



INFORME EJECUTIVO COMPETENCIA MIAD2024-12

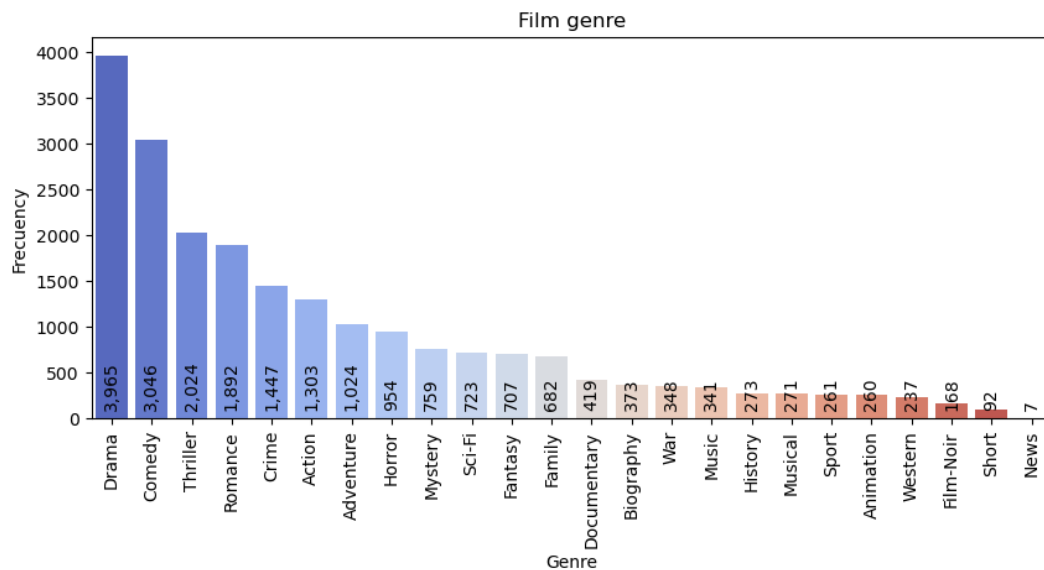
Clasificación de género de películas

Desarrollado por:

- KAREN YORLADY ROJAS GIRALDO
- JHOCEL DUVAN SUESCUN TORRES
- JUAN SEBASTIAN HERNANDEZ RAMIREZ
- JUAN PABLO MOGOLLÓN AVAUNZA

1 Exploración preliminar de los datos

- 1.1 Se tienen dos archivos, uno para entrenar y comprobar el desempeño del algoritmo (dataTraining.zip) y otro para obtener las predicciones para subir a la competencia en Kaggle (dataTesting.zip). Donde el conjunto de dataTraining tiene 7895 registros y 5 columnas o variables, mientras que el conjunto dataTesting tiene 3383 registros y un total de 3 columnas o variables.
- 1.2 Los conjuntos de datos presentan información de películas, específicamente 3 columnas categóricas (title, plot, genres) y 2 numéricas (year, rating), donde la variable genres corresponde a la variable de análisis a predecir, mientras que la variable plot es la variable con la que se hará el entrenamiento del modelo y a partir de ella se generarán las correspondientes predicciones.
- 1.3 Debido a que la única variable que se usará como predictora es 'plot', no se hará un análisis exploratorio más profundo sobre las demás variables.
- 1.4 No hay valores faltantes en ninguno de los dos conjuntos de datos.
- 1.5 Se observa que la variable 'genres' es una combinación de múltiples valores únicos del género de la película, lo que quiere decir que las películas pueden tener uno o más géneros a los que pertenecen. Donde existen 1336 combinaciones únicas de géneros que se conforman de 24 géneros base únicos.
- 1.6 Por último, es posible observar la distribución de los géneros en dataTraining, donde es posible observar que la moda del género es 'Drama' y el género menos representativo es 'News'.



2 Preprocesamiento de datos

- 2.1 Se ajusta la naturaleza de la variable a predecir 'genre'. Inicialmente, esta tiene tipo string, por lo que es necesario aplicar la función eval sobre esta variable para que pase de strings a listas reales.
- 2.2 Estas listas de géneros se binarizan, para esto se hace uso de una instancia del objeto MultiLabelBinarizer que permite pasar de una columna de listas categóricas a una matriz binaria que representa la pertenencia o no pertenencia de una película a un género específico.
- 2.3 Se realizó la división de los datos en set de entrenamiento y prueba usando train_test_split utilizando el conjunto de datos de entrenamiento "dataTraining" esto con el fin de poder realizar evaluación de los resultados antes de obtener las predicciones en "dataTest" para subir a la competencia.
- 2.4 Este es un fragmento del código con el que se realizó la implementación descrita en los puntos anteriores:

```

1. # Cargar el dataframe
2. allData = pd.read_csv('https://github.com/albahnsen/MIAD_ML_and_NLP/raw/main/datasets/dataTraining.zip', encoding='UTF-8', index_col=0)
3.
4. # Preparar los datos
5. X = allData['plot'].tolist()
6. y = allData['genres'].map(lambda x: eval(x)) # Asegurarse de que los géneros están en formato de lista
7.
8. # Convertir las etiquetas a formato binarizado
9. mlb = MultiLabelBinarizer()
10. y_binarized = mlb.fit_transform(y)
11.
12. # Dividir los datos en conjuntos de entrenamiento y prueba
13. X_train, X_test, y_train, y_test = train_test_split(X, y_binarized, test_size=0.33, random_state=42)

```

- 2.5 Para la transformación del texto que describe la película incluido en la columna 'plot' se probaron diversos métodos y su impacto en la capacidad predictiva del modelo a través de la métrica AUC('macro').
- 2.6 Utilizamos las siguientes estrategias para preprocesar el texto: CountVectorizer, CountVectorizer con stop words, lematización, n_gramas, TfidfVectorizer(), y combinaciones de ellas. A continuación, un ejemplo de la implementación de esta transformación.

```

11. # Descargar las stopwords
12. nltk.download('stopwords')
13.
14. # Lista de stop words
15. stop_words = set(stopwords.words('english'))

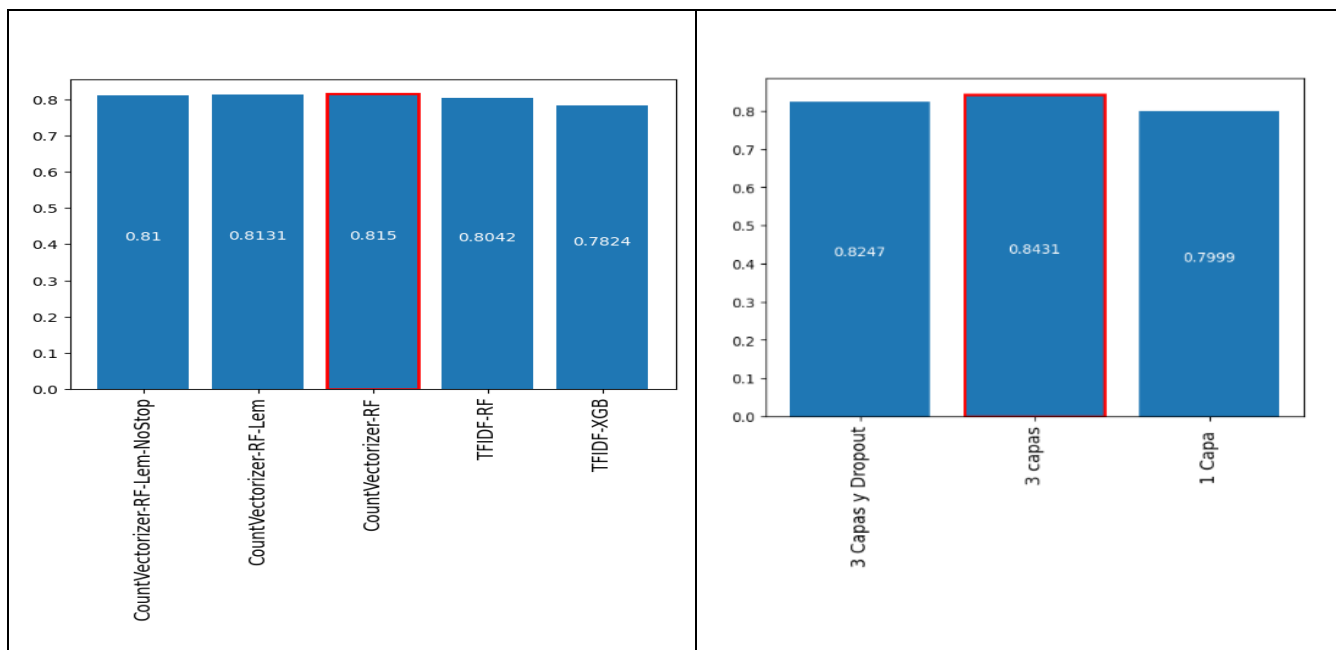
```

```

16.
17. # Función para eliminar stop words
18. def remove_stop_words(text):
19.     # Dividir el texto en palabras
20.     words = re.findall(r'\b\w+\b', text.lower())
21.     # Filtrar las palabras que no están en las stop words
22.     filtered_words = [word for word in words if word not in stop_words]
23.     # Unir las palabras filtradas de nuevo en una oración
24.     return ' '.join(filtered_words)
25.
26. # Descargar recursos de nltk
27. nltk.download('punkt')
28.
29. # Cargar el modelo en inglés de spaCy
30. nlp = spacy.load("en_core_web_sm")
31.
32. # Función para lematizar una frase
33. def lemmatize_sentence(sentence, nlp):
34.     # Procesar la frase con spaCy
35.     doc = nlp(sentence)
36.     # Extraer las lemas
37.     lemmas = [token.lemma_ for token in doc]
38.     # Unir las lemas en una frase
39.     lemmatized_sentence = " ".join(lemmas)
40.     return lemmatized_sentence
41.
42. # Cargar y preprocesar los datos
43. data = pd.read_csv('https://github.com/albahnsen/MIAD_ML_and_NLP/raw/main/datasets/dataTraining.zip', encoding='UTF-8', index_col=0)
44. data.drop_duplicates(inplace=True)
45. data.dropna(subset=['plot', 'genres'], inplace=True)
46.
47. # Tokenizar las tramas de las películas
48. data['plot'] = data['plot'].apply(lambda x: ' '.join(word_tokenize(x.lower())))
49. data['plot'] = data['plot'].apply(remove_stop_words)
50. data['plot'] = data['plot'].apply(lambda x: lemmatize_sentence(x, nlp))
51.
52. # Definición de variable de interés (y)
53. data['genres'] = data['genres'].map(lambda x: eval(x))
54.
55. # Convertir géneros a etiquetas binarizadas
56. mlb = MultiLabelBinarizer()
57. genre_labels = mlb.fit_transform(data['genres'])
58.
59. # Separación de variables predictoras (X) y variable de interés (y) en set de entrenamiento y test usando la función train_test_split
60. X_train, X_test, y_train_genres, y_test_genres = train_test_split(data['plot'], genre_labels, test_size=0.33, random_state=42)
61.
62. # Vectorizar los textos con TfidfVectorizer
63. tfidf_vectorizer = TfidfVectorizer()
64. X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
65. X_test_tfidf = tfidf_vectorizer.transform(X_test)

```

- 2.7 En este ejemplo realizamos la tokenización usando la biblioteca nltk, posteriormente eliminamos “stop words” y lematizamos utilizando spaCy. Esto antes de dividir los datos en un conjunto de entrenamiento y pruebas. Luego vectorizamos los textos utilizando TfidfVectorizer(), para lo cual aplicamos “fit_transform” a los datos de entrenamiento y “transform” a los datos de pruebas.
- 2.8 Para obtener datos preliminares de cómo las diferentes formas de preprocesamiento de los datos podrían afectar el rendimiento predictivo de un modelo, utilizamos random forest, xgboost y redes neuronales, calibrando sus parámetros y así como cambiando el procesamiento de los datos.
- 2.9 En las gráficas siguientes un resumen breve de los resultados:



2.10 Para calibrar RandomForest y XGBoost usamos el método RandomizedSearchCV debido a su eficiencia computacional, las redes neuronales las calibramos utilizando ParameterGrid. A continuación, un ejemplo del código utilizado para realizar predicciones con XGBoost y calibrarlo utilizando RandomizedSearchCV. En este caso obtuvimos los siguientes resultados: En este caso obtuvimos los siguientes resultados:

- Best hyperparameters: {'subsample': 0.7, 'n_estimators': 500, 'min_child_weight': 3, 'max_depth': 7, 'learning_rate': 0.1, 'colsample_bytree': 0.5}
- Best macro ROC AUC: 0.7499537185773792
- Test macro ROC AUC: 0.8148595931613704

```

10. # Descargar recursos de nltk
11. nltk.download('punkt')
12.
13. # Cargar y preprocesar los datos
14. data = pd.read_csv('https://github.com/albahnsen/MIAD_ML_and_NLP/raw/main/datasets/dataTraining.zip', encoding='UTF-8',
index_col=0)
15. data.drop_duplicates(inplace=True)
16. data.dropna(subset=['plot', 'genres'], inplace=True)
17.
18. # Tokenizar las tramas de las películas
19. data['plot'] = data['plot'].apply(lambda x: ' '.join(word_tokenize(x.lower())))
20.
21. # Convertir géneros a etiquetas binarizadas
22. data['genres'] = data['genres'].map(lambda x: eval(x))
23. mlb = MultiLabelBinarizer()
24. genre_labels = mlb.fit_transform(data['genres'])
25.
26. # Dividir los datos en conjunto de entrenamiento y prueba
27. X_train, X_test, y_train, y_test = train_test_split(data['plot'], genre_labels, test_size=0.2, random_state=42)
28.
29. # Vectorizar los textos con TfidfVectorizer
30. tfidf_vectorizer = TfidfVectorizer()
31. X_train_tfidf = tfidf_vectorizer.fit_transform(X_train)
32. X_test_tfidf = tfidf_vectorizer.transform(X_test)
33.
34. def macro_roc_auc_score(y_true, y_pred_proba):
35.     aucs = []
36.     for i in range(y_true.shape[1]):
37.         auc = roc_auc_score(y_true[:, i], y_pred_proba[:, i])
38.         aucs.append(auc)
39.     return np.mean(aucs)
40.
41. def macro_roc_auc_scorer(estimator, X, y):
42.     y_pred_proba = estimator.predict_proba(X)

```

```

43.     return macro_roc_auc_score(y, y_pred_proba)
44.
45. # Configurar el modelo XGBoost
46. xgb_clf = xgb.XGBClassifier(objective='binary:logistic', use_label_encoder=False, eval_metric='logloss')
47.
48. # Definir los hiperparámetros a ajustar
49. param_distributions = {
50.     'learning_rate': [0.01, 0.1, 0.3],
51.     'max_depth': [3, 5, 7, 10],
52.     'min_child_weight': [1, 3, 5, 10],
53.     'subsample': [0.5, 0.7, 1.0],
54.     'colsample_bytree': [0.5, 0.7, 1.0],
55.     'n_estimators': [50, 100, 200, 500]
56. }
57.
58. # Configurar la búsqueda de hiperparámetros con RandomizedSearchCV
59. opt = RandomizedSearchCV(
60.     estimator=xgb_clf,
61.     param_distributions=param_distributions,
62.     n_iter=5,
63.     scoring=macro_roc_auc_scorer,
64.     cv=2,
65.     n_jobs=-1,
66.     verbose = 1
67. )
68.
69. # Realizar la búsqueda de hiperparámetros
70. opt.fit(X_train_tfidf, y_train)
71.
72. # Obtener los mejores hiperparámetros y el mejor resultado
73. print("Best hyperparameters:", opt.best_params_)
74. print("Best macro ROC AUC:", opt.best_score_)
75.
76. # Evaluar el modelo en el conjunto de prueba
77. best_model = opt.best_estimator_
78. y_pred_proba_test = best_model.predict_proba(X_test_tfidf)
79. test_macro_roc_auc = macro_roc_auc_score(y_test, y_pred_proba_test)
80. print("Test macro ROC AUC:", test_macro_roc_auc)
81.

```

3 Selección del modelo para la competencia

Hasta el momento los modelos probados no lograron tener un desempeño superior a 0.85. De forma intuitiva consideramos que el mejor modelo podría ser una red preentrenada, pero ¿cuál? Al realizar una búsqueda rápida en EBSCO nos llamó la atención el uso de BERT en la solución de este tipo de problemas, particularmente una publicación de Alwyn y otros (2024) quienes refieren haber usado BERT y LSTM para encontrar una solución a este problema, obteniendo en ambos métodos una exactitud superior al 95%. El artículo completo no está disponible, aun así, decidimos probar los dos modelos. No logramos implementar un modelo LSTM adecuado para el conjunto de datos, obteniendo desempeños muy por debajo de los modelos previamente probados (con AUC inferiores a 0.75); por otro lado con el modelo de BERT logramos obtener AUC superiores a 0.89, por lo cual nos concentramos en dicho modelo.

En conclusión, para la competencia seleccionamos un modelo preentrenado de BERT. A continuación describimos su implementación.

- a) Cargamos los datos de training en allData, usamos estos datos para hacer predicciones preliminares antes de subir los datos a la competencia.

```
1. allData = pd.read_csv('dataTraining.zip', encoding='UTF-8', index_col=0)
```

- b) Separamos los predictores y la variable de interés, y transformamos la variable de interés.

```

1. X = allData['plot'].tolist()
2. y = allData['genres'].map(lambda x: eval(x))
3. mlb = MultiLabelBinarizer()
4. y_binarized = mlb.fit_transform(y)
5. # como la transformación de y se hace antes del Split de datos, no es necesario aplicar transform en los datos de prueba.

```

c) Dividimos los datos en conjuntos de entrenamiento y prueba

```
1. X_train, X_test, y_train, y_test = train_test_split(X, y_binarized, test_size=0.2, random_state=42)
```

d) Definimos una clase de Dataset para la compatibilidad con la biblioteca preentrenada

```
1. class MovieGenreDataset(Dataset):
2.     def __init__(self, texts, labels, tokenizer, max_length):
3.         self.texts = texts
4.         self.labels = labels
5.         self.tokenizer = tokenizer
6.         self.max_length = max_length
7.
8.     def __len__(self):
9.         return len(self.texts)
10.
11.    def __getitem__(self, idx):
12.        text = self.texts[idx]
13.        labels = self.labels[idx]
14.        encoding = self.tokenizer(
15.            text,
16.            padding='max_length',
17.            truncation=True,
18.            max_length=self.max_length,
19.            return_tensors='pt'
20.        )
21.        item = {key: val.squeeze(0) for key, val in encoding.items()}
22.        item['labels'] = torch.tensor(labels, dtype=torch.float)
23.        return item
```

e) Preparamos el tokenizer y el dataset con el modelo preentrenado

```
1. tokenizer = BertTokenizer.from_pretrained('bert-base-uncased')
2. train_dataset = MovieGenreDataset(X_train, y_train, tokenizer, max_length=128)
3. test_dataset = MovieGenreDataset(X_test, y_test, tokenizer, max_length=128)
```

Nota: se transforman los datos de entrenamiento y prueba usando el modelo Bert, por la clase implementada no se aplican los métodos `fit_transform`, ni `transform`, sino que se usa la clase definida para la compatibilidad con `transform` de Hugging Face.

f) Configuramos el modelo

```
1. model = BertForSequenceClassification.from_pretrained('bert-base-uncased', num_labels=len(mlb.classes_))
```

4 Calibración y entrenamiento del modelo

- 4.1 Para la calibración del modelo utilizamos el método `ParameterGrid` de `sklearn`. Este es un método similar a `GridSearchCV`, que nos permite probar múltiples combinaciones de parámetros aunque con un costo computacional alto. Los hiperparámetros que se calibraron fueron: `'num_train_epochs'`, `'per_device_train_batch_size'`, `'per_device_eval_batch_size'`, `'learning_rate'`, `'weight_decay'` y `'warmup_steps'`.
- 4.2 Con el fin de mitigar el costo computacional, corrimos el modelo múltiples veces, probando diferentes combinaciones de parámetros, sin que la combinación máxima superara 30 posibilidades.

```
1. from sklearn.model_selection import ParameterGrid
2.
3. param_grid = {
4.     'num_train_epochs': [3, 5],
5.     'per_device_train_batch_size': [8, 16, 32],
6.     'per_device_eval_batch_size': [8, 16, 32],
7.     'learning_rate': [1e-5],
8.     'weight_decay': [0.01],
9.     'warmup_steps': [500]
10. }
11.
12. best_auc = 0
13. best_params = None
14.
15. for params in ParameterGrid(param_grid):
16.     training_args = TrainingArguments(
```

```

17.     output_dir='./results',
18.     num_train_epochs=params['num_train_epochs'],
19.     per_device_train_batch_size=params['per_device_train_batch_size'],
20.     learning_rate=params['learning_rate'],
21.     weight_decay=params['weight_decay'],
22.     warmup_steps=params['warmup_steps'],
23.     logging_dir='./logs',
24.     logging_steps=10,
25.     evaluation_strategy="epoch"
26. )
27.
28. trainer = Trainer(
29.     model=model,
30.     args=training_args,
31.     train_dataset=train_dataset,
32.     eval_dataset=test_dataset
33. )
34.
35. trainer.train()
36. predictions, labels, _ = trainer.predict(test_dataset)
37. predictions = torch.sigmoid(torch.tensor(predictions)).numpy()
38. auc_macro = roc_auc_score(y_test, predictions, average='macro')
39.
40. print(params, " auc: ", auc_macro)
41.
42. if auc_macro > best_auc:
43.     best_auc = auc_macro
44.     best_params = params

```

4.3 Dentro del for que toma la combinación de parámetros de la grilla previamente escrita, se define también el código de entrenamiento del modelo y se obtienen y almacenan las predicciones de cada combinación probada, para seleccionar el mejor modelo probado. Estos son algunos de los resultados obtenidos en la calibración:

- {'learning_rate': 1e-05, 'num_train_epochs': 3, 'per_device_eval_batch_size': 16, 'per_device_train_batch_size': 32, 'warmup_steps': 500, 'weight_decay': 0.01} auc: 0.8722502314632695
- {'learning_rate': 1e-05, 'num_train_epochs': 3, 'per_device_eval_batch_size': 8, 'per_device_train_batch_size': 8, 'warmup_steps': 500, 'weight_decay': 0.01} auc: 0.8767313582601867
- {'learning_rate': 1e-05, 'num_train_epochs': 3, 'per_device_eval_batch_size': 16, 'per_device_train_batch_size': 8, 'warmup_steps': 500, 'weight_decay': 0.01} auc: 0.884627919970742

4.4 De acuerdo, con el comportamiento observado de las diferentes corridas podemos anotar lo siguiente sobre los hiperparámetros calibrados:

- num_train_epochs (Número de épocas de entrenamiento): Indica cuántas veces el modelo pasará por todo el conjunto de entrenamiento. Aumentar el número de épocas puede mejorar el rendimiento del modelo hasta cierto punto; sin embargo, demasiadas épocas pueden llevar al sobreajuste, donde el modelo se ajusta demasiado a los datos de entrenamiento y su rendimiento empeora.
- per_device_train_batch_size (Tamaño del lote de entrenamiento por dispositivo): Indica cuántos ejemplos de entrenamiento se procesan juntos en cada paso de optimización. Un tamaño de lote más grande puede disminuir el tiempo de entrenamiento y a una estimación más estable del gradiente, pero requiere más memoria. Un tamaño de lote más pequeño puede permitir un mejor ajuste en caso de limitaciones de memoria.
- per_device_eval_batch_size (Tamaño del lote de evaluación por dispositivo): Indica cuántos ejemplos de evaluación se procesan juntos durante la evaluación del modelo. Similar al tamaño del lote de entrenamiento, un tamaño de lote de evaluación más grande puede hacer que la evaluación sea más rápida pero requiere más memoria. No afecta directamente al entrenamiento, pero puede influir en la rapidez con que se pueden evaluar los resultados durante la validación.
- learning_rate (Tasa de aprendizaje): Define el tamaño de los pasos que el optimizador dará al actualizar los pesos del modelo. Una tasa de aprendizaje alta puede hacer que el modelo converja rápidamente, pero corre el riesgo de saltarse mínimos óptimos y puede causar oscilaciones o

divergencia. Una tasa de aprendizaje baja puede llevar a una convergencia más estable y precisa, pero también puede hacer que el entrenamiento sea muy lento.

- `weight_decay`: Tasa de decaimiento de los pesos utilizada para evitar el sobreajuste. El decaimiento de los pesos es una forma de regularización que puede ayudar a evitar que el modelo se sobreajuste a los datos de entrenamiento. Sin embargo, si es demasiado alto, puede hacer que el modelo subajuste y no aprenda bien los patrones de los datos.
- `warmup_steps`: Número de pasos de calentamiento durante los cuales la tasa de aprendizaje aumenta linealmente desde cero hasta su valor inicial. Los pasos de calentamiento pueden ayudar a estabilizar el entrenamiento en sus primeras etapas, evitando cambios bruscos en los gradientes que pueden causar inestabilidad. Una fase de calentamiento adecuada puede mejorar la convergencia del modelo.
- La calibración combinada de estos hiperparámetros es crucial para obtener un rendimiento óptimo del modelo BERT

4.5 Hay otro hiperparámetro que no puede pasar desapercibido y es el `'max_length'` de la clase `MovieGenreDataset`. Un mayor valor de este parámetro permite capturar más información y mejora el rendimiento predictivo del modelo, pero requiere más recursos computacionales y más tiempo de entrenamiento. Para efectos del desarrollo de este ejercicio dejamos fijo el valor de este hiperparámetro en 128.

4.6 Seleccionamos tres de las mejores combinaciones obtenidas para realizar las predicciones para la competencia:

- `{'learning_rate': 1e-05, 'num_train_epochs': 3, 'per_device_eval_batch_size': 16, 'per_device_train_batch_size': 8, 'warmup_steps': 500, 'weight_decay': 0.01}` AUC: 0.896693211994311
- `{'learning_rate': 1e-05, 'num_train_epochs': 5, 'per_device_eval_batch_size': 16, 'per_device_train_batch_size': 8, 'warmup_steps': 500, 'weight_decay': 0.01}` AUC: 0.8942387180530393
- `{'learning_rate': 1e-05, 'num_train_epochs': 10, 'per_device_eval_batch_size': 16, 'per_device_train_batch_size': 8, 'warmup_steps': 500, 'weight_decay': 0.01}` AUC: 0.8903683340544387

4.7 Para realizar las predicciones en los datos de prueba de la competencia fue necesario hacer un ajuste en la clase de compatibilidad para no usar `'labels'`, así:

```
1. class MovieGenreTestDataset(Dataset):
2.     def __init__(self, texts, tokenizer, max_length):
3.         self.texts = texts
4.         self.tokenizer = tokenizer
5.         self.max_length = max_length
6.
7.     def __len__(self):
8.         return len(self.texts)
9.
10.    def __getitem__(self, idx):
11.        text = self.texts[idx]
12.        encoding = self.tokenizer(
13.            text,
14.            padding='max_length',
15.            truncation=True,
16.            max_length=self.max_length,
17.            return_tensors='pt'
18.        )
19.        item = {key: val.squeeze(0) for key, val in encoding.items()}
20.        return item
```

Los resultados obtenidos en la competencia con este modelo, y los hiperparámetros anotados en el punto anterior fueron:

	pred_genres_text_RF_Bert_E03 (1).csv Complete · now	0.88587	<input type="checkbox"/>
	pred_genres_text_RF_Bert_E10.csv Complete · 24s ago	0.89249	<input type="checkbox"/>
	pred_genres_text_RF_Bert_E03.csv Complete · 3m ago	0.90553	<input type="checkbox"/>

- 4.8 Realizamos un ensamble promediando estos tres resultados, y los subimos a la competencia, obteniendo un desempeño superior a 0.91 en la métrica de la competencia.
- 4.9 Ya cerrando la competencia cambiamos el hiperparámetro 'max_length' de la clase MovieGenreDataset a 256 y el número de épocas a 5 y 10; promediamos los resultados obtenidos y los subimos los resultados a la competencia y obteniendo nuestra mejor predicción con un score de 0.91754.

5 Publicación del modelo

Para la publicación del modelo se realizaron los siguientes pasos:

- Se creó en el servicio EC2 de AWS Cloud la instancia t2.small donde se habilitó el puerto 5000 para consumir la api. [Este es el enlace a la API](#). En caso que no esté disponible al momento de probarla por favor contactarnos.
- El api tiene una única variable de entrada que corresponde al plot con la descripción de la película.

```
17. parser.add_argument(
18.     'Plot',
19.     type=str,
20.     required=True,
21.     help='Movie description',
22.     location='args')
```

- Una vez realizadas las predicciones y obtenidas las probabilidades, se definió un umbral de 0.5 para determinar los géneros válidos para el plot enviado a la API.

```
1. umbral = 0.5
2.
3. # Encontrar los índices donde numeros es igual a 1
4. indices_1 = np.where(prediction >= umbral)[1]
5.
6. total_genres = len(indices_1)
7.
8. if total_genres == 0:
9.     rta = 'No fue posible establecer el genero de la pelicula'
10. else:
11.     generos = ['Action', 'Adventure', 'Animation', 'Biography', 'Comedy', 'Crime', 'Documentary', 'Drama', 'Family',
12.               'Fantasy', 'Film-Noir', 'History', 'Horror', 'Music', 'Musical', 'Mystery', 'News', 'Romance',
13.               'Sci-Fi', 'Short', 'Sport', 'Thriller', 'War', 'Western']
14.
15. # Obtener las palabras correspondientes a los índices encontrados
16. genero = [generos[i] for i in indices_1]
17. rta = f'Movie genres: {", ".join(genero)}'
```

- Se implementó Docker para incluir dentro de un container todo lo necesario para ejecutar la aplicación. Para esto se desarrolló el respectivo Dockerfile, donde dentro del folder proyecto tenemos los archivos: model.pkl: librería del modelo desarrollado; app.py : api desarrollada; requirements.txt: contiene las diferentes librerías de Python utilizadas en la api
- Una vez creado el container, se ejecuta y podemos consultar en el navegador la api desarrollada, se probaron 4 plot del set de prueba obteniendo los siguientes resultados:

- "alison parker is a successful top model , living with the lawyer michael lerman in his apartment . she tried to commit suicide twice in the past : the first time , when she was a teenager and saw her father cheating her mother with two women in her home , and then when michael 's wife died . since then , she left christ and the catholic church behind . alison wants to live alone in her own apartment and with the help of the real state agent miss logan , she finds a wonderful furnished old apartment in brooklyn heights for a reasonable rental . she sees a weird man in the window in the last floor of the building , and miss logan informs that he is father francis matthew halloran , a blinded priest who lives alone supported by the catholic church . alison moves to her new place , and

once there , she receives a visitor : her neighbor charles chazen welcomes her and introduces the new neighbors to her . then , he invites alison to his cat jezebel ' s birthday party in the night . on the next day , weird things happen with alison in her apartment and with her health . alison looks for miss logan and is informed that she lives alone with the priest in the building . a further investigation shows that all the persons she knew in the party were dead criminals . frightened with the situation , alison embraces christ again , while michael investigates the creepy events . alison realizes that she is living in the gateway to hell . " { "result": "Movie genres: Drama, Horror, Thriller" }

- "last time we see saw bella swan she was narrowly escaping the clutches of the evil vampire james while finding love with ' ' vegetarian ' ' vampire edward cullen . bella and edward ' s lives have been full of nothing but love and bliss however , it all changes one fateful day . on bella ' s birthday , her new found friend and sister of edward , alice , decides to throw her lavish party , complete with balloons , ribbons and cake that could feed an army . all is well until bella accidentally cuts her finger whilst opening a present . the result is that jasper hale , the newest addition to the cullen clan , succumbs to his blood lust and attacks bella . edward decides that while he and his family are around , bella ' s life will always be at risk . so he decides to leave her for her own good . bella feels her life is over . enter jacob black , a member of the quilite tribe who manages to bring some joy and meaning back into bella ' s life . however as the two become closer , bella discovers jacob has a secret of his own - he ' s a werewolf . as if that wasn ' t bad enough bella can ' t seem to get the love of her life , edward out of her mind . with new dangers , new friends and new enemies , bella finds herself choosing between holding on to the past or accepting a new future . but what and more importantly who will she choose ?" { "result": "Movie genres: Adventure, Drama, Fantasy, Romance" }
- "at her husband ' s funeral , pearl (shirley maclaine) , jewish mother of two divorced and antagonistic daughters , meets an old italian friend (marcello mastroianni) of her husband , whose advice years previously had stopped the husband leaving home . for N years he , now a widower , has secretly loved pearl . . ." { "result": "Movie genres: Comedy, Drama, Romance" }
- "john rambo is a vietnam veteran , winner of the medal of honour for serving his country in the vietnam war and the last surviving member of the unit he was in . rambo arrives in a small town , where he is arrested by the abusive local sheriff will tease for refusing to leave town . rambo is mistreated and he relives his painful memories of being tortured in a prison camp , which goes too far and rambo escapes from police custody . rambo is pursued by tease and the local police into the woods and rambo begins a personal war with tease , and uses his combat skills and hunts down tease and his men . rambo ' s former commanding officer colonel samuel trautman arrives believing tease and his men don ' t stand a chance with rambo , and tries to put rambo ' s personal war to a end , as tease wants rambo dead . " { "result": "Movie genres: Drama, Thriller, War" }

6 Conclusiones

- 6.1 La red neuronal pre entrenada puesta a prueba en este contexto presentó los mejores resultados (esto era de esperarse teniendo en cuenta que ha sido entrenada con millones de parámetros por lo que termina siendo altamente robusta)
- 6.2 Los métodos de tokenización convencionales y las redes neuronales también generaron resultados satisfactorios a pesar de no ser los mejores modelos para este contexto.
- 6.3 Aunque las redes neuronales pre entrenadas suelen generar los mejores resultados al tener los entrenamientos más robustos, la implementación y calibración no es tan sencilla como con los demás métodos convencionales.
- 6.4 Las redes neuronales pre entrenadas demandan la implementación de GPU, de no ser así el gasto computacional es tan severo que no hace sentido implementarlas.
- 6.5 El mejor desempeño en la competencia se logro utilizando la red preentrenada de BERT, sin embargo, no se logro calibrar para obtener los resultados de alto desempeño mencionados por Alwyn y otros (2024). Para este set de datos los mejores resultados se obtuvieron con un número de épocas pequeño menor a 10, siendo el mejor resultado obtenido con 3 épocas.
- 6.6 Tal como se ha evidenciado en otros desarrollos en general el valor de los hiperparámetros en relación con el desempeño predictivo del modelo describe una curva en la que el incremento o decremento del valor del hiperparámetro mejora el desempeño hasta cierto punto, a partir del cual el modelo se sobreajusta y pierde poder predictivo.

7 Referencias

Alwyn, A., Pranoto, E. J. P., Ichsan, I., Halim, K., Justin, W., & Girsang, A. S. (2024). Movie genre classification using BERT and LSTM. AIP Conference Proceedings, 2927(1), 1–5. <https://doi-org.ezproxy.uniandes.edu.co/10.1063/5.0193424>