UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

FACULTAD DE INGENIERÍA

DIVISIÓN DE INGENIERÍA ELÉCTRICA

PROYECTO FINAL

“CALIFICADOR DE PELÍCULAS BASADO EN CRÍTICAS”

BARCENAS MARTÍNEZ EDGAR DANIEL

MARTÍNEZ TRONCOSO JULIO CÉSAR

SILVA SANDOVAL CECILIA

ANÁLISIS Y PROCESAMIENTO INTELIGENTE DE TEXTOS

PROFESOR: M.P. OCTAVIO AUGUSTO SANCHEZ VELAZQUEZ

2932 - 1

FECHA DE ENTREGA: 12/06/2020

SEMESTRE 2020-2

ÍNDICE

INTRODUCCIÓN

Somos un equipo amante del arte en expresión cinematográfica.
Actualmente la gran cantidad de películas que se han producido, que se están produciendo y se producirán, nos hace cuestionarnos en la gran cantidad que esto representa si le pusieramos un número y siendo realistas nunca vamos a poder ver todas las películas existentes, teniendo como principal limitante: al tiempo.
Esto nos presenta un problema interesante a resolver, ¿Habrá alguna forma de calificar una película antes de verla? pero sin las reseñas de otras personas (somos seres bastante complejos, cada quien de acorde a sus vivencias, cultura, influencias y gustos puede disfrutar de una tipo de película o no. Entonces ¿Cómo hacer esto sin involucrar una análisis estadístico de reseñas de personas? y la respuesta común que encontramos es por el texto del que viene acompañada toda película, es decir su reseña y valorando el tiempo, ¿Por qué no entrenar varios algoritmos para ayudarnos a calificar una película antes de verla?, con el objetivo de tener un resultado a considerar para saber si una película será una buena función e inversión de tiempo.

MARCO TEÓRICO-METODOLÓGICO

Realizamos un análisis de varias funciones en los métodos que nos proporciona el lenguaje de programación python, para asegurarnos de tener distintos rangos de exactitud en cuanto a nuestras predicciones, ya que no nos limitamos a solo una en general, creemos que para ser un comienzo lo haremos bastante bien.

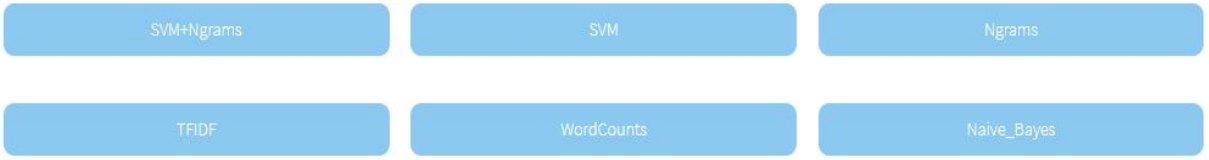
Creemos en que podemos orientar adecuadamente al usuario a poder definir su elección de película, poniendo a su disposición varios métodos y algoritmos nuestros para evaluar la misma.

Se tomó como iniciativa un trabajo encontrado en el 2016 publicado por la colaboración de estudiantes de la Benemérita Universidad Autónoma de Puebla , Universidad Autónoma Metropolitana , donde se hacía un Análisis comparativo entre diferentes entornos de aprendizaje automático para el análisis de sentimientos, el objetivo de este artículo era el aprendizaje automático para realizar una comparativa de clasificadores implementados en una plataforma de aprendizaje WEKA y el lenguaje de programación python.

De la misma forma quisimos aplicar nuestros conocimientos previos en el software sklearn (biblioteca para aprendizaje automático de software libre para el lenguaje de programación Python), para realizar una comparativa de las películas con las que nos encontramos dia a dia , y de la misma forma , adentrarnos en distintas formas de clasificarlos , así como verificar la precisión que cada método tiene.

En proyectos escolares , la forma más normal en la que se desarrolla esta búsqueda es de manera sistemática , en comparar cada palabra con una anterior.Esta no es la manera más elegante de realizar este procesamiento de textos, ni la más precisa.Creemos que el hecho de utilizar una herramienta de aprendizaje acelerará nuestra búsqueda y nos dará mejores resultados en cuanto a la importancia de las palabras, ya que se le está dando un sentido mayor que la simple posición en un arreglo unidimensional.

- Descripción:
Se decidió realizar una webApp para que cualquier usuario por medio de la url pusiera consultar nuestra aplicación para calificar su película ingresando la reseña en formato texto o adjuntando un archivo en el mismo formato.
De la misma forma puede hacer uso de cualesquiera de los siguientes métodos:



Se hizo uso de un micro framework llamado Flask es open source y trabaja con python para crear aplicaciones web, es decir, páginas web dinámicas, APIs, etc …
En este caso con Flask nos toma un promedio de 9 líneas de código tener un servidor completamente funcional.
Y como punto de partida lo consideramos óptimo ya que en un futuro si se requieren agregar más más funcionalidades no permitirá escalar el proyecto.

MÉTODO EXPERIMENTAL

DATOS:
Se utilizó un conjunto de datos dados por la universidad de Stanford. El dataset tiene 50,000 críticas de películas donde de estas la mitad son para entrenamiento el la otra para prueba. Cada parte tiene 12,500 críticas positivas y 12,500 críticas negativas. El dataset se obtuvo del siguiente link https://ai.stanford.edu/~amaas/data/sentiment/.
Para tener un mejor control de los datos creamos un script en bash para poder combinar las críticas y de este obtener 2 archivos txt que contengan las reviews para procesarlas de una mejor manera.

```bash
#!/bin/bash
gunzip -c aclImdb_v1.tar.gz | tar xopf -
cd aclImdb
mkdir movie_data
for split in train test;
do
  for sentiment in pos neg;
  do
    for file in $split/$sentiment/*;
    do
                cat $file >> movie_data/full_${split}.txt;
                echo >> movie_data/full_${split}.txt;
        # Esta línea agrega archivos que contienen las rev
              # echo $file | cut -d '_' -f 2 | cut -d "." -
    done;
  done;
done;
```

Script:
Combina las reviews positivas y negativas y las divide en el set de entrenamiento y prueba

El resultado es que nos creó un folder llamado movie_data, este folder contiene nuestros 2 archivos para trabajar con las críticas, uno es para el entrenamiento y otras para el test.

Conjunto de datos de entrenamiento y prueba.

DESCRIPCIÓN DE LOS MÉTODOS:

Para clasificar utilizamos diferentes métodos de clasificación.

N-grams: El modelo de n-grams es un modelo probabilístico que permite hacer predicciones estadísticas del próximo elemento de un conjunto de elementos. Puede ser definido como una cadena de cadena de Márkov. Con n-grams podemos agrupar datos en paquete de n elementos.

TF-IDF: (Term frequency – Inverse document frequency)
Obtiene con base a la siguiente fórmula una indicador numérico que mide la frecuencia de las palabras de un texto:
w (ij) = tf (ij) * log2 N / n
wij = peso del término Tj en el documento Di
tfij = frecuencia del término Tj en el documento Di
N = número de documentos en el corpus
n = número de documentos donde el término Tj aparece al menos una vez
Representa el número de veces que una palabra aparece en un documento.

Word Counts: Es una función que retorna el número de ocurrencias de una palabra dentro de un texto.
Funciona de la siguiente manera, se inicializa una variable contador y una para la palabra la cual sirve para comparación que al encontrar un valor exactamente igual, aumenta la variable contador, hasta que recorre todo el texto, regresa la variable contador.

Naive Bayes: Este módulo tiene un algoritmos orientados al aprendizaje supervisado basado en la aplicación del teorema de Bayes con el supuesto de independencia condicional entre cada par de características dado el valor de la variable de clase.
Funcionan bastante bien en clasificación de documentos y filtrado de spam. Solo requieren una pequeña cantidad de datos de entrenamiento para estimar los parámetros necesarios.

SVM: Máquinas de vector soporte o Support Vector Machines (SVM) son algoritmos de machine learning supervisado aplicable a modelos de clasificación. Las máquinas de vector soporte suponen una generalización de un clasificador simple denominado maximal margin classifier. Sin embargo, este clasificador no puede aplicarse a sets de datos donde las clases de la variable respuesta no son separables mediante un límite lineal.

DESCRIPCIÓN DEL EXPERIMENTO:

1.Preparación de los Datos:

Leemos el conjunto de datos de entrenamiento y prueba:
Con los datos que tenemos preparados vamos a leerlos y procedemos a limpiarlos les quitaremos los signos de puntuación, quitaremos etiquetas html y lo volveremos minúsculas.

```
'''Lectura de datos'''
def leer_review_entrenamiento():
    reviews_train = []
    for line in open('movie_data/full_train.txt', 'r'):
        reviews_train.append(line.strip())
    return reviews_train

def leer_review_prueba():
    reviews_test = []
    for line in open('movie_data/full_test.txt', 'r'):
        reviews_test.append(line.strip())
    return reviews_test
```

Métodos de Lectura de datos

```
Metodo de limpieza de las review
Quitamos signos de puntuacion.
Etiquetas HTML y conversion del texto a minuscula
'''
def preprocesamiento_reviews(reviews):
    remplazar_sin_espacio = re.compile("(\.)|(\;)|(\:)|(\!)|(\?)|(\,)|(\")|(\()|(\))|(\[)|(
    remplazar_con_espacio = re.compile("(<br\s*/><br\s*/>)|(\-)|(\/)")
    sin_espacio = ""
    espacio = " "
    reviews = [remplazar_sin_espacio.sub(sin_espacio, line.lower()) for line in reviews]
    reviews = [remplazar_con_espacio.sub(espacio, line) for line in reviews]
    return reviews
```

Método de preprocesamiento

2.-Vectorización:
Utilizamos vectorización para convertir el texto en una representación numérica, esto es para que los datos puedan ser procesados por nuestros algoritmos de aprendizaje automático .

```
cv = CountVectorizer(binary=True)
cv.fit(reviews_train_clean)
X = cv.transform(reviews_train_clean)
X_test = cv.transform(reviews_test_clean)
```

Para esto se crea una matriz con una columna para cada palabra única del dataset, luego transformamos el texto en una fila que contiene 0s y 1s esto quiere decir que las palabras con 1 son las que coinciden en nuestra review y en el dataset.

3.-Construir clasificación:
Construimos 6 modelos con la finalidad de encontrar la mejor precisión de los datos. Para la construcción de los modelos utilizaremos el mismo número de targets y etiquetas para el entrenamiento y pruebas porque estructuramos los datos de la misma forma con 12,500 críticas positivas y 12,5000 críticas negativas para cada cada set de datos es decir el de entrenamiento y prueba.

```
target = [1 if i < 12500 else 0 for i in range(25000)]
```

Para encontrar una mejor exactitud probaremos varios valores de C = 1 / λ que es el parámetro de regularización inversa, utilizaremos los siguientes valores 0.01, 0.05, 0.25, 0.5, 1 para encontrar el modelo más preciso. Este parámetro tiene la finalidad de aplicar una penalización para aumentar la magnitud de los valores de los parámetros con el fin de reducir el sobre ajuste. Con C vamos a poder encontrar el mejor ajuste de nuestro modelo.

Ngrams:
Utilizamos este modelo para agregar más precisión al modelo, con n-grams agregamos secuencias de dos o 3 palabras estos se pueden llamar bigramas o trigramas, para este método utilizamos una secuencia de 2 palabras para que no sean consideradas de forma individual. En este método incluimos pares de palabras.
Prueba para encontrar la mejor exactitud del modelo.

```
ngram_vectorizer = CountVectorizer(binary=True, ngram_range=(1, 2))
ngram_vectorizer.fit(reviews_train_clean)
X = ngram_vectorizer.transform(reviews_train_clean)
X_test = ngram_vectorizer.transform(reviews_test_clean)

X_train, X_val, y_train, y_val = train_test_split(
    X, target, train_size = 0.75
)

for c in [0.01, 0.05, 0.25, 0.5, 1]:
    lr = LogisticRegression(C=c)
    lr.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(y_val, lr.predict(X_val))))

# Exactitud para C=0.01: 0.88416
# Exactitud para C=0.05: 0.892
# Exactitud para C=0.25: 0.89424
# Exactitud para C=0.5: 0.89456
# Exactitud para C=1: 0.8944

final_ngram = LogisticRegression(C=0.5)
final_ngram.fit(X, target)
print ("Final Accuracy: %s"
       % accuracy_score(target, final_ngram.predict(X_test)))
```

Con este método podemos decir que al considerar secuencias de 2 palabras además de secuencias simples nos daría un modelo con precisión del **0.898**

Word Counts:

Con este método planteamos incluir la cantidad de veces que aparece una palabra determinada en el texto. Con este método pretendemos mejorar la predicción de nuestro modelo. Es decir con este método si una palabra es clasificada como positiva y aparece muchas veces en el texto es posible que el resultado sea positivo.

```
wc_vectorizer = CountVectorizer(binary=False)
wc_vectorizer.fit(reviews_train_clean)
X = wc_vectorizer.transform(reviews_train_clean)
X_test = wc_vectorizer.transform(reviews_test_clean)

X_train, X_val, y_train, y_val = train_test_split(
    X, target, train_size = 0.75,
)

for c in [0.01, 0.05, 0.25, 0.5, 1]:

    lr = LogisticRegression(C=c)
    lr.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(y_val, lr.predict(X_val))))

# Exactitud para C=0.01: 0.87456
# Exactitud para C=0.05: 0.88016
# Exactitud para C=0.25: 0.87936
# Exactitud para C=0.5: 0.87936
# Exactitud para C=1: 0.87696

final_wc = LogisticRegression(C=0.05)
final_wc.fit(X, target)
print ("Final Accuracy: %s"
       % accuracy_score(target, final_wc.predict(X_test)))

# Final Exactitud: 0.88184
```

Con este método de clasificación logramos una exactitud de **0.88184**

TFIDF:

Con este método TF IDF (término frecuencia de documento inversa de frecuencia) tenemos que cada palabra será utilizada como un factor de ponderación en vez de utilizar representaciones binarias o de recuento de palabras. Con este método representamos la cantidad de veces que aparece una palabra determinada en un texto en relación con la cantidad de documentos en el dataset, las palabras que más aparecen en el dataset tienen valores cercanos a 0 y las palabras que aparecen menos tienen valores cercanos a 1.

```
tfidf_vectorizer = TfidfVectorizer()
tfidf_vectorizer.fit(reviews_train_clean)
X = tfidf_vectorizer.transform(reviews_train_clean)
X_test = tfidf_vectorizer.transform(reviews_test_clean)

X_train, X_val, y_train, y_val = train_test_split(
    X, target, train_size = 0.75
)

for c in [0.01, 0.05, 0.25, 0.5, 1]:

    lr = LogisticRegression(C=c)
    lr.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(y_val, lr.predict(X_val))))

# Exactitud para C=0.01: 0.79632
# Exactitud para C=0.05: 0.83168
# Exactitud para C=0.25: 0.86768
# Exactitud para C=0.5: 0.8736
# Exactitud para C=1: 0.88432

final_tfidf = LogisticRegression(C=1)
final_tfidf.fit(X, target)
print ("Final Accuracy: %s"
       % accuracy_score(target, final_tfidf.predict(X_test)))

# Final Exactitud: 0.882
```

Con este método de clasificación logramos una exactitud de **0.882**

SVM:

Con SVM vamos a representar cada texto como un vector escaso es decir con muchos ceros con un espacio para cada n grams único en el dataset. SVM es un clasificador lineal y funciona mejor con datos escasos. Para este método de clasificación utilizaremos n-gramas de 1 a 2.

```
ngram_vectorizer = CountVectorizer(binary=True, ngram_range=(1, 2))
ngram_vectorizer.fit(reviews_train_clean)
X = ngram_vectorizer.transform(reviews_train_clean)
X_test = ngram_vectorizer.transform(reviews_test_clean)

X_train, X_val, y_train, y_val = train_test_split(
    X, target, train_size = 0.75
)

for c in [0.01, 0.05, 0.25, 0.5, 1]:

    svm = LinearSVC(C=c)
    svm.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(y_val, svm.predict(X_val))))

# Exactitud para C=0.01: 0.89104
# Exactitud para C=0.05: 0.88736
# Exactitud para C=0.25: 0.8856
# Exactitud para C=0.5: 0.88608
# Exactitud para C=1: 0.88592

final_svm_ngram = LinearSVC(C=0.01)
final_svm_ngram.fit(X, target)
print ("Final Accuracy: %s"
       % accuracy_score(target, final_svm_ngram.predict(X_test)))

# Final Exactitud: 0.8974
```

Con este método de clasificación logramos una exactitud de **0.8974**

Naive Bayes:

El clasificador de Naive Bayes es simple funciona clasificando en función de las probabilidades de las palabras.

```
def Naive_Bayes(reviews_train_clean,reviews_test_clean):
    target = [1 if i < 12500 else 0 for i in range(25000)]
    stop_words = ['in', 'of', 'at', 'a', 'the']
    movie_vec = CountVectorizer(binary=True, ngram_range=(1, 3), stop_words=stop_words)
    movie_vec.fit(reviews_train_clean)
    X = movie_vec.transform(reviews_train_clean)
    X_test = movie_vec.transform(reviews_test_clean)
    X_train, X_val, y_train, y_val = train_test_split(X, target, train_size = 0.75,)
    final_movie_vec = MultinomialNB()
    final_movie_vec.fit(X, target)
    exactitudFinal(final_movie_vec,target,X,X_test)
    return final_movie_vec,movie_vec
```

Final Exactitud: 0.88164

Modelo Final:

SVM con N Grams(1,3) y con un conjunto de palabras funcionales de menor cantidad.

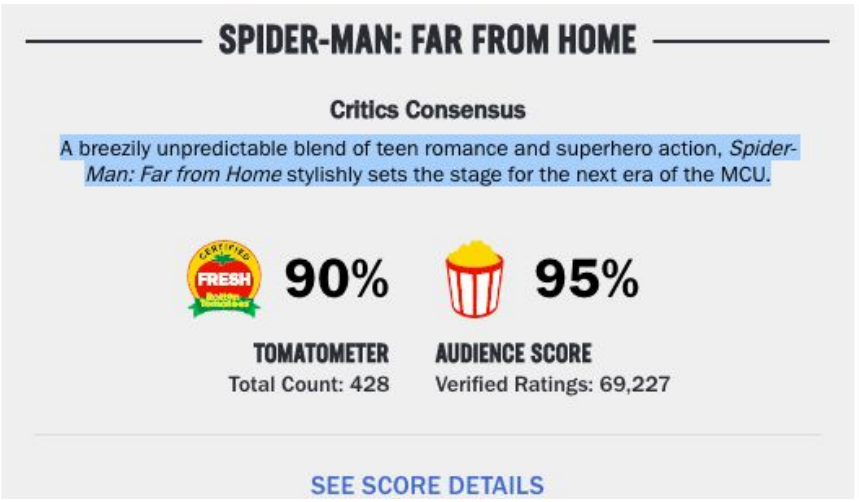
palabras funcionales = ['in' , 'of' , 'at' , 'a' , 'the']

```
stop_words  = [ 'in' , 'of' , 'at' , 'a' , 'the' ]
ngram_vectorizer  =  CountVectorizer ( binario = True , ngram_range = ( 1 , 3 ), stop_words = stop_words )
ngram_vectorizer . ajuste ( reviews_train_clean )
X  =  ngram_vectorizer . transformar ( reviews_train_clean )
X_test  =  ngram_vectorizer . transformar ( reviews_test_clean )

X_train , X_val , y_train , y_val  =  train_test_split (
    X , target , train_size  =  0.75
)

stop_words = ['in', 'of', 'at', 'a', 'the']
ngram_vectorizer = CountVectorizer(binary=True, ngram_range=(1, 3), stop_words=stop_words)
ngram_vectorizer.fit(reviews_train_clean)
X = ngram_vectorizer.transform(reviews_train_clean)
X_test = ngram_vectorizer.transform(reviews_test_clean)

X_train, X_val, y_train, y_val = train_test_split(
    X, target, train_size = 0.75
)

for c in [0.001, 0.005, 0.01, 0.05, 0.1]:

    svm = LinearSVC(C=c)
    svm.fit(X_train, y_train)
    print ("Accuracy for C=%s: %s"
           % (c, accuracy_score(y_val, svm.predict(X_val))))

# Precisión para C = 0.001: 0.88784
# Precisión para C = 0.005: 0.89456
# Precisión para C = 0.01: 0.89376
# Precisión para C = 0.05: 0.89264
# Precisión para C = 0.1: 0.8928

final = LinearSVC(C=0.01)
final.fit(X, target)
print ("Final Accuracy: %s"
       % accuracy_score(target, final.predict(X_test)))

# Final Precisión: 0.90064
```

Con estos cambios logramos un modelo con mejor precisión del 90%

Prueba de los modelos:

La review de prueba es la siguiente:



review = A breezily unpredictable blend of teen romance and superhero action, Spider-Man: Far from Home stylishly sets the stage for the next era of the MCU.

Modelo N grams

```
#Entreamos el modelo modeloNgrams
modelo,vector = modeloNgrams(reviews_train_clean,reviews_test_clean)
review = "A breezily unpredictable blend of teen romance and superhero action, Spider-Man: Far from Ho
reviews_new = [review]
resultado = prediccion(modelo,reviews_new,vector)
print("N-grams")
print(resultado)
```

Final Exactitud: 0.89784 Resultado = [1] Es decir es una review positiva

Modelo Word Counts

```
#Entreamos el modelo word_Counts
modelo,vector = word_Counts(reviews_train_clean,reviews_test_clean)
review = "A breezily unpredictable blend of teen romance and superhero action, Spider-Man: Far from Ho
reviews_new = [review]
resultado = prediccion(modelo,reviews_new,vector)
print("Modelo Word Counts")
print(resultado)
```

Final Exactitud: 0.88164 Resultado = [1] Es decir es una review positiva

Modelo Naives Bayes

```
#Entreamos el modelo Naive_Bayes
modelo,vector = Naive_Bayes(reviews_train_clean,reviews_test_clean)
review = "A breezily unpredictable blend of teen romance and superhero action, Spider-Man: Far from Home
reviews_new = [review]
resultado = prediccion(modelo,reviews_new,vector)
print("Modelo Naive_Bayes: ")
print(resultado)
```

Final Exactitud: 0.88164 Resultado = [1] Es decir es una review positiva

Modelo TF IDF

```
#Entreamos el modelo TFIDF
modelo,vector = TFIDF(reviews_train_clean,reviews_test_clean)
review = "A breezily unpredictable blend of teen romance and superhero action, Spider-Man: Far from Ho
reviews_new = [review]
resultado = prediccion(modelo,reviews_new,vector)
print("Modelo TFIDF")
print(resultado)
```

Final Exactitud: 0.882 Resultado = [1] Es decir es una review positiva

Modelo SVM

```
#Entreamos el modelo svm
modelo,vector = svm(reviews_train_clean,reviews_test_clean)
review = "A breezily unpredictable blend of teen romance and superhero action, Spider-Man: Far from Ho
reviews_new = [review]
resultado = prediccion(modelo,reviews_new,vector)
print("Modelo SVM: ")
print(resultado)
```

Final Exactitud: 0.8974 Resultado = [1] Es decir es una review positiva

Modelo SVM N grams(1,3)

```
#Entreamos el modelo svm con g-grams(1,3)
modelo,vector = svmModificado(reviews_train_clean,reviews_test_clean)
review = "A breezily unpredictable blend of teen romance and superhero action, Spider-Man: Far from Ho
reviews_new = [review]
resultado = prediccion(modelo,reviews_new,vector)
print("Modelo SVM con Ngram (1,3)")
print(resultado)
```

Final Exactitud: 0.90064 Resultado = [1] Es decir es una review positiva

Salida de Resultados de los modelos:

```
Final Exactitud: 0.88164
Modelo Naive_Bayes:
[1]
Final Exactitud: 0.8974
Modelo SVM:
[1]
Final Exactitud: 0.90064
Modelo SVM con Ngram (1,3)
[1]
```

```
Final Exactitud: 0.89784
N-grams
[1]
Final Exactitud: 0.882
Modelo TFIDF
[1]
Final Exactitud: 0.88164
Modelo Word Counts
```

[1]

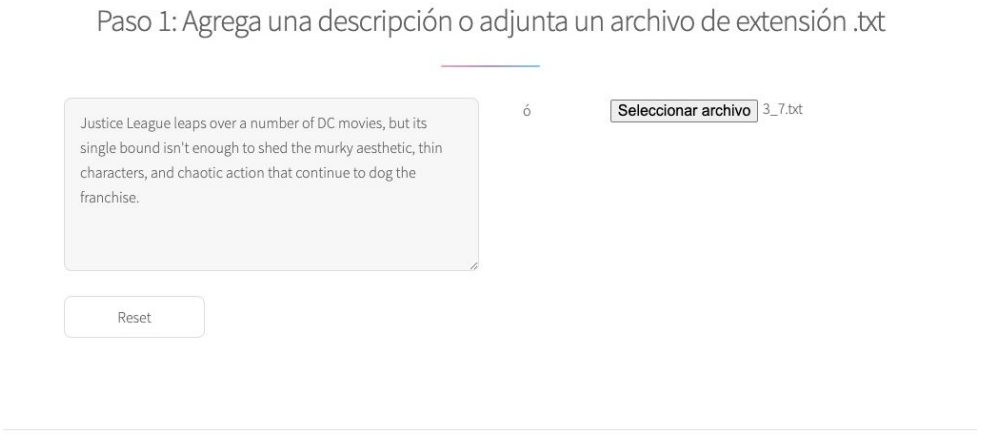
Palabras con mejor coeficiente para críticas para críticas positivas:

```
('excellent', 1.2625227657929066)
('perfect', 1.0781696042420938)
('great', 0.9268799522413366)
('wonderful', 0.8587926452009534)
('enjoyable', 0.8559600664270693)
('amazing', 0.8422757817306767)
('superb', 0.820634456531842)
('today', 0.73076433209877)
('incredible', 0.6827724184476873)
('brilliant', 0.6658585382129879)
('better than', 0.6653801702381954)
('fun', 0.655137888881649)
('loved', 0.6501006403309566)
('must see', 0.6482766124700786)
('enjoyed', 0.6474734948367261)
('rare', 0.6353274743193945)
('well worth', 0.6327066041891203)
('wonderfully', 0.6222348817401389)
('fantastic', 0.6094132804631667)
('refreshing', 0.6045131678693332)
('definitely worth', 0.5883705521469296)
('gem', 0.5857156382906169)
('favorite', 0.5819241452801973)
('the best', 0.5810205917559149)
('perfectly', 0.5793960354674211)
('very good', 0.575667630167155)
('bit', 0.5719146032950013)
('beautiful', 0.5681125415646566)
('liked', 0.5553780503800838)
('moving', 0.5552404152446396)
```

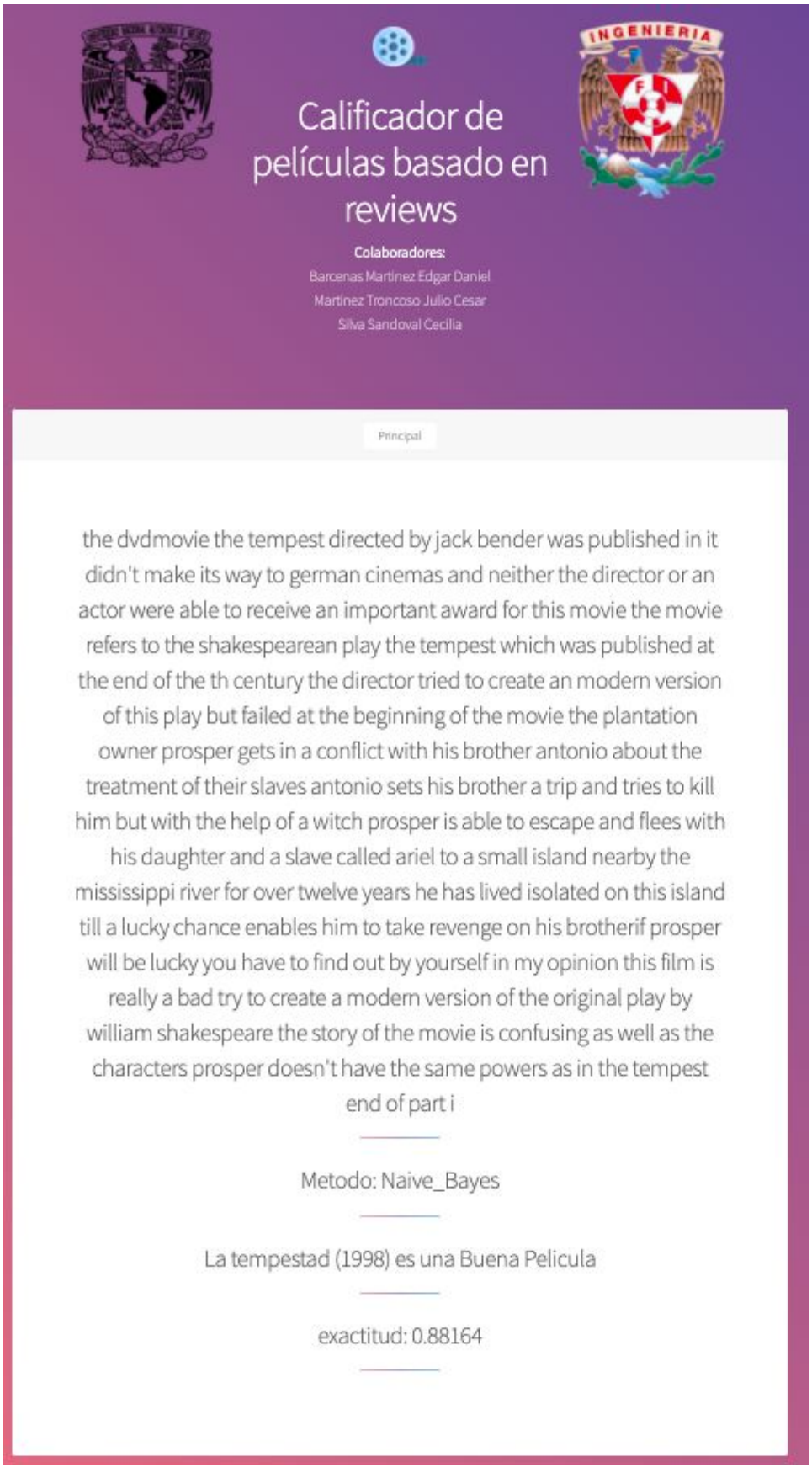
Palabras con menor coeficiente negativas:

```
('worst', -1.46343672244191)
('awful', -1.3949467484422602)
('boring', -1.2910225401547575)
('waste', -1.169186784516566)
('terrible', -1.0952222830725369)
('poor', -1.0630438066674512)
('disappointment', -1.0536162565844518)
('bad', -1.0527557945119834)
('poorly', -1.0442216011046561)
('dull', -1.025462485693274)
('the worst', -1.0147237050840479)
('disappointing', -0.9816825626990631)
('mess', -0.8575174139677783)
('worse', -0.8473637137965027)
('horrible', -0.8466780077630193)
('unfortunately', -0.822765403779086)
('stupid', -0.8110191427538358)
('not worth', -0.7974832173285571)
('lacks', -0.7925780718493837)
('lame', -0.7713653438835576)
('avoid', -0.7581512154206074)
('annoying', -0.7432521213597103)
('save', -0.7423181373530607)
('fails', -0.7312535507390289)
('laughable', -0.7270993875583401)
('oh', -0.7232828487700109)
('badly', -0.7157972731860618)
('weak', -0.7121664599787861)
('ridiculous', -0.7029660427810077)
('than this', -0.6992110735823298)
```

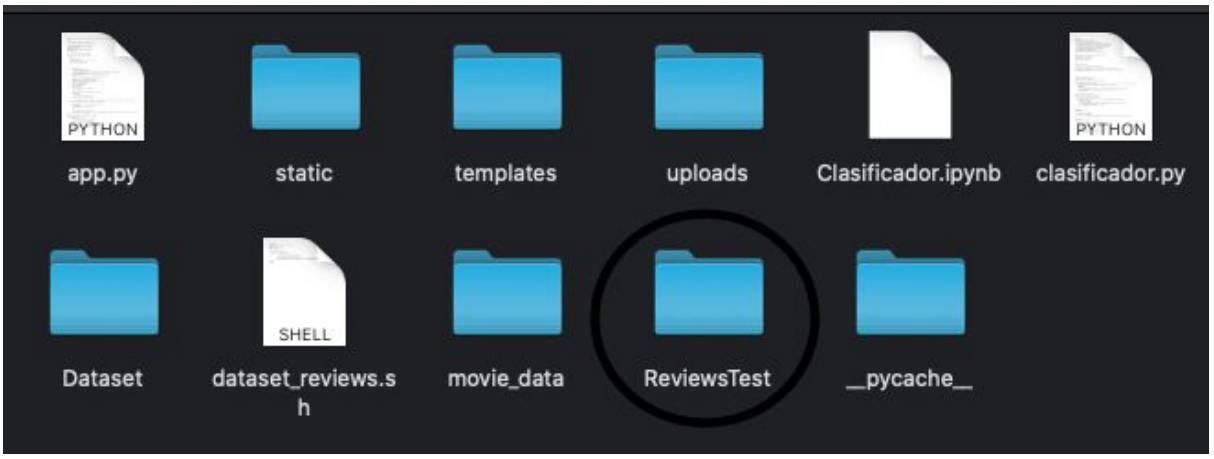
Presentación del proyecto con interfaz:



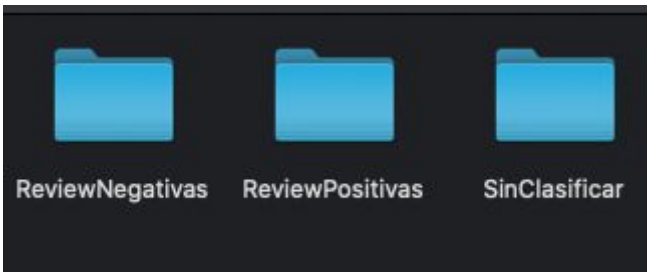
Resultado:


Calificador de películas basado en reviews
Colaboradores:
Barcenas Martinez Edgar Daniel
Martinez Troncoso Julio Cesar
Silva Sandoval Cecilia
Principal
the dvdmovie the tempest directed by jack bender was published in it didn't make its way to german cinemas and neither the director or an actor were able to receive an important award for this movie the movie refers to the shakespearean play the tempest which was published at the end of the th century the director tried to create an modern version of this play but failed at the beginning of the movie the plantation owner prosper gets in a conflict with his brother antonio about the treatment of their slaves antonio sets his brother a trip and tries to kill him but with the help of a witch prosper is able to escape and flees with his daughter and a slave called ariel to a small island nearby the mississippi river for over twelve years he has lived isolated on this island till a lucky chance enables him to take revenge on his brotherif prosper will be lucky you have to find out by yourself in my opinion this film is really a bad try to create a modern version of the original play by william shakespeare the story of the movie is confusing as well as the characters prosper doesn't have the same powers as in the tempest end of part i
Metodo: Naive_Bayes
La tempestad (1998) es una Buena Pelicula
exactitud: 0.88164

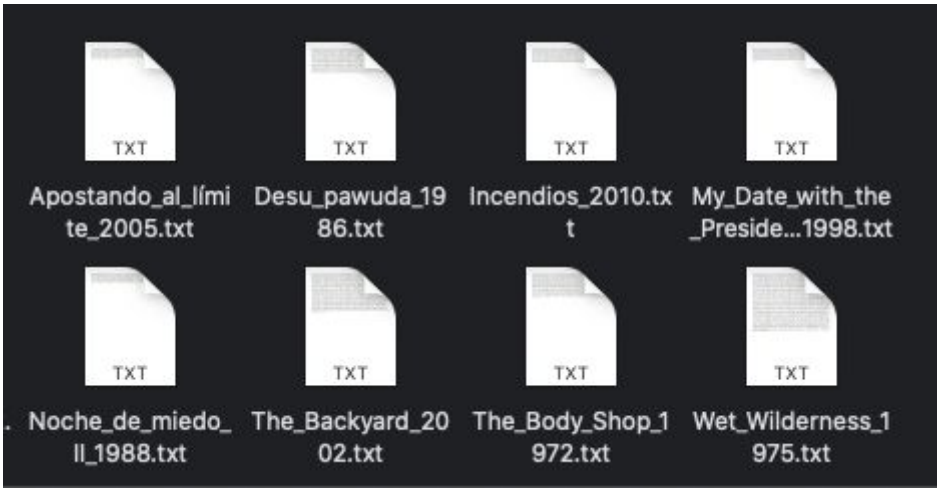
Probar buscador de reviews:



Selecciona carpeta de reviews



Escoge una review de pelicula:



CONCLUSIÓN

En este proyecto se cumplio el objetivo que era clasificar peliculas de acuerdo a su crítica, con el conjunto de datos pudimos generar modelos bastantes buenos debido a que el dataset contiene 50,000 reviews de películas, con estas pudimos entrenar nuestros modelos y encontrar los mejores resultados. Cada modelo utilizado tiene características muy particulares y cada uno tiene ventajas sobre otro pero para nuestro conjunto de datos encontramos que el mejor modelo es el de SVM con n-grams (1,3) lo que nos lleva a tener una precisión del 90%. En este modelo hicimos una modificación en las palabras funcionales las cuales reducimos para tener una mejor precisión.

Con el método de n-grams sabemos que no hay límite en el tamaño de n y podemos poner un valor más amplio pero descubrimos que si aumentamos n no nos dará un mejor rendimiento, también el tamaño de su matriz crece exponencialmente a medida que aumenta n y si el dataset es muy grande entonces el modelo puede tardar en formarse. Lo ideal para nuestro modelo de datos es tener de n-gram (1,3)
Con la combinación de 2 o más métodos podemos lograr una mejor precisión. También tenemos que la regresión logística es un buen modelo de referencia porque podemos interpretar los datos y los modelos lineales tiene a funcionar bien en conjuntos de datos dispersos y su velocidad de aprendizaje es buena.Con la frecuencia de término – frecuencia inversa de documento comprobamos que es una medida numérica que expresa cuán relevante es una palabra para un texto en un dataset. Comprobamos que es una medida que se utiliza a menudo como un factor de ponderación en la recuperación de información y sirve para clasificación.Con word count comprobamos que aunque podemos tener las palabras con mayor coeficiente de positividad y se repiten no mejoran mucho el modelo de predicción. Con el clasificador de Naive Bayes comprobamos que es simple y funciona clasificando en función de las probabilidades de las palabras. En un futuro se pretende poder realizar clasificaciones más enfocadas y encontrar herramientas que nos permitan llegar a una mejor precisión y obtener resultados más rápido. En conclusión el mejor modelo es el de SVM con N Grams (1,3) con un menor número de palabras funcionales con una precisión del 90%.

REFERENCIAS

Dataset:
author= {Maas, Andrew L. and Daly, Raymond E. and Pham, Peter T. and Huang, Dan and Ng, Andrew Y. and Potts, Christopher},title= {Learning Word Vectors for Sentiment Analysis},book title = {Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies},month = {June}, year ={2011},
address = {Portland, Oregon, USA},publisher = {Association for Computational Linguistics},
pages = {142--150},url = {http://www.aclweb.org/anthology/P11-1015}

Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng,. (2011, 3 mayo). Learning Word Vectors for Sentiment Analysis. Recuperado 1 de junio de 2020, de https://ai.stanford.edu/~amaas/papers/wvSent_acl2011.pdf

Abhishek Ghose. (2020, 15 agosto). Support Vector Machine (SVM) Tutorial. Recuperado 1 de junio de 2020, de https://blog.statsbot.co/support-vector-machines-tutorial-c1618e635e93

Aiyappan, J. (2019, 8 septiembre). Naive Bayes Classifier for Text Classification - Analytics Vidhya. Recuperado 2 de junio de 2020, de https://medium.com/analytics-vidhya/naive-bayes-classifier-for-text-classification-556fabaf252b#:%7E:text=The%20Naive%20Bayes%20classifier%20is,time%20and%20less%20training%20data

Karen L. Vazquez , Mireya Tovar , Jose A. Reyes-Ortiz , Darnes Vilarino. (2016, 30 septiembre). An´alisis comparativo entre diferentes entornos de aprendizaje automático para el análisis de sentimientos. Recuperado 3 de junio de 2020, de https://www.rcs.cic.ipn.mx/2016_124/Analisis%20comparativo%20entre%20diferentes%20entornos%20de%20aprendizaje%20autmatico.pdf

Build a Sentiment Analysis app with Movie Reviews – Python For Engineers. (s. f.). Recuperado 4 de junio de 2020, de https://www.pythonforengineers.com/build-a-sentiment-analysis-app-with-movie-reviews/

UNAM. (s. f.). PLN con Python. Recuperado 3 de junio de 2020, de http://www.corpus.unam.mx/cursopln/

Badreesh, S. (2018, 24 noviembre). Natural Language Processing(NLP) for Machine Learning. Recuperado 4 de junio de 2020, de https://towardsdatascience.com/natural-language-processing-nlp-for-machine-learning-d44498845d5b

Goyal, S. (2020, 6 abril). MachineX: Sentiment analysis with NLTK and Machine Learning. Recuperado 5 de junio de 2020, de https://blog.knoldus.com/machinex-sentiment-analysis-with-nltk-and-machine-learning/

Mayo, M. (2020, 12 marzo). Preprocesamiento de datos de texto: un tutorial en Python. Recuperado 6 de junio de 2020, de https://medium.com/datos-y-ciencia/preprocesamiento-de-datos-de-texto-un-tutorial-en-python-5db5620f1767

Monsters, D. (2020, 10 abril). Text Preprocessing in Python: Steps, Tools, and Examples. Recuperado 7 de junio de 2020, de https://medium.com/@datamonsters/text-preprocessing-in-python-steps-tools-and-examples-bf025f872908

https:\u002F\u002Feditorial.rottentomatoes.com\u002Fwp-json\u002Fusers\u002F7. (2020, 12 junio). Spider-Man: Far From Home (2019). Recuperado 11 de junio de 2020, de https://www.rottentomatoes.com/m/spider_man_far_from_home

html5up. (s. f.). makes spiffy HTML5 site templates that are. Recuperado 8 de junio de 2020, de https://html5up.net/