FIB

Universitat Politècnica de Catalunya

Facultat d'Informàtica de Barcelona

Detección de Emoción en Críticas de Películas

Grado de Inteliencia Artificial

Preprocesamiento del Lenguaje Humano

Autores:

Eduard Barnadas

Pablo Barrenechea

9 de abril de 2024

Índice

1. Abstract

En este informe, exploramos dos enfoques diferentes para la detección de opiniones: uno supervisado y otro no supervisado. Dentro de estos veremos las diferentes técnicas utilizadas para poder determinar si las críticas de películas de *nltk.corpus MovieReviews* son positivas o negativas. Una vez formados los diferentes modelos los compararemos para ver sus diferencias, extraer conclusiones y ver cual es el mejor modelo dentro de cada campo, es decir, *supervised* o *unsupervised*

2. Modelos Supervisados

Para esta primera parte de la práctica, probaremos distintos modelos supervisados con el objetivo de que clasifiquen las distintas críticas del corpus de forma binaria, clasificándolas entre positivas (**pos**) o negativas (**neg**). Para ello, hemos escogido 4 modelos clasificadores que, o bien suelen, o bien pueden utilizarse para realizar tareas de clasificación binaria. Por un lado, tendremos el modelo *Multinomial Naive Bayes*, que se nos recomendó en la propia asignatura de PLH como modelo útil para la clasificación de los sentimientos expresados por un texto. Además de este, hemos escogido la regresión logística, puesto que es un clasificador binario que hemos estudiado en varias asignaturas. Finalmente, también hemos decidido experimentar con el *Support Vector Machines* (SVM) y el *Random Forest*. Puesto que son clasificadores que también hemos estudiado en otras asignaturas centradas en el Aprendizaje Automático.

2.1. Preprocesado

Antes de comenzar con el entrenamiento, ajuste y rendimiento de los distintos modelos, se debe realizar un buen preprocesado del corpus, con el objetivo de facilitar a los modelos la tarea de clasificación. El preprocesado de dicho corpus podría dividirse en dos partes. La primera, a la que llamaremos la "esencial", será la encargada de transformar el texto en algo más homogéneo y sencillo para poder separar por palabras de manera sencilla; y la "adicional" constará de cambios como la eliminación de *stopwords*, teniendo esta el objetivo de reducir el ruido y ayudar al modelo a poder predecir y generalizar de mejor manera.

En lo que al preprocesado esencial respecta, primero se han sustituido todas las letras mayúsculas por minúsculas y se han eliminado cualquier carácter no considerado *alpha* (todos los caracteres que no sean las letras de la "a" a la "z", puesto que al nivel al que trabajamos, sería demasiado complicado extraer cualquier tipo de subjetividad de un número o un signo de puntuación con este tipo de modelos. Habría sido una opción entrenar un modelo a nivel de oraciones, y después ejecutar dicho modelo para cada oración, sin embargo, al no tener la subjetividad de cada oración en el corpus, no habríamos sido capaces de entrenar ningún modelo supervisado de esta forma. Finalmente, uniremos cada palabra por un espacio. Así pues, nos quedarán todas las palabras del corpus unidas por un espacio, sin caracteres que no sean letras.

Ya pasado el corpus por este primer preprocesado, el preprocesado "adicional", se basa en los siguientes pasos. Primero, se lematiza cada una de las palabras mediante el lematizador de *NLTK*, con el objetivo de que palabras con la misma raíz sean transformadas en dicha raíz; facilitando así el reconocimiento de palabras con la misma raíz, que en general, tienen el mismo significado subjetivo. De esta forma, también lograremos reducir la dimensionalidad, y por tanto, la complejidad de los datos con los que se entrenarán los modelos, haciendo que se reduzca el ruido. Finalmente, hemos decidido también eliminar las *stopwords* del corpus, pues al nivel de nuestra práctica (no realizaremos análisis morfosintáctico, pues aumentaría extremadamente la complejidad del problema) estas palabras no aportan sentido de positividad o negatividad.

Habiendo preprocesado de esta forma el corpus, podemos pasar a analizar los modelos.

2.2. Multinomial Naive Bayes

El modelo de *Naive Bayes Multinomial*, supone una gran ventaja en lo que a coste computacional respecta, debido a lo fácil que es calcular las distintas probabilidades dados los datos como la matriz de vectorizada de las palabras del corpus. Este, presenta un hiper-parámetro α, el encargado de la regularización del modelo. Para obtener una α que se adapte al corpus de entrenamiento, hemos decidido utilizar una búsqueda *greedy* utilizando el *k-fold cross validation* que intente maximizar el accuracy del modelo (mediante la clase **GridSearchCV** del módulo **Sklearn** de python). Este será el método que utilizaremos para escoger los hiper-parámetros del resto de modelos también. En este caso, hemos realizado la búsqueda para los siguientes posibles valores de *alpha*: 0.1, 0.5, 1.0, 2, 3, 5, 10, 20, 50 y 100.

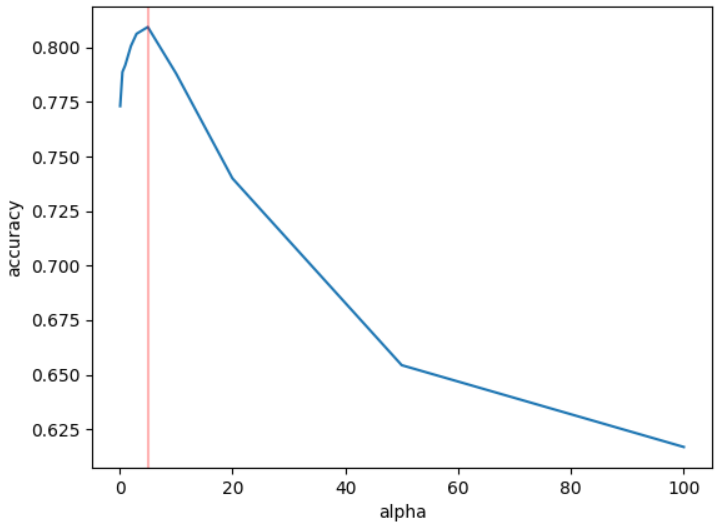

Figura 1: *Accuracy* del modelo según el hiperparámetro α según la *greedy search* con *cross-validation*

Tal y como se puede ver en la figura 1, el valor de α que mejor se ajusta según la *cross validation* es $alpha = 5$. Así pues, habiendo elegido el único hiper-parámetro requerido por este modelo, hemos entrenado el modelo con el corpus de entrenamiento completo, y una vez entrenado, hemos predicho con él la división del test del corpus, obteniendo así la matriz de confusión de la figura 2.

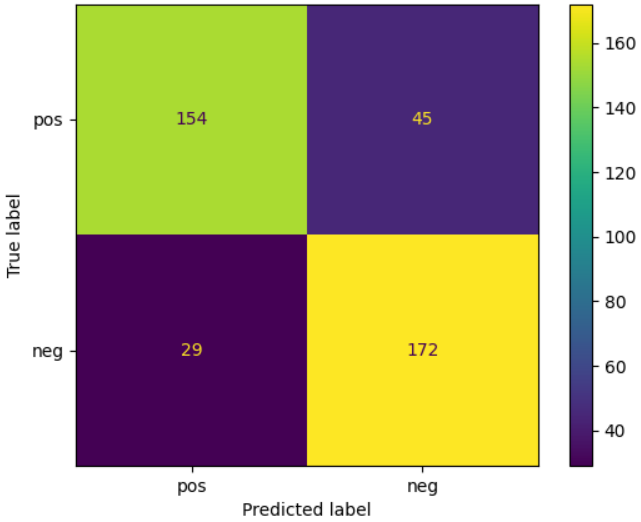

Figura 2: Matriz de confusión con los resultados del modelo multinomial al predecir el corpus de test

	Métricas	
	Precision	Recall
neg	0.79	0.86
pos	0.84	0.77
Accuracy: 0.81		

Cuadro 1: Métricas de evaluación

Como se puede observar en la matriz, el modelo tiende a clasificar mejor las críticas negativas que las positivas (teniendo estas mejor métrica de **precision** en la tabla con las métricas resultantes (tabla 1)), teniendo más falsos negativos que falsos positivos, y consecuentemente, acertando más con las críticas negativas. En general, con un *accuracy* de 0.81, podemos decir que es un modelo relativamente bueno, sobre todo sabiendo que tarda alrededor de 0.5 segundos en ejecutar los 10 entrenamientos con distintas α, estando cada entrenamiento compuesto por 5 entrenamientos con 4/5 de la partición de entrenamiento (debido a que se hace *cross validation* con 5 *folds*), podemos decir que es un algoritmo muy eficiente en relación *accuracy*/coste-computacional.

2.3. Regresión Logística

Al tener una variable respuesta binaria, pensamos que podríamos usar un modelo de *Logistic Regression* para predecir las opiniones de las películas una vez vectorizadas. Igual que con el resto de modelos, usamos *GridSearchCVC* para encontrar los mejores parámetros para este. En este caso, fijamos el máximo de iteraciones a 1000, y buscamos el mejor hiperparámetro C entre 0.1, 1, 2, 5, 10, 20, 50, 75y 100; y probamos diferentes solvers que nos ofercía el modelo de *Sklearn*: *newton-cg*, *lbfgs* y *liblinear*. Así, siguiendo el mismo método de búsqueda que en el modelo explicado en la sección anterior (sección 2.2), obtenemos la gráfica de la figura 3.

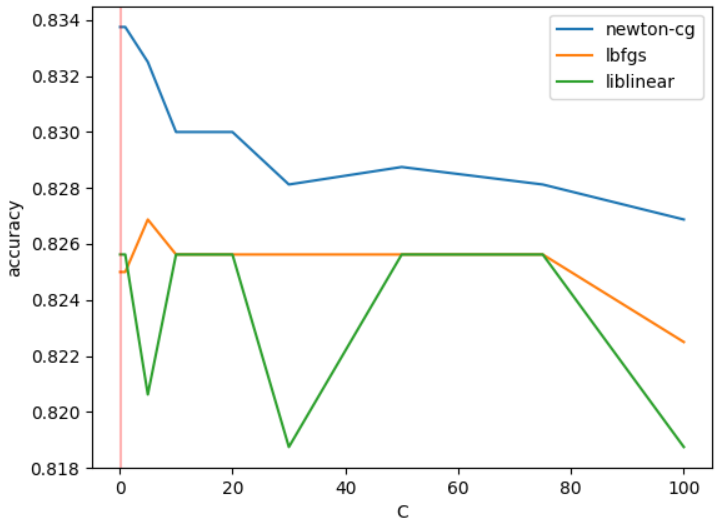

Figura 3: *Accuracy* de la regresión logística en función de el hiperparámetro C y los distintos *solvers*

Como podemos apreciar en el gráfico (figura 3), el *solver* de optimización de Newton da resultados considerablemente mejores que los otros dos solvers. *Liblinear* (según Scikitlearn) tiene una mejor actuación con bases de datos pequeñas, siendo esta seguramente la causa de sus picos de peor rendimiento en este caso, aunque una *accuracy* de alrededor de 0.82 está bastante bien. Por otro lado, no sabemos la razón exacta para que *lbfgs* sea menos eficaz que el método newtoniano. Sin embargo, queda claro que en caso de querer exprimir el máximo de este modelo, debemos escoger el solver *newton-cg*.

En lo que al hiperparámetro C respecta, se puede observar una clara tendencia a peor con el aumento de dicha variable. Esto se debe, seguramente, al hecho de que una C pequeña representa más regularización, hecho que obliga al modelo a mantener valores paramétricos pequeños, reduciendo así su complejidad y tendencia a realizar overfitting. Así pues, tiene sentido que un valor de C pequeño obtenga buenos resultados a la hora de mirar el *accuracy* medio con lo que serían las 5 particiones de validación.

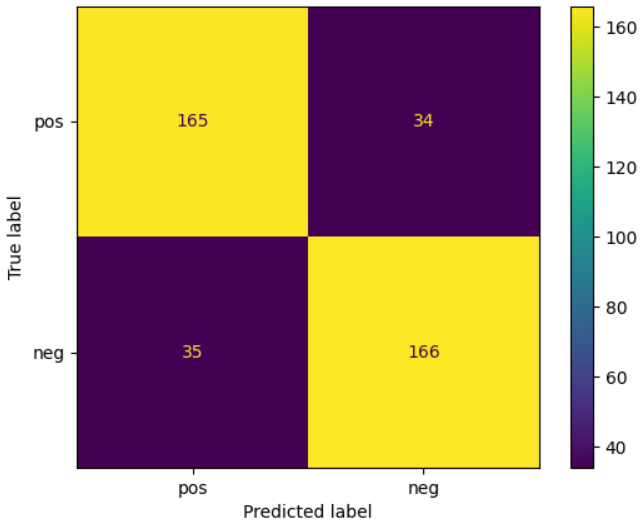

Figura 4: Matriz de confusión con los resultados de la Regresión Logística entrenada con la partición de entrenamiento del corpus al predecir la partición de test

	Métricas	
	Precision	Recall
neg	0.83	0.83
pos	0.82	0.83
Accuracy: 0.83		

Cuadro 2: Métricas de evaluación

Una vez escogidos la C = 0 y el solver *newton-cg*, hemos obtenido la matriz de confusión de la figura 4, junto con los datos de la tabla 2. Tal y como se puede observar, la matriz de confusión se encuentra un poco más equilibrada que aquella del modelo multinomial (figura 2). Esto hace que sus valores de *precision* y *recall* sean muy similares entre **neg** y **pos**, siendo todos 0.83, a excepción de la *precision* de **pos**, que es 0.82. En general, para un modelo que tarda 1:30 minutos en ejecutar

sus 135 entrenamientos, no está nada mal. Si tenemos algo más de tiempo, podría preferirse frente al *multinomial*, debido principalmente a su equilibrio en cuanto a las predicciones (haciendo los mismos falsos positivos que falsos negativos) y a su *accuracy* algo más elevado.

2.4. Support Vector Machine

Elegimos probar con un modelo de *SVM* ya que nos interesaba su capacidad para trabajar con variables de alta dimensionalidad y la opción de poder explorar relaciones no lineales entre las variables, además de los diferentes *kernels* que este nos ofrece para poder interpretar nuestros datos de la forma más precisa, aplicando distintos aumentos de dimensionalidad, tal que pueda así dividir en datos linealmente no separables. En este caso, para realizar el *grid search* elegimos como hiperparámetros el tipo de *kernel*, que podrá ser *linear o RBF*, y el valor de *C*, que varía entre 0.1, 1, 5, 10 y 20. Así pues, ejecutamos el *GridSearchCV* de *Sklearn* tal y como se ha realizado con los anteriores 2 modelos, obteniendo los resultados tal y como se ven en la figura 5.

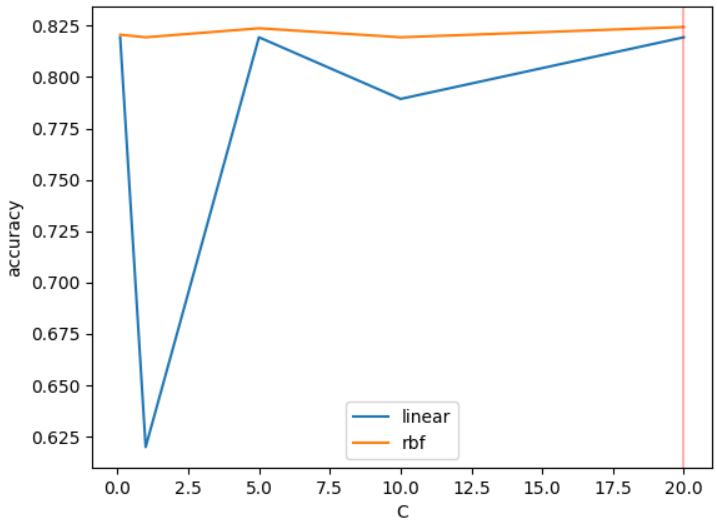

Figura 5: *Accuracy* de la regresión logística en función de los dos tipos de kernel y el hiperparámetro C

En la figura 5 observamos una clara y destacable mejora de *accuracy* entre los dos tipos de *kernel*s

implementados. Vemos como el *rbf* tiene un rendimiento prácticamente igual, sin variar según C. Su gran actuación se deba, seguramente, a la capacidad del kernel de adaptar los datos a las condiciones de los cortes lineales que realiza el *SVM*. Por otro lado, el kernel *lineal* ha tenido más problemas a la hora de adaptarse a unos datos que, probablemente, no sea tan adaptable como el kernel que aumenta la dimensionalidad de los datos. Sin embargo, si parece que con una algunas regularizaciones de C sí consigue generalizar de alguna manera.

En lo que a la C respecta, poco se puede comentar de este hiperparámetro de regularización, puesto que los valores escogidos no parece afectar al *kernel* no lineal, y en el *kernel* lineal parece afectar de forma irregular, sin patrones aparentes. Así pues, debido a que su accuracy es más alta, y sobre todo, a su buen desempeño, hemos decidido quedarnos con el kernel *rbf* y con C 20, a pesar de que este último no parezca influir tanto.

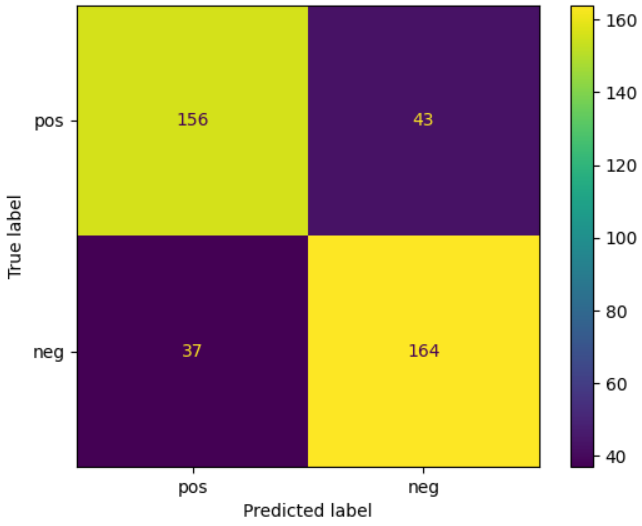

Figura 6: Matriz de confusión con los resultados de el *SVM* entrenado con la partición de entrenamiento del corpus al predecir la partición de test

Finalmente, una vez entrenado el SVM con los hiperparámetros seleccionados, hemos podido obtener la matriz de confusión de la figura 6 y los resultados de la tabla 3. Como se puede observar, su accuracy es un poco más bajo que aquel de la regresión logística y el del modelo multinomial, haciendo de este un modelo ligeramente peor. En lo que al *precision* y *recall* respecta, observamos

	Métricas	
	Precision	Recall
neg	0.79	0.82
pos	0.81	0.78
Accuracy: 0.80		

Cuadro 3: Métricas de evaluación

una tendencia a clasificar más falsos negativos, haciendo que tenga una *precision* más baja a la hora de clasificar críticas negativas. Sin embargo, tiene unas métricas de *recall* y *precision* mejores que el modelo multinomial, haciéndolo los resultados algo más equilibrados. Finalmente, comentar que este es el modelo que más tarda en ejecutarse, haciendo que, debido a su tiempo y rendimiento, no sea el mejor modelo, quedando por debajo de la "performance" de la regresión logística.

2.5. Random Forest

También decidimos hacer pruebas entrenando un modelo de *Random Forest* de la libreria *Sklearn*. Creíamos que este modelo podría funcionar bien, ya que, a pesar del hecho de que un *decision tree* por si sólo no es el algoritmo óptimo para manejar una base de datos con tantas dimensiones como en este caso una reseña vectorizada, sus propiedades para separar entre dos clases nos parecían muy interesantes para alcanzar nuestro objetivo. La idea era que utilizando las propiedades de diversos árboles en un *random forest*, estos lograran generalizar de alguna manera tantas dimensiones, obteniendo buenos resultados.

En el **GridSearch** que realizamos escogimos los parámetros de *max_depth* (máxima profundidad de ramas que se le permite tener a un árbol de decisión) y *n_estimators* (número de árboles), dando como resultado el gráfico de a continuación 7. Como se puede observar, hemos decidido probar con 10, 25, 50 y 100 árboles de decisión, y con profundidad **None**, 10, 20, 30, 40, 50, 10, 20, 30, 40 y 50, siendo **None** profundidad ilimitada (hasta que el árbol separe perfectamente todas las clases).

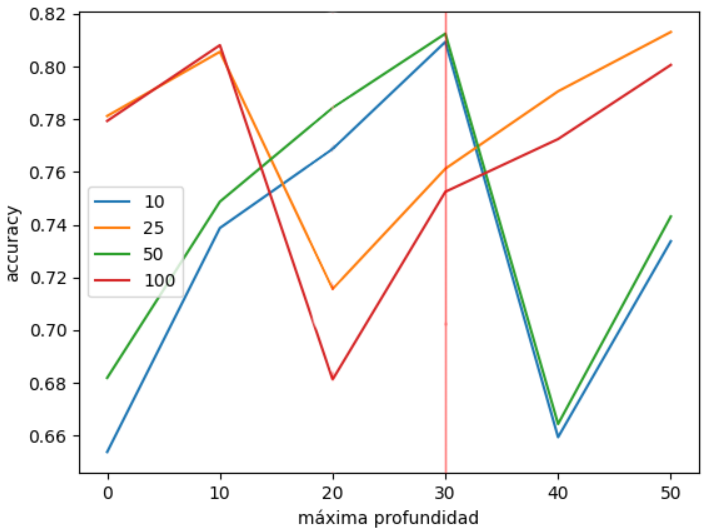

Figura 7: Comparación de *accuracy* entre los diferentes parámetros de *GridSearch* para el modelo *Random Forest*

Como resultado podemos ver que los mejores parámetros que hemos encontrado son *n_estimators*=**50** y *max_depth*=**30**. Usando estos parámetros para entrenar nuestro modelo final, obtenemos un *accuracy* con la partición de *train* de **0.83**, tal y como se puede ver en la tabla con las métricas resultantes 8.

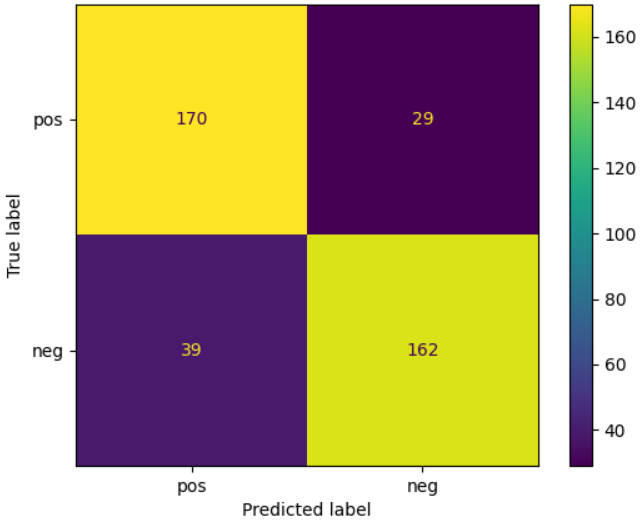

Figura 8: Matriz de confusión con los resultados de el *Random Forest* entrenado con la partición de entrenamiento del corpus al predecir la partición de test

	Métricas	
	Precision	Recall
neg	0.79	0.82
pos	0.81	0.78
Accuracy: 0.80		

Cuadro 4: Métricas de evaluación

Tal y como podemos ver, la matriz de confusión resultante (figura 8), el *random forest* estima bien los positivos, teniendo un bajo número de falsos negativos, pero se queda un poco ambigua a la hora de predecir los negativos, teniendo estos una precisión del 0.79, frente al 0.81 de los positivos. En definitiva, para el tiempo que tarda en entrenarse con el 80 % de los datos (apróximadamente el doble que la regresión logística), debido a su pequeño "bias" hacia lo positivo y, sobre todo, su menor métrica de *accuracy*, tampoco es un modelo mejor.

3. Modelos No Supervisados

3.1. Partición Train-Test

Para nuestro modelo no supervisado, hemos escogido una partición de *train* y *test* igual a la que hemos usado para nuestros modelos supervisados, de esta forma podremos comparar los resultados para ver cuál de los modelos es mejor. Una vez hecha la partición podemos pasar a explicar la función que se encarga de clasificar las críticas.

3.2. Definición de nuestro modelo no supervisado

En este caso, nuestro modelo se trata de una función llamada *movie_review_categorizer*, esta toma como parámetros *review*, que se trata de la crítica que estamos intentando clasisicar, *func*, que será la función que aplicaremos a la suma de puntuaciones, *mode*, que puede ser un número entero entre 1 y 4, que corresponde a los modos definidos en el enunciado de la práctica, es decir, que la función solo tenga en cuenta para clasificar:

- Adjetivos
- Adjetivos y nombres
- Adjetivos, nombres y adverbios
- Adjetivos, nombres, adverbios y verbos

Como parámetros también tenemso *max_sum*, que elige si solo sumar la puntuación más alta de una palabra, es decir si la puntuación positiva supera a la negativa, solo sumar el valor de esa, y viceversa. Finalmente tenemos un hiperparámetro llamado *theta* (θ) que representa una ponderación sobre nuestros resultados para poder ajustar nuestro modelo al mínimo de fallos, este puede ser cualquier número real.

3.3. Preprocesado

Es en las primeras líneas de la función donde encontramos el preprocesado del texto de nuestras críticas, este empieza por la eliminación de cualquier carácter que no forme parte del alfabeto usando el método *isalpha()*, de esta forma conseguimos eliminar todos los puntos, comas y otros caràcteres que no aportan información sobre la positividad o negatividad de la crítica. Después de esto, pasamos todos los carácteres restantes a minúscula y lematizamos las palabras con *nltk.WordNetLemmatizer*, de esta forma podremos obtener la misma puntuación para palabras que significan lo mismo, por ejemplo *hate* y *hatred*, que en este caso serían las dos igual de negativas. Una vez realizado este preprocesado podemos proceder a la puntuación de cada palabra para clasificar a la crítica.

3.4. Puntuación y clasificación

Para clasificar nuestra crítica pasamos a cada palabra por un proceso de puntuación usando *sentiwordnet* para obtener la puntuación positiva y negativa de cada synset de nuestro texto. Para obtener estos, lo primero que hacemos es separar cada frase de nuestro texto en palabras usando *nltk.tokenizer*, y, una vez tokenizada la frase, usamos *lesk* para poder *desambiguar* el synset de cada una de las palabras teniendo como contexto el resto de palabras de la frase. És cuándo tenemos el synset *desambiguado* de cada palabra que podemos proceder al proceso de puntuación, este consiste en sumar la puntuació positiva y negativa en dos variables que irán recolectando la suma de cada palabra de nuestro texto. És aquí donde entran en juego nuestros parámetros, ya que en el caso de que *sum_max* sea *True*, solo sumaríamos aquella puntuación más elevada, y antes de hacer-lo, este número tiene que pasar por *func* que será la función que nosotros hayamos escogido para modificar el valor de nuestros resultados. Este proceso se repite para cada palabra de cada frase hasta haber acabado todas las del texto, es entonces cuando nuestros valores finales de **pos** y **neg** son elevados a nuestro hiperparámetro *theta* (θ), como este por defecto es 1, los resultados no serán modificados, pero nos puede ser útil si detectamos un sesgo hacia cierto valor en nuestras predicciones. Por último, la función compara los valores de **pos** y **neg** y retorna la categoria del más grande de los dos, ya que esa será nuestra predicción para la crítica escogida.

3.5. GridSearch y búsqueda de los mejores parámetros

Ahora que ya tenemos una función que clasifica nuestras críticas de películas, tenemos que ajustar sus parámetros para tener el modelo que mejor prediga la opinión de cada una. Para conseguir esto usaremos *GridSearch* como en los apartados anteriores, aún que en este caso, al ser una función creada por nosotros, tendremos que crear una clase que tenga *fit* y *predict*, como estamos trabajando con un modelo no supervisado, nos hemos encargado de que el método *fit* simplemente retorne la misma función que hemos creado anteriormente, de forma que siga siendo el mismo modelo no supervisado. Somos conscientes de que no se suelen usar métodos como *GridSearch* para el aprendizaje no supervisado, pero hemos llegado a la conslusión que, para poder saber que valores eran los más indicados para nuestros parámetros, este método nos facilitaba la búsqueda, a demás de darnos herramientas para poder justificar esta elección de valores, es por eso que hemos decidido usarlo.

Una vez realizada9, la búsqueda nos retorna que el mejor valor para *theta* es $\theta = 0{,}5$, el mejor modo de aceptación de palabras es el 1, que es el que solo toma en cuenta los adjetivos del texto, la mejor función para la suma de valores es el logarítmo natural ln y com *max_sum* tomando valor *False*. En este caso podemos ver la matrix que expresa la Accuracy teniendo en cuenta los valores de *func* [log,x,tanh] (tenemos que añadir la suma de $\epsilon = 10^{-10}$ para evitar error en la función) y los diferentes modos de aceptación de *synsets*.

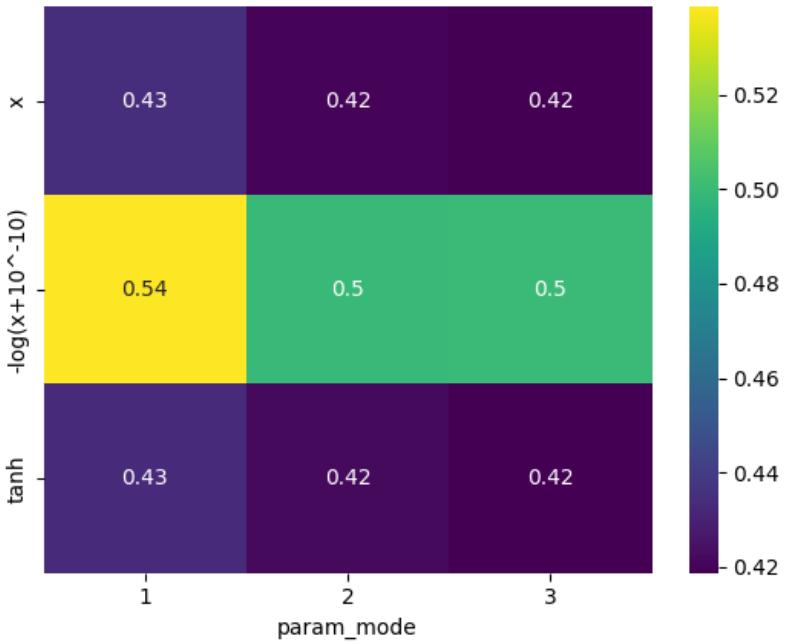

Figura 9: Comparación de Accuracy entre los diferentes parámetros de GridSearch

3.6. Resultados y conclusiones

Después de haber ajustado los parámetros de nuestro modelo, la matriz de confusión que obtenemos sobre nuestros datos de *test*10 presenta una gran cantidad de valores que han sido clasificados de forma errónea, estos se encuentran repartidos de forma homogénea entre falsos positivos y falsos negativos, al igual que los valores de la diagonal, queriendo decir que nuestro modelo no está sesgado hacia la positividad ni negatividad.

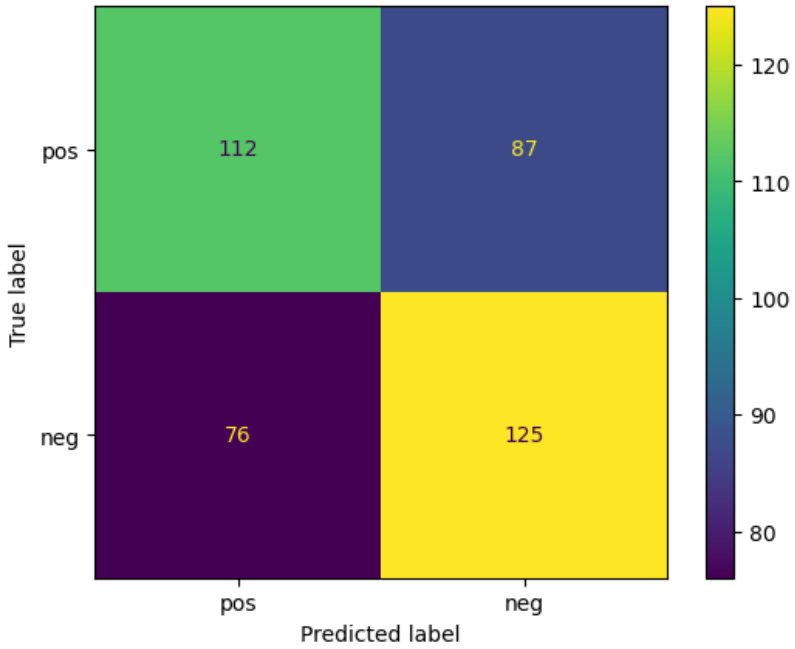

Figura 10: Matriz de confusión mostrando los resultados del modelo no supervisado

Esto se puede ver mejor representado si vemos el informe de clasificación de nuestro modelo, ya que presenta valores muy parecidos o incluso iguales para muchos de los parámetros de este. 5

	Métricas	
	Precision	Recall
neg	0.59	0.62
pos	0.60	0.56
Accuracy: 0.60		

Cuadro 5: Métricas de evaluación

3.6.1. Conclusiones

Por un lado, en lo que a los modelos supervisados se refiere, está claro que el mejor modelo a escoger, es la regresión logística, debido sobre todo, a su mejor *accuracy*, poco *bias* hacia ninguna de las clases y poco reducido tiempo de entrenamiento. Sin embargo, cabe destacar que los 4 modelos han tenido un buen rendimiento, sin bajar ninguno de ellos del 80 % de *accuracy* en los resultados finales.

Como conclusiones para nuestro modelo no supervisado, hemos acabado escogiendo el modelo que fuera más imparcial con nuestros datos, es decir, el menos sesgado, aún que hayamos visto otras combinaciones de parámetros que resultaban una *accuracy* más alta, hemos considerado que es mejor tener un modelo que sea menos preciso pero que muestre una base sólida, que no otro que de un mejor resultado numérico pero no sea realmente aplicable.

A demás de esto, creemos que un factor decisivo por el que hemos obtenido unas métricas tan bajas es el método que hemos usado para *desambiguar* las palabras y obtener *synsets*, el algoritmo de *lesk*, a demás de ser muy poco eficiente en cuanto a recursos se refiere, no da muy buenos resultados *desambiguando* palabras, por eso creemos que muchas palabras que podrían haber sido consideradas positivas o negativas se han omitido por ser representadas por el *synset* incorrecto.

En resumen, creemos que el modelo no supervisado podría haber competido en cuanto a métricas se refiere con el no supervisado, pero la falta de recursos para poder aplicar nuestros métodos ha hecho que este rinda por debajo de su máximo.