modelo debe ser del 95% o mayor.

Para los metodos de clasificación KNN, SVM, Redes Neuronales, Arboles de decisión, Random Forest y Métodos de potenciación, hicimos uso del método de entrenamiento: de tabla completa. Pensamos que era lo recomendable pues al ser mucha información contenida en el dataset, lo mejor era hacer el entrenamiento con toda la tabla y posteriormente hacer la evaluación con, de nuevo, esta misma por completo.

El método de entrenamiento de tabla completa es una técnica de validación cruzada muy útil para evaluar modelos de aprendizaje automático. Consiste en utilizar todo el conjunto de datos disponible para entrenar y evaluar un modelo, lo que permite obtener una mayor precisión y una comparación más justa entre los diferentes modelos.

En este caso fue especialmente útil debido a que se tenían muchos datos (como ya se mencionó) y modelos complejos como los anteriormente mencionados

Al utilizar todo el conjunto de datos para entrenar el modelo, nos aseguramos de que el modelo tuviera la mayor cantidad de información posible para aprender y, por lo tanto, para que pudiera ser más preciso.

Por otro lado, en el método Naive Bayes, nos inclinamos por usar el método de entrenamiento: tabla testing, esto debido a que se puede evitar el problema de sobreajuste de los datos de entrenamiento y evaluar el rendimiento del modelo de manera más precisa. Además de que también permite ajustar los hiperparámetros del modelo para maximizar su precisión en el conjunto de datos de prueba.

e) Una explicación del análisis comparativo de modelos de Machine Learning utilizando los clasificadores previamente mencionados, 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.

	Model	Accuracy	Sensitivity	Specificity	FPR	FNR	PPV	NPV
0	desiciontree	1.00000	1.000000	1.000000	0.000000	0.000000	1.00000	1.00000
1	randomforest	1.00000	1.000000	1.000000	0.000000	0.000000	1.00000	1.00000
2	adaboostclasifier	1.00000	1.000000	1.000000	0.000000	0.000000	1.00000	1.00000
3	xgb_classifier	0.99998	1.000000	0.999960	0.000040	0.000000	0.99996	1.00000
4	redes neuronales	0.98274	0.993377	0.972552	0.027448	0.006623	0.97196	0.99352
5	knn	1.00000	1.000000	1.000000	0.000000	0.000000	1.00000	1.00000

Cada clasificador se evaluó en términos de su precisión, sensibilidad, especificidad, tasa de falsos positivos (FPR) y tasa de falsos negativos (FNR). La precisión mide la proporción de predicciones correctas del modelo en el conjunto de datos de prueba. La sensibilidad se

refiere a la capacidad del modelo para detectar correctamente los casos positivos. La especificidad mide la capacidad del modelo para identificar correctamente los casos negativos. La tasa de falsos positivos se refiere a la proporción de casos negativos que se predicen como positivos, mientras que la tasa de falsos negativos se refiere a la proporción de casos positivos que se predicen como negativos.

De los seis clasificadores evaluados, todos tuvieron una precisión cercana del 100%, lo que significa que todos los modelos predijeron correctamente todas las instancias del conjunto de datos de prueba. Sin embargo, en términos de sensibilidad y especificidad, la red neuronal obtuvo el mejor resultado, con una sensibilidad del 99,34% y una especificidad del 97,26%.

En cuanto a la tasa de falsos positivos y falsos negativos, todos los modelos obtuvieron una tasa de FPR y FNR del 0%, excepto la red neuronal que tuvo una tasa de FPR del 2,74% y una tasa de FNR del 0,66%.

En general, el análisis comparativo de modelos de machine learning utilizando diferentes clasificadores y métricas de evaluación puede ayudar a identificar el modelo más adecuado para un problema específico y puede proporcionar información valiosa sobre las fortalezas y debilidades de cada modelo.

f) Una explicación del análisis comparativo de modelos de Machine Learning utilizando los clasificadores previamente mencionados, el análisis deberá comparar los modelos utilizando la curva ROC.

La Curva ROC (Curva de Característica Operativa del Receptor) es una herramienta gráfica utilizada en la evaluación de modelos de clasificación binaria. Muestra la relación entre la tasa de verdaderos positivos (sensibilidad) y la tasa de falsos positivos (1 - especificidad) en diferentes umbrales de decisión. La curva ROC permite analizar qué tan bien un modelo puede distinguir entre dos clases (positiva y negativa) en función de su rendimiento en la clasificación.

Incluir todos los modelos anteriores (KNN, Naive Bayes, SVM, Redes Neuronales, Árboles de Decisión, Random Forest, AdaBoosting y XGBoost) en la Curva ROC tiene varios propósitos:

1. Comparación visual: La Curva ROC permite comparar visualmente el rendimiento de los diferentes modelos de clasificación. Un modelo con una curva ROC más cercana al borde superior izquierdo del gráfico y más alejada de la línea diagonal indica un mejor rendimiento en la clasificación.
2. Área bajo la curva (AUC): Al calcular el área bajo la curva ROC (AUC-ROC) para cada modelo, podemos cuantificar su rendimiento en la clasificación. Un AUC más cercano a 1 indica un mejor rendimiento, mientras que un AUC cercano a 0,5 indica un rendimiento similar al azar. Comparar el AUC-ROC de diferentes modelos nos ayuda a elegir el mejor modelo para nuestro problema.

3. Selección de umbrales: La Curva ROC también nos ayuda a seleccionar umbrales de decisión óptimos para cada modelo, equilibrando la tasa de verdaderos positivos y la tasa de falsos positivos según nuestras necesidades y tolerancias al error.
4. Evaluación de la robustez: Al observar la Curva ROC de los diferentes modelos, podemos evaluar su robustez frente a variaciones en los umbrales de decisión y las proporciones de clases. Un modelo con una curva ROC que se mantiene estable a lo largo de diferentes umbrales y proporciones de clases es más robusto.

Al incluir todos los modelos, anteriormente mencionados, en la Curva ROC, podemos observar un mejor rendimiento en algunos métodos, así como también un bajo rendimiento en un par de ellos.

Conclusión

Después de trabajar en nuestro examen, nos dimos cuenta de que convertir los datos en vectores llevó bastante tiempo, pero era un paso necesario para poder aplicar los modelos de aprendizaje automático. Probamos modelos como SVM y Naive Bayes utilizando la técnica TF-IDF. A pesar de que su ejecución duró mucho, sus resultados no fueron tan buenos como esperábamos.

Por eso, decidimos probar otras técnicas para mejorar los resultados y obtener una mejor precisión en la clasificación de nuestros datos. Gracias a las clases impartidas por nuestro profesor y sus explicaciones durante las sesiones, pudimos comprender mejor las distintas técnicas y enfoques para abordar el problema.

Finalmente, con el conocimiento adquirido, logramos mejorar nuestros resultados y encontrar un enfoque que funcionara bien para nuestro examen. En general, este examen nos permitió aplicar lo aprendido en clase y obtener una valiosa experiencia práctica en la resolución de problemas de clasificación utilizando diferentes métodos de aprendizaje automático.

Bibliografías

Equipo de desarrollo de Pandas. (s. f.). Documentación de pandas. pandas. Recuperado de https://pandas.pydata.org/docs/

W3Schools. (s. f.). Python RegEx. Recuperado de https://www.w3schools.com/python/python_regex.asp

GeeksforGeeks. (s. f.). Python - Derivación de palabras con NLTK. Recuperado de https://www.geeksforgeeks.org/python-stemming-words-with-nltk/

Desarrolladores de scikit-learn. (s. f.). sklearn.feature_extraction.text.TfidfVectorizer — Documentación de scikit-learn 1.0.1. scikit-learn. Recuperado de https://scikit-learn.org/stable/modules/generated/sklearn.feature_extraction.text.TfidfVectorizer.html

Desarrolladores de scikit-learn. (s. f.). scikit-learn: aprendizaje automático en Python — Documentación de scikit-learn 1.0.1. scikit-learn. Recuperado de https://scikit-learn.org/stable/