Project 2

Movie Genre Classification

API: http://18.216.214.146:8888/ (http://18.216.214.146:8888/)

Team12 (Grupo en Kaggle)

Angélica Viviana Parrado Cubillos

Robert Angarita Bermúdez

Clasificación de texto

Introducción

La clasificación de texto es una de las tareas más importantes en el procesamiento del lenguaje natural. Es el proceso de clasificación de cadenas de texto o documentos en diferentes categorías, dependiendo del contenido de las cadenas. La clasificación de texto tiene una variedad de aplicaciones, como detectar el sentimiento del usuario a partir de un tweet, clasificar un correo electrónico como correo no deseado o spam, clasificar publicaciones de blogs en diferentes categorías, etiquetar automáticamente las consultas de los clientes, etc.

El presente proyecto tiene como objetivo la clasificación de texto en el mundo real. Entrenaremos un modelo de aprendizaje automático capaz de predecir a qué género pertenece la película dado la synapsis. Este es un proyecto de análisis para clasificar un género en particular.

Conjunto de datos

El conjunto de datos de entrenamiento que vamos a utilizar para este proyecto corresponde a los títulos, reseñas y genero. El conjunto de datos consta de un total de 7895 registros. La mitad de los documentos contienen comentarios positivos sobre una película, mientras que la mitad restante contiene comentarios negativos. Más detalles sobre el conjunto de datos se pueden encontrar en este enlace .

Clasificación del genero para películas

En este proyecto, realizaremos una serie de pasos necesarios para predecir los géneros de las reseñas de diferentes películas. Estos pasos se pueden utilizar para cualquier tarea de clasificación de texto. Utilizaremos la biblioteca Scikit-Learn de Python para el aprendizaje automático para entrenar un modelo de clasificación de texto.

Los siguientes son los pasos necesarios para crear un modelo de clasificación de texto en Python:

1. Importando Bibliotecas
2. Importando el conjunto de datos
3. Preprocesamiento de texto
4. CountVectorizer
5. Conjuntos de entrenamiento y prueba

6. Modelo de clasificación de textos de entrenamiento y predicción de sentimientos

1. Importando Bibliotecas

Ejecutamos el siguiente script para importar las bibliotecas requeridas:

In [8]:
```
import pandas as pd
import os
import numpy as np
from sklearn.feature_extraction.text import CountVectorizer
from sklearn.preprocessing import MultiLabelBinarizer
from sklearn.multiclass import OneVsRestClassifier
from sklearn.ensemble import RandomForestRegressor, RandomForestClassifier, ExtraTreesClassi
from sklearn.metrics import r2_score, roc_auc_score
from sklearn.model_selection import train_test_split
from sklearn.ensemble import VotingClassifier

import xgboost as xgb
import sklearn.model_selection
```

2. Importando el conjunto de datos

Utilizaremos la función *PANDAS* de la biblioteca sklearn_datasets para importar el conjunto de datos a nuestra aplicación. La función toma el nombre asignado a cada conjunto de datos en conjuntos de datos y objetivos. Por ejemplo, en nuestro caso, la ruta del directorio "dataTraining.csv y dataTesting.csv" debe ser la misma donde se está ejecutando el *NOTEBOOK JUPYTER*. Los pandas tratarán cada carpeta dentro de la misma carpeta como una categoría y a todos los documentos dentro de esa carpeta se les asignará su categoría correspondiente, de entrenamiento o testeo.

Ejecutamos el siguiente script para ver la función pandas en acción:

In [9]:
```
dataTraining = pd.read_csv('dataTraining.csv', encoding='UTF-8', index_col=0)
dataTesting = pd.read_csv('dataTesting.csv', encoding='UTF-8', index_col=0)
```

En el script anterior se cargan los dos dataframe de entrenamiento y test.

3. Preprocesamiento de texto

Una vez que se haya importado el conjunto de datos, el siguiente paso es preprocesar el texto. Ejecute el siguiente script para preprocesar los datos:

In [10]:
```
from nltk.stem.snowball import SnowballStemmer
stemmer = SnowballStemmer('english')
def split_into_lemmas(text):
    text = text.lower()
    words = text.split()
    return [stemmer.stem(word) for word in words]
```

Steamming y lemmatization son los métodos básicos de procesamiento de texto. El objetivo de la derivación de éstas, es reducir las formas de inflexión y, a veces las formas relacionadas de una palabra a una base común. Utilizamos el metodo Steamming con Snowball que es un lenguaje de procesamiento de cadenas pequeñas diseñado para crear algoritmos de derivación para su uso en la recuperación de información.

4. Creamos count vectorizer

In [11]:
```
vect = CountVectorizer(max_features=11000,stop_words='english',lowercase=True,analyzer=split
X_dtm = vect.fit_transform(dataTraining['plot'])
X_dtm.shape
```

Out[11]:
```
(7895, 11000)
```

El script anterior utiliza la clase CountVectorizer de la biblioteca sklearn.feature_extraction.text . Hay algunos parámetros importantes que deben pasarse al constructor de la clase. El primer parámetro es el parámetro max_features , que se establece en 11000. Esto se debe a que cuando convierte palabras en números utilizando la bolsa de palabras, todas las palabras únicas en todos los documentos se convierten en características. Todos los documentos pueden contener decenas de miles de palabras únicas. Pero las palabras que tienen una frecuencia de aparición muy baja no son, en general, un buen parámetro para clasificar documentos. Por lo tanto, establecemos el parámetro max_features en 11000, lo que significa que queremos usar las 11000 palabras más comunes como características para entrenar a nuestro clasificador.

La función fit_transform de la clase CountVectorizer convierte los documentos de texto en las características numéricas correspondientes.

Creamos la variable Y

In [12]:
```
dataTraining['genres'] = dataTraining['genres'].map(lambda x: eval(x))
le = MultiLabelBinarizer()
y_genres = le.fit_transform(dataTraining['genres'])
```

5. Conjuntos de entrenamiento y pruebas

Al igual que cualquier otro problema de aprendizaje automático supervisado, debemos dividir nuestros datos en conjuntos de capacitación y pruebas. Para hacerlo, usaremos la utilidad train_test_split de la biblioteca sklearn.model_selection . Ejecutamos el siguiente script:

In [13]:
```
X_train, X_test, y_train_genres, y_test_genres = train_test_split(X_dtm, y_genres,test_size=
```

El script anterior divide los datos en 25% de conjunto de prueba y 75% de conjunto de entrenamiento.

6. Modelo de clasificación de textos de entrenamiento y predicción de género

Hemos dividido nuestros datos en conjunto de entrenamiento y pruebas. Ahora es el momento de ver la acción real. Usaremos el algoritmo de bosque aleatorio para entrenar nuestro modelo. Puedes usar cualquier otro modelo de tu elección.

Para este notebook del proyecto *Movie Genre Classification*, combinaremos las predicciones de 2 clasificadores diferentes, con la parametrización específica.

Los clasificadores son:

1) `ExtraTreesClassifier`
2) `RandomForestClassifier`

Vemos que podemos mejorar los rendimientos combinando predicciones a través de *"Voting Soft"*. *Voting Soft* implica calcular una suma ponderada de las probabilidades pronosticadas de todos los modelos para cada clase. Si somos iguales para todos los clasificadores, entonces las probabilidades son simplemente los

promedios para cada clase.

Voting Soft se puede realizar fácilmente usando un VotingClassifier, que "volverá a entrenar" el modelo de prototipo que especificamos anteriormente. Debemos tener en cuenta que uno podría evitar volver a capacitar a los modelos (en este caso) y realizar manualmente *Voting Soft* utilizando la salida de cada método clasificador "predic_proba".

El método de fit de esta clase se utiliza para entrenar el algoritmo. Necesitamos pasar los datos de entrenamiento y los conjuntos de objetivos de entrenamiento a este método. Echa un vistazo a la siguiente secuencia de comandos:

In [14]:
```
%%time
clf1 = OneVsRestClassifier(ExtraTreesClassifier(n_jobs=-1, n_estimators=700, max_depth=24, r
clf1.fit(X_train, y_train_genres)

clf2 = OneVsRestClassifier(xgb.XGBClassifier(objective ="multi:softprob",num_class = 24,seed
clf2.fit(X_train, y_train_genres)

clf = OneVsRestClassifier(VotingClassifier(estimators=[('etc', clf1), ('xgb', clf2)], voting
clf.fit(X_train, y_train_genres)
```

```
Wall time: 2h 51s
```

Finalmente, para predecir los géneros de los conjunto de prueba, podemos usar el método de predict de la clase *Weighted Average Probabilities (Soft Voting)* como se muestra a continuación:

In [15]:
```
y_pred_genres = clf.predict_proba(X_test)
```

Se ha entrenado con éxito el modelo de clasificación ENSEMBED de texto y ha realizado todas predicciones. A continuación es el momento de ver el rendimiento del modelo que acabamos de desarrollar.

In [16]:
```
print("AUC Score = ", roc_auc_score(y_test_genres, y_pred_genres, average='macro'))
```

```
AUC Score =  0.8703638058158182
```

Predecimos *the testing datasets*

In [17]:
```
X_test_dtm = vect.transform(dataTesting['plot'])

cols = ['p_Action', 'p_Adventure', 'p_Animation', 'p_Biography', 'p_Comedy', 'p_Crime', 'p_D
        'p_Fantasy', 'p_Film-Noir', 'p_History', 'p_Horror', 'p_Music', 'p_Musical', 'p_Myst
        'p_Sci-Fi', 'p_Short', 'p_Sport', 'p_Thriller', 'p_War', 'p_Western']

y_pred_test_genres = clf.predict_proba(X_test_dtm)
```

In [25]:
```
res = pd.DataFrame(y_pred_test_genres, index=dataTesting.index, columns=cols)
```

In [ ]:
```

```