UNIVERSIDAD DE LOS ANDES
Departamento de Ingeniería Industrial
Minería de Datos MIIA4200- Métodos avanzados de análisis de datos

Heidy Tatiana Garzón Parra	**Código:** 202020026
Carolina Salazar Lara	**Código:** 200923246
Nicolás Soto Novoa	**Código**: 200814680
Giovani Daniel Lara Acosta	**Código:** 201920339
Fabian Camilo Lara Acosta	**Código:** 201920066

Multi-label Text Classification: Predicción de Género(s) de películas a partir de la sinopsis

Objetivo

Clasificar el o los géneros de las películas a partir de la sinopsis:

- Procesar los textos de la sinopsis de las películas para extraer información relevante sobre el género
- Predecir la probabilidad que una película pertenezca a un género.

Contexto

En la actualidad son muchas las fuentes que generan grandes cantidades de datos en poco tiempo: redes sociales, foros, e-mails, librerías y libros digitales, páginas web, canales de auto respuesta (como chatbots), entre otros. Extraer información relevante de forma manual resulta un proceso costoso, en tiempo y dinero, y está sujeto a error humano – perder detalles relevantes o sesgos por subjetividad.

Las técnicas de clasificación de texto permiten procesar y organizar automáticamente grandes cantidades de texto para que puedan ser navegados de forma eficiente (con identificadores o categorías). De esa forma, incrementando la eficiencia de procesos y apoyando la oportuna toma de decisiones a partir de esa información escrita. Son muchos los escenarios donde se puede ver el valor de esta metodología: entidades regulatorias como INVIMA o FDA quienes reciben, clasifican y procesan textos que provienen del sector farmacéutico, alimenticio, cosmético; empresas privadas como Avianca que deben asegurar el lineamiento de sus manuales de aviación con la regulación aérea; o incluso en las redes sociales, donde es crítico determinar de forma oportuna la favorabilidad de algún candidato electoral tras la publicación de un trino[1].

En este trabajo, exploraremos técnicas de procesamiento del lenguaje para lograr predecir los géneros de una película a partir de su sinopsis. Para esto, trabajaremos con clasificación multi-etiqueta en Natural Language Processing (NLP), un escenario interesante donde cada observación puede tener 2 o más outputs de salida (ver Figura 1, tabla 3).

Table 1

X	y
X_1	t_1
X_2	t_2
X_3	t_1
X_4	t_2
X_5	t_1

Binary Classification

Table 2

X	y
X_1	t_2
X_2	t_3
X_3	t_4
X_4	t_1
X_5	t_3

Multi-class Classification

Table 3

X	y
X_1	$[t_2, t_5]$
X_2	$[t_1, t_2, t_3, t_4]$
X_3	$[t_3]$
X_3	$[t_2, t_4]$
X_3	$[t_1, t_3, t_4]$

Multi-label Classification

Figura 1. *Diferencias entre los tipos de clasificación (https://www.analyticsvidhya.com/blog/2019/04/predicting-movie-genres-nlp-multi-label-classification/*

[1] Text Classification - an overview | ScienceDirect Topics, Link: https://www.sciencedirect.com/topics/computer-science/text-classification

1. Datos

Base de datos de 7895 películas, cada una con su título, género, **rating y sinopsis (en Inglés).**

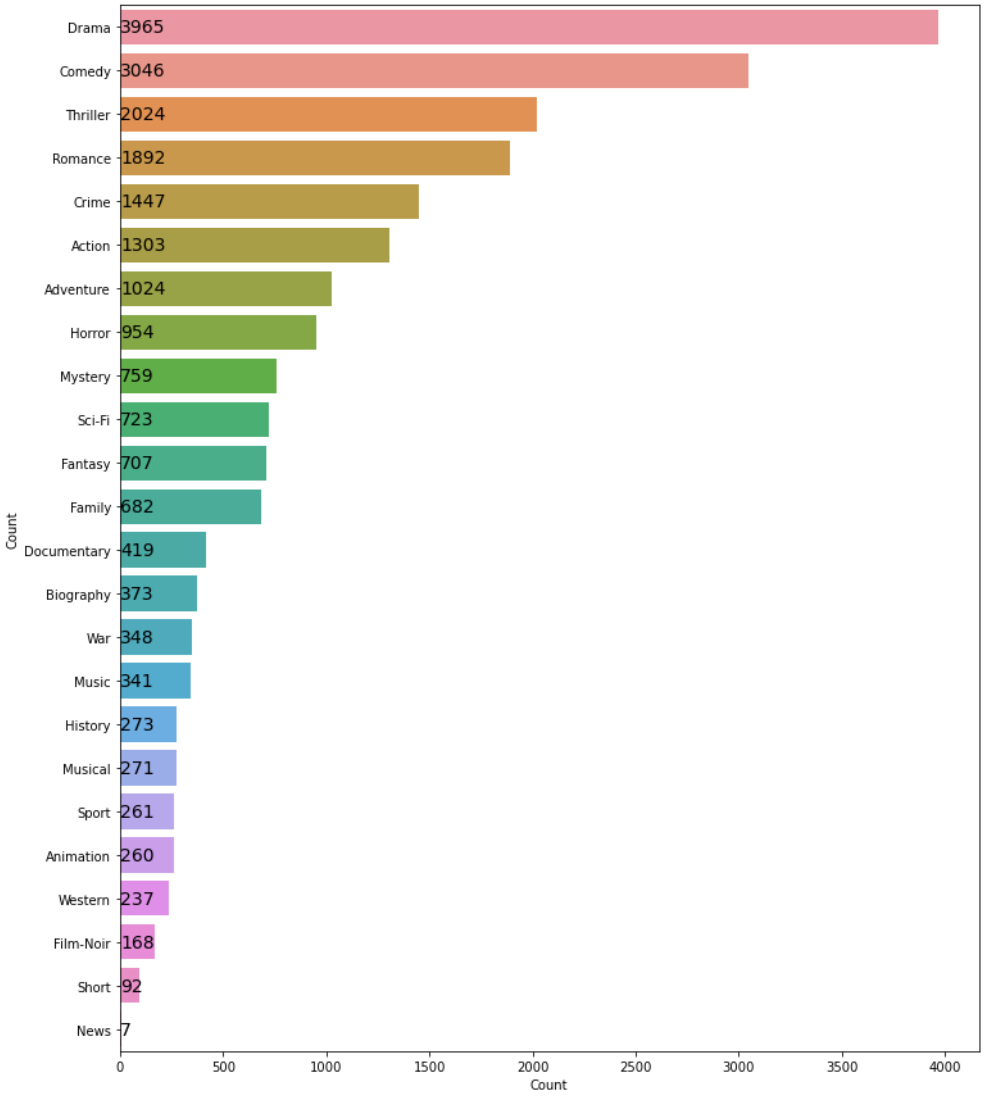
Variable Predictora:
Sinopsis de la película

'When motocross and heavy metal obsessed thirteen - year - old Jacob ' s increasing delinquent behavior forces cps to place his little brother, Wes, with his aunt, Jacob and his emotionally absent father, Hollis, must finally take responsibility for their actions and for each other in order to bring Wes home clerk'.

Variable de salida: Probabilidad de que una película pertenezca a cada género.

p_Action	p_Adventure	p_Animation	p_Biography	p_Comedy	p_Crime	p_Documentary	p_Drama	p_Family	p_Fantasy	...	p_Musical	p_Mystery
0.143030	0.101960	0.024454	0.029938	0.354552	0.138830	0.030787	0.490140	0.073159	0.101339	...	0.025069	0.063208

Los géneros de las películas encontrados en la base de datos, con su respectiva frecuencia se encuentra en la siguiente gráfica:



2. Preparación y limpieza de datos

Para analizar el texto de las variables predictoras se realizó el siguiente preprocesamiento, en este caso se utilizó librería NLTK.

- **Limpieza del texto:** La siguiente función busca eliminar caracteres especies en el texto y dejar en letras minúsculas todo el string.

```
import re

def clean_text(text):
    text = re.sub("\'", "", text)
    text = re.sub("[^a-zA-Z]"," ",text)
    text = text.lower()
    return text
dataTraining['plot'] = dataTraining['plot'].apply(lambda x: clean_text(x))
dataTesting['plot'] = dataTesting['plot'].apply(lambda x: clean_text(x))
```

- **Eliminación de números:** La siguiente función elimina cualquier número encontrado en el texto.

```
from string import digits

def eliminate_numbers(texto):
    remove_digits = str.maketrans('', '', digits)
    texto = texto.translate(remove_digits)

    return texto

dataTraining['plot'] = dataTraining['plot'].apply(lambda x: eliminate_numbers(x))
dataTesting['plot'] = dataTesting['plot'].apply(lambda x: eliminate_numbers(x))
```

- **Tokenización**: El siguiente paso fue convertir cada texto en unidades más pequeñas en este caso palabras.

```
def tokenization(text):
    text = re.split('\W+', text)
    return text

dataTraining['plot'] = dataTraining['plot'].apply(lambda x: tokenization(x.lower()))
dataTesting['plot'] = dataTesting['plot'].apply(lambda x: tokenization(x))
```

- **Stemming:** La siguiente función reduce las palabras a una raíz, eliminando caracteres innecesarios y dejando un sufijo principal

```
def stemming(text):
    text = [stemmer.stem(word) for word in text]
    return text

dataTraining['plot'] = dataTraining['plot'].apply(lambda x: stemming(x))
dataTesting['plot'] = dataTesting['plot'].apply(lambda x: stemming(x))
```

3. Evaluación de modelos

Creación de la Y:

```
dataTraining['genres'] = dataTraining['genres'].map(lambda x: eval(x))

le = MultiLabelBinarizer()
y_genres = le.fit_transform(dataTraining['genres'])
```

Embedings:
Para la realización del embeding, se realizaron diferentes técnicas para convertir el texto. Con el tratamiento realizado, se obtuvo una matriz en la cual quedan las variables predictoras de los modelos que se entrenarán. De esta manera, se identificó cuál técnica era la mejor, en términos del poder de predicción de los modelos ajustados.

Count vectorizer:

```
vect = CountVectorizer(max_features=1000)
X_dtm = vect.fit_transform(dataTraining['plot'])
X_dtm.shape
```

Term-Frequency Times Inverse Document-Frequency (TFIDF):

```
vect = TfidfVectorizer(max_features=10000)
X_dtm = vect.fit_transform(dataTraining['plot'])
X_dtm.shape

X_train, X_test, y_train_genres, y_test_genres = train_test_split(X_dtm, y_genres, test_size=0.33, random_state=42)
```

Universal-Sentence-Encoder:

```
import tensorflow as tf
import tensorflow.compat.v1 as tf
#To make tf 2.0 compatible with tf1.0 code, we disable the tf2.0 functionalities
tf.disable_eager_execution()
import tensorflow_hub as hub

module_url = "https://tfhub.dev/google/universal-sentence-encoder-large/3" #@param ["https://tfhub.dev/google/universal

# Import the Universal Sentence Encoder's TF Hub module
embed = hub.Module(module_url)

with tf.Session() as session:
    session.run([tf.global_variables_initializer(), tf.tables_initializer()])
    sentences_embeddings = session.run(embed(dataTraining['plot']))

sentences_embeddings
```

Hashing Vectorizer

```
vect = HashingVectorizer(n_features=20, stop_words=eng_stopwords)
X_dtm = vect.fit_transform(dataTraining['plot'])
```

Luego de realizar las anteriores metodologías, el embeding que tuvo mejor AUC fue el Term Frequency-Inverse Document Frequency (TF-IDF), lamentablemente el Universal-Sentence-Encoder no logró correr con el total de datos, lo cual probablemente se debió a la capacidad computacional disponible.

4. Modelamiento

Para la evaluación de modelos, dad la competencia y teniendo en cuenta que el conjunto de datos de prueba no contenía la variable respuesta, se decidió dividir de manera aleatoria el conjunto de entrenamiento, en 67% y 33%, esto con el fin de tener una idea preliminar y una estimación del AUC. Sin embargo, en cada iteración que se subía a Kaggle se entrenaba el modelo con el conjunto de datos completo y de esta forma no se perdía información.

Para construir el modelo se probaron varias de las técnicas abordadas en el numeral 2, esto con el fin de saber si alguno de estos procesos agregaba valor a la predicción. Así mismo, cabe resaltar que todos los modelos fueron calibrados basados en los parámetros que mejor AUC lograban.

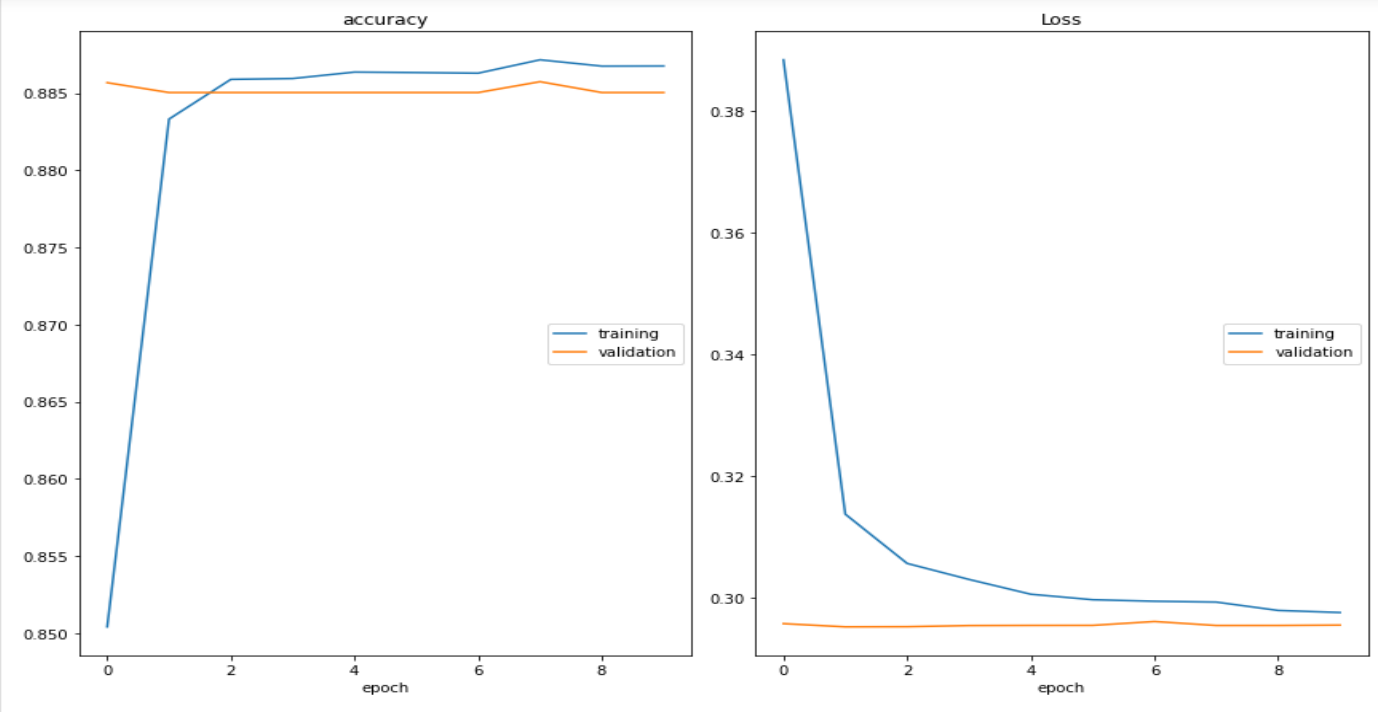
A continuación, por simplicidad del documento se presentarán algunos de los resultados de todas las iteraciones que se realizaron para encontrar la mejor predicción:

LSTM Model:

Para realizar la calibración del modelo LSTM se tuvo en cuenta principalmente el número de salidas que debía tener la última capa, por otro lado, se probaron diferentes números de neuronas en la hidden layer, aunque no demasiadas, dado que el tiempo que toma la corrida de entrenamiento es muy larga. De lo anterior, se obtuvo lo siguiente:

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
embedding_1 (Embedding)      (None, 1500, 128)         3584
_________________________________________________________________
lstm_1 (LSTM)                (None, 21)                12600
_________________________________________________________________
dropout_1 (Dropout)          (None, 21)                0
_________________________________________________________________
dense_1 (Dense)              (None, 24)                528
=================================================================
Total params: 16,712
Trainable params: 16,712
Non-trainable params: 0
_________________________________________________________________
```

Al efectuar el análisis de la precisión y de la pérdida, en los conjuntos de datos de entrenamiento y validación, para 10 epochs, se obtuvieron los siguientes resultados:



Train multi-class multi-label model

Se corrieron varias iteraciones de los modelos de Random Forest, XGBoost, Ridge y regresión logística, a continuación, algunos de los ejemplos que presentaron mejores resultados:

```
clf = OneVsRestClassifier(RandomForestClassifier(n_jobs=-1, n_estimators=100, max_depth=10, random_state=42))
```

```
clf2 = OneVsRestClassifier(XGBClassifier(learning_rate=0.1, colsample_bytree=0.5, gamma=4, random_state=1))
```

```
clf3 = OneVsRestClassifier(Ridge(alpha=0.91, random_state=1))
```

```
clf4 = OneVsRestClassifier(LogisticRegression(penalty='l2', solver='lbfgs', random_state=1))
```

De los modelos mencionados, el que mejor desempeño arrojó fue el de regresión logística como se resume en la siguiente tabla, la cual muestra la relación de accuracy que obtuvimos con los distintos modelos:

Metodología	Estimación Accuracy
Random Forest	0.814770
XGBoost	0.8408335
Regresión Logística	0.88217652
LSTML	0.5409845
Stacking Regresión Logistica+XGboost	0.88560986

5. Selección del modelo final, análisis y conclusiones:

Dado el modelamiento explicado en el numeral anterior, se seleccionó el modelo de Stacking LR+XGboost el cual presentaba un mejor desempeño frente a las demás alternativas.

Allí pudimos observar que el stacking presenta una mejora respecto al AUC versus los modelos independientes, por otro lado, el modelo de red neuronal no da buenos resultados, pero muy probablemente con más trabajo de calibración podría mejorar, este tipo de modelos requiere mucha capacidad computacional y tiempo para lograr buena calibración.

También corroboramos que la limpieza de los datos es esencial para realizar un modelo de predicción, dado que nuestras primeras iteraciones en donde no habíamos aplicado algunas de las técnicas descritas, hacían que el desempeño de los modelos no fuera tan bueno.

Destacamos el stemming, el TF-IDF y la eliminación de números en el proceso, dado que los modelos mejoraron considerablemente al efectuar estas técnicas.

En el modelamiento fue clave la calibración de los distintos hiperparámetros que hacen parte de los modelos. Allí observamos que se podían obtener mejores resultados con determinadas combinaciones.