# Detecció d'opinions

Pau Hidalgo Pujol i Cai Selvas Sala

9 d'abril de 2024



Universitat Politècnica de Catalunya
Grau en Intel·ligència Artificial
Processament del Llenguatge Humà

**Resum**

En aquest informe s'explica el desenvolupament que s'ha fet de la pràctica sobre detecció d'opinions (*sentiment analysis*) corresponent a l'assignatura de Processament del Llenguatge Humà (PLH) del Grau en Intel·ligència Artificial de la Universitat Politècnica de Catalunya (UPC).

Al llarg d'aquest document s'expliquen els passos que s'han seguit, l'estructura del codi implementat, la justificació de les decisions preses i conclusions extretes.

# Índex

# 1 Introducció

Una de les aplicacions clàssiques del processament del llenguatge humà és l'anàlisi de sentiments (en anglès *sentiment analysis*). Hi ha diverses maneres d'afrontar aquest problema i diversos objectius. En aquest treball, ens centrarem en l'anàlisi de la polaritat d'un text, concretament de ressenyes de pel·lícules.

Es presentaran dues implementacions per dur a terme aquesta tasca: una basada en un codificador *bag of words* i models d'aprenentatge supervisat, i una altra basada en *SentiWordnet* i aprenentatge no supervisat. A continuació, s'analitzaran els seus resultats i se n'extrauran conclusions.

# 2 Estructura general de l'entrega

L'entrega conté diversos fitxers, tant de python com JSON. Els dos principals són els dos notebooks: *supervised.ipynb* i *unsupervised.ipynb*.

Aquests, en el seu interior, contenen la part principal del codi necessària per realitzar els dos apartats, així com una petita explicació. Estan dividits per apartats. En el cas del *supervised*, es divideix en una primera part d'inicialització, que llegeix fitxers i defineix funcions, seguida d'una que serveix per provar diversos models. A continuació, amb els models escollits realitza el *Cross-Validation* per finalment acabar amb l'apartat de Test, on s'executa el millor i es creen diverses gràfiques.

En el fitxer de *unsupervised*, el primer apartat s'encarrega d'inicialitzar les funcions necessàries. Els tres següents (Lesk, UKB i frequencies) corresponen als tres mètodes de desambiguació de sentits que hem usat (es comentaran més endavant). Com que ja s'han executat en el passat, no cal executar-los i es pot utilitzar directament els fitxers *lesk_test_synsets.json*, per exemple. L'apartat següent realitza una petita validació, i després es realitza el test complet. A més, hi ha un apartat *extra* que permet provar algunes alternatives.

A part d'aquests dos fitxers principals, hi ha alguns auxiliars. *split_dataset.py* és el que s'encarrega de llegir les dades del corpus i separar-les en train, test i validation. *ukb.py* conté una pseudo-implementació d'ukb en Python realitzada per nosaltres, *frequencies.py* s'encarrega de realitzar un diccionar i de freqüències i textserver és el fitxer encarregat de connectar-se amb aquest aplicatiu (encara que al final no l'hem fet servir). A més, en el directori */data* trobem una gran quantitat de fitxers JSON. En distingim dos tipus: els que comencen per X i y (en el subdirectori */original_data*), que són simplement les particions corresponents de les dades, i els que acaben amb synsets (en el subdirectori */synsets*), que són els resultats de la desambiguació de sentits.

Finalment, també s'ha afegit un document *compare_sup_unsup.ipynb* on es compara el model supervisat amb el no supervisat.

# 3 Supervisat

El primer pas per tal de poder realitzar el *sentimental analysis* supervisat era definir les dades d'entrenament i de test. S'ha usat el corpus *movie reviews* de nltk, ja que conté diverses opinions ja etiquetades com a positives o negatives.

En total conté 1000 opinions positives i 1000 de negatives. Per tal de dividir en train i test, s'ha usat la funció train_test_split de sklearn ([1]). La mida de test escollida ha sigut el 25%, és a dir, un total de 500 opinions. A més, s'ha optat per usar la opció stratify, que assegura que la distribució de la classe que volem predir és la mateixa en les dues particions.

Addicionalment, per tal de provar alguns paràmetres, s'ha creat una subpartició de la de train de validació. Aquesta té una mida del 25% també, és a dir, 375 opinions.

Per no haver d'estar llegint el corpus i fent la divisó cada cop, les llistes s'han guardat en quatre fitxers JSON: X_train, X_test, y_train i y_test.

## 3.1 Preprocessament

Un cop les dades estaven separades degudament, s'ha procedit amb el preprocessament.

Després d'estar valorant les diferents tècniques que es poden aplicar a les paraules per realitzar aquest preprocessament, s'ha arribat a tres opcions diferents.

En la primera, el text no pateix cap modificació i simplement es crida el CountVectorizer (codificador *bag of words* de sklearn) amb min_df 0.0, stopwords english i strip_accents ascii. Bàsicament, realitza un preprocessament pràcticament mínim, eliminant tan sols les paraules que no aporten informació (les stopwords). Min_df 0.0 indica que no ignori cap paraula.

L'alternativa, abans de cridar al CountVectorizer lemmatitza les paraules, D'aquesta manera, per exemple els plurals els identificarà com la mateixa paraula, reduïnt així la mida de la matriu del codificador bag of words. A més, elimina els signes de puntuació i les stop_words del corpus de nltk.

Finalment, per tal de provar també els paràmetres del CountVectorizer, i després de realitzar alguns tests amb una partició de validació, s'ha optat per crear un altre preprocessament on el text no pateix cap modificació, igual que al primer, amb la diferència que aquest cop el Count Vectorizer és binari (és a dir, només indica si apareix la paraula, no quants cops).

Mencionar que, a part d'aquests tres preprocessaments, s'han provat també alternatives diferents, com per exemple en lloc de lemmatitzar stemmatitzar. A més, cal dir que el Count Vectorizer realitza un fit transform amb el train i tan sols el transform al test.

Després d'una breu recerca, també hem trobat que existeixen altres vectorizers, com el Tf-idf, que ajusta els valors en funció de quants cops apareixen i que mesura la importància de les paraules ([2]). Tot i això, se'ns demanava que fèssim servir el Count Vectorizer, o sigui que aquest és simplement un apunt d'una de les formes que es podria millorar aquest detector de sentiments.

## 3.2 Models

Existeixen una gran quantitat de models disponibles a la llibreria *sklearn* ([1]) per tal de realitzar classificació. D'aquests, en vam provar alguns d'entrada, usant una sub partició del train de validació. Els resultats d'aquesta primera validació rondaven el 80% d'accuracy, arribant a 0.8693 en el Random Forest. Es van descartar els models de KNeighbors i de GaussianNB, ja que obtenien menys d'un 70% d'accuracy, molt per sota la resta de models.

Finalment en vam escollir un subset per tal d'usarlos durant el procés de validació. Els que hem triat són:

- **GradientBoosting**: Mètode d'ensemble (boosting) que usa el gradient negatiu de la log-loss, creant regression trees. Té paràmetres com nombre d'estimadors, learning rate o max_depth.
- **AdaBoost**: Inicia creant i ajustant un classificador al dataset, i a continuació en crea còpies que es centren més en els errors. Basat en DecisionTrees. Té paràmetres sobre el nombre d'estimadors, l'algorisme i el learnign_rate.
- **RandomForest**: També basat en DecisionTrees, ajustant-los a samples del dataset. Té paràmetres de nombre d'estimadors, max_depth, criterion, min_samples_split...
- **LogisticRegression**: Regressió logística clàssica, amb paràmetres com la regularització o el màxim d'iteracions.
- **SVC**: Mètode basat en Support Vector Machines. Paràmetres de regularització (C), així com de kernel o gamma.
- **MLPClassifier**: Classificador basat en un perceptró multi capa. Se li indica amb paràmetres la mida de les *hidden layers*, la funció d'activació, el learning rate...

## 3.3 Validació

Amb tants de models i paràmetres possibles, per tal d'escollir el millor s'ha realitzat un *Grid Search Cross Validation*. Aquesta tècnica de validació parteix el train en un nombre determinat de talls, per exemple 5, i escull els paràmetres que obtenen millors resultats (en aquest cas d'accuracy). Per cada model, s'ha realitzat aquest Grid Search per determinar els seus millors paràmetres, i a continuació s'ha escollit el model amb millor accuracy de validació. Al codi es poden observar tots els paràmetres triats.

Aquest procés s'ha realitzat tant amb el preprocessament com el que tan sols realitza el Count Vectorizer.

**Resultats sense preprocessament:**

| **Classifier** | **Best Parameters** | **Accuracy** |
|---|---|---|
| GradientBoosting | {learning_rate: 0.1, max_depth: 6, random_state: 42} | 0.8020 |
| AdaBoost | {algorithm: 'SAMME.R', learning_rate: 0.5, n_estimators: 100, random_state: 42} | 0.7940 |
| RandomForest | {max_depth: 12, n_estimators: 1500, random_state: 42} | 0.8527 |
| LogisticRegression | {max_iter: 1000, random_state: 42} | 0.8293 |
| SVC | {C: 1, degree: 1, kernel: 'poly', random_state: 42} | 0.8233 |
| MLPClassifier | {alpha: 0.01, hidden_layer_sizes: (20, 10), learning_rate: 'constant', random_state: 42, solver: 'lbfgs'} | 0.8427 |
| **Best Model** | RandomForest with params: {max_depth: 12, n_estimators: 1500, random_state: 42} | 0.8527 |

Taula 1: Millors paràmetres i accuracy pels diferents classificadors, dades sense preprocessar

**Resultats amb preprocessament:**

| **Classifier** | **Best Parameters** | **Accuracy** |
|---|---|---|
| GradientBoosting | {learning_rate: 0.5, max_depth: 6, random_state: 0} | 0.7827 |
| AdaBoost | {algorithm: 'SAMME.R', learning_rate: 0.5, n_estimators: 200, random_state: 0} | 0.7960 |
| RandomForest | {max_depth: 12, n_estimators: 1500, random_state: 0} | 0.8373 |
| LogisticRegression | {max_iter: 1000, random_state: 0} | 0.8140 |
| SVC | {C: 1, degree: 1, kernel: 'poly', random_state: 0} | 0.8027 |
| MLPClassifier | {alpha: 0.001, hidden_layer_sizes: (20, 10), learning_rate: 'constant', random_state: 0, solver: 'adam'} | 0.8273 |
| **Best Model** | RandomForest with params: {max_depth: 12, n_estimators: 1500, random_state: 0} | 0.8373 |

Taula 2: Millors paràmetres i accuracy pels diferents classificadors, dades preprocessades

**Resultats vectoritzador alternatiu:**

| Classifier | Millors Paràmetres | Precisió |
|---|---|---|
| GradientBoosting | {learning_rate: 0.25, max_depth: 6, random_state: 0} | 0.7947 |
| AdaBoost | {algorithm: 'SAMME', learning_rate: 1, n_estimators: 200, random_state: 0} | 0.7907 |
| RandomForest | {max_depth: 14, n_estimators: 1500, random_state: 0} | 0.8613 |
| LogisticRegression | {max_iter: 1000, random_state: 0} | 0.8513 |
| SVC | {C: 1, degree: 1, kernel: 'poly', random_state: 0} | 0.8520 |
| MLPClassifier | {alpha: 0.001, hidden_layer_sizes: (20, 10), learning_rate: 'constant', random_state: 0, solver: 'lbfgs'} | 0.8593 |
| **Millor Model** | RandomForest amb paràmetres: {max_depth: 14, n_estimators: 1500, random_state: 0} | 0.8613 |

Taula 3: Millors paràmetres i precisió pels diferents classificadors, dades preprocessades i vectoritzador alternatiu

Sorprenentment, el preprocessament provoca que el model obtingui pitjors resultats, passant d'una accuracy de 0.8613 i 0.8527 a 0.8373. En els dos casos que usen el primer vectoritzador, el millor model era un Random Forest amb 1500 estimadors i profunditat màxima 12, mentre que el que usa el codificador binari era també un Random Forest però amb profunditat 14.

Aquests resultats però, són amb les dades d'entrenament (encara que sigui validació). Per tal de saber la precisió real del model, cal executar-los amb la partició de test.

## 3.4 Resultats

Escollint el millor model, RandomForest with params: {max_depth: 14, n_estimators: 1500, random_state: 42} i el millor preprocessament (no realitzar-ne, amb el vectorizer alternatiu), l'accuracy obtinguda era d'un 0.854. Aquest valor és lleugerament inferior a l'obtingut durant el validation i el cross-validation, però també cal tenir en compte que aquesta partició de test és més gran. El segon millor model, el perceptró multicapa, assoleix un resultat de 0.842, també bastant bo. Tot i això, si el cridem amb els valors per defecte i el deixem entrenar més d'un minut, aconsegueix millorar el resultat, assolint 0.864 (aquest fet ens indica que igual, durant el GridSearch, podiem haver provat més paràmetres. Tot i això, amb els que hem provat el temps d'execució ja es disparava fins a més de 100 minuts).

Amb el bag of words (sense binari), i max_depth 12, l'accuracy és de 0.848, lleugerament inferior, i per tant podem observar com el validation realment ens ha retornat el model millor.

Es pot visualitzar també la matriu de confusió (de les prediccions del Random Forest, ja que és l'escollit amb validation. Escollir un model basant-se en els resultats de test no és recomanable),

que ens permet veure si el model està més esbiaxiat cap a falsos positius o negatius.

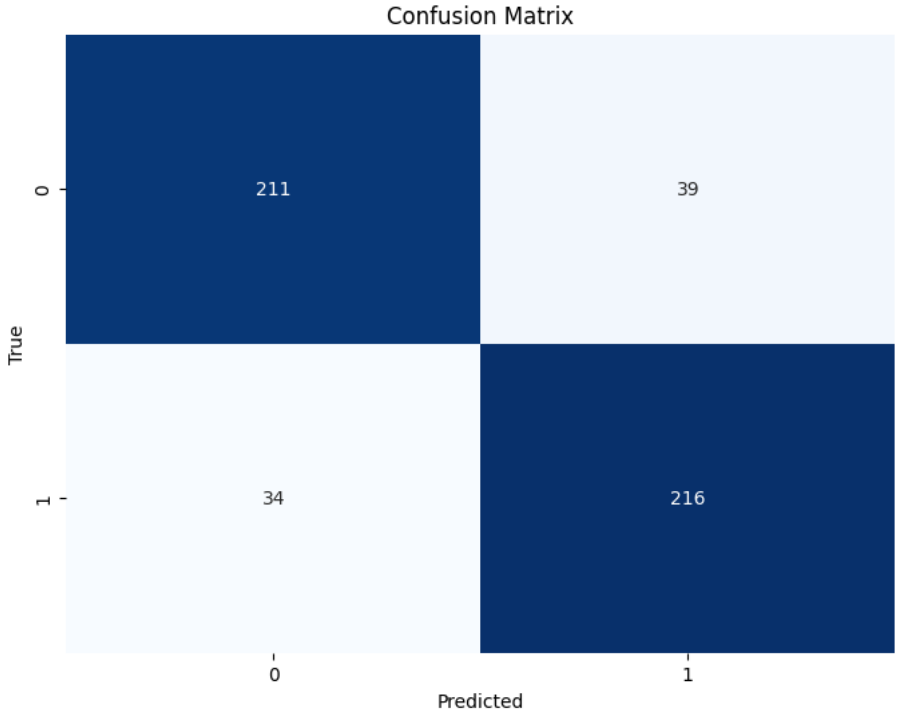


Figura 1: Matriu de confusió del Random Forest

Observem com els resultats estan molt anivellats: no té una tendència especial a predir valors extremadament positius ni negatius. És un fet positiu, ja que indica que no hi ha un biaix. Una altra mètrica bastant comuna en els classificadors binaris, com és el cas, és la ROC curve i l'AUC (area under curve). Aquesta relaciona els falsos positius amb els positius reals. A més, ens ofereix la mètrica AUC, que en aquest cas és de 0.92, bastant elevada, i indica un molt bon model. La podem observar a la figura 2:

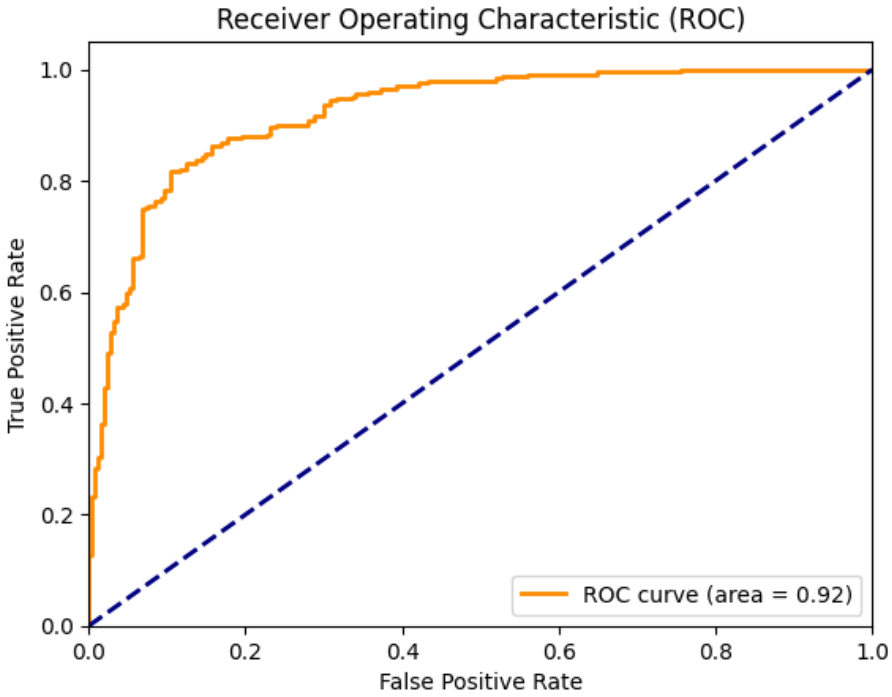


Figura 2: ROC curve del Random Forest

També podem veure la matriu de confusió del model que utilitza el vectorizer "normal" (3).

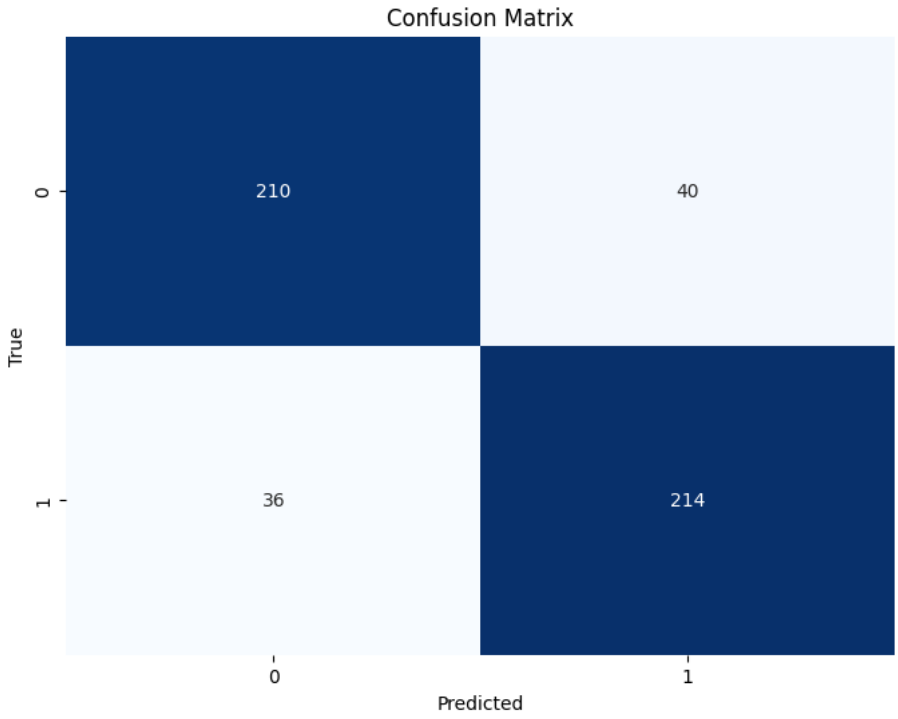


Figura 3: Matriu de confusió del Random Forest amb el vectorizer normal

Observem com realment, només s'equivoca en una sola opinió més, és a dir que tampoc estaria malament usar aquest model. En general, aquests petits canvis no afecten molt al resultat, i hem vist com amb aquest mètodes supervisats solen ser molt bons.

A més, podem observar els casos en els que s'equivoca. Alguns d'ells, és perquè enmig de la crítica es dediquen a comentar altres coses, o bé perquè tot i criticar negativament algunes parts, acaben dient que els hi ha agradat. Per exemple, un dels falsos negatius comença tal que així: "the american action film has been slowly drowning to death in a sea of asian wire - fu copycats . it's not a pretty death , and it's leaving the likes of schwartznager , stallone , and van damme wearing cement galoshes at the bottom of a kung fu sea". Aquesta part és clarament negativa, i no és d'estranyar doncs que el model pugui equivocar-se.

# 4 No supervisat

En la majoria de casos, no disponem de dades etiquetades per tal de poder aplicar aquests algorismes supervisats, i per tant hem de utilitzar altres mètodes d'aprenentatge no supervisat.

Se'ns demanava usar UKB per obtenir els synsets de les paraules, per tal de posteriorment usar SentiWordnet ([3]) per obtenir els valors referents al sentiment de cada synset. Al no haver sigut possible connectar-se al TextServer (el límit era massa baix com per poder realitzar tot el test), s'ha optat per utilitzar algunes alternatives.

## 4.1 Desambiguació de sentits

Com hem comentat, el primer pas era realitzar una desambiguació de sentits per tal d'escollir el més adequat en cada moment.

El procediment és senzill: per cada frase, es tokenitzava per paraules, es realitzava el POS tagging (usant Spacy, [4] ja que obtenia resultats més semblants a la realitat que el tagger de nltk), i dels noms, adjectius, verbs i adverbis s'obtenia el synset adequat en aquell context usant la funció Lesk de nltk (enllaç, de [5]), que usa l'algorisme de Lesk ([6]).

Tot i això, no estàvem del tot convençuts amb la idea d'usar Lesk, ja que realment és un algorisme bastant simple i crèiem que hi havia marge de millora.

Una alternativa que vam provar va ser realitzar una espècie d'implementació de un algorisme basant-nos en el que coneixiem de UKB. D'entrada, es crea un graf amb els Synsets de Wordnet, on es relacionen amb els seus hiperònims, hipònims, holònims, merònims i altres paraules relacionades. Aquest serà la base sobre la qual treballarà l'algorisme de UKB, que està basat en PageRank.

A continuació, es realitza el postagging de la frase, i s'escullen tan sols les paraules de context, i es guarden en un diccionari els seus possibles synsets per aquella POS. A partir d'aquest punt, vam decidir implementar (a la nostra manera) els tres algorismes de grafs que usa UKB més un que realitza el pagerank directament (i que és molt més ràpid, però no té base teòrica ni relaciona realment els sentits), segons estan descrits a [7] i [8]. Aquests són: ppr_w2w (PageRank personalitzat per cada paraula), ppr (PageRank personalitzat) i dfs (en el nostre cas, traditional_pagerank). Segons l'article, els millors resultats s'haurien d'obtenir amb el primer d'aquests, mentre que l'últim és el més ràpid. Tot i això, en la nostra implementació és més ràpid el ppr, ja que el dfs, per crear el subgraf ha de calcular tots els camins mínims.

Per evitar implementar l'algorisme de PageRank a mà, s'usa la llibreria networkx ([9]), que permet que el codi sigui més eficient. En teoria, existeix una implementació d'aquesta llibreria que permet usar la GPU (cuGraph, [10]), i que per tant igual permetria executar-lo de forma més ràpida (tot i això, no disposem de targetes gràfiques de NVIDIA i al Google collab no semblava que el temps d'execució fos menor).

Llastimosament, la implementació d'aquest PageRank, tot i ser capaç de desambiguar els sentits de forma prou bona (almenys amb alguns exemples que hem triat), tarda molt a executar-se (per posar-se en context, en 2 hores desambiguava tan sols unes 16 opinions usant ppr w2w). És per

això que finalment, no s'ha usat en la validació i tan sols s'ha usat alguna de les implementacions per comparar els resultats del test i veure com, canviant el desambiguador de sentits, canvien els resultats.

Finalment, una altra opció que vam decidir considerar també va ser la de simplement agafar el synset més freqüent. Per això, necessitàvem un corpus anotat, i vam escollir usar el SemCor, ([11], [12] a NLTK), ja que es mencionava a l'article de UKB. Amb aquest, vam crear un tercer desambiguador de sentits que simplement agafa el synset més freqüent de cada paraula.

Amb aquestes diferents formes de generar synsets, podiem crear comparacions addicionals a part de la resta de mètriques que comentarem.

Per realitzar la desambiguació, simplement s'iterava per cada opinió i s'obtenien les seves frases (Sentence tokenizer de NLTK), ja que els desambiguadors funcionen per frases. A més, per no haver d'estar recalculant, els resultats es guarden en un fitxer JSON per cada tipus d'algorisme.

## 4.2 Algorisme i puntuacions

Un cop creats els fitxers de synset, calia obtenir els sentiments de cada opinió. Per fer-ho, s'itera per totes les opinions, per cada frase. En aquestes, s'agafen només les posicions sintàctiques que volem (ja que una de les combinacions que ens demanaven era per exemple, usar tan sols adjectius, o també adverbis i noms...). A partir d'aquí, s'obté el SentiScore de cada token (paraula) amb SentiWordnet ([**?**]). Aquest score es divideix en 3: el positiu, el negatiu, i l'obj (la neutralitat). Si el synset no està present a SentiWordnet, simplement no es té en compte (una alternativa seria intentar trobar alguna opció diferent a SentiWordnet, però aquest fet es produeix pocs cops).

El primer dilema té lloc a aquest punt: quins scores agafem per paraula i com combinem aquests scores entre diferents paraules, per obtenir el sentiment d'una frase? Per determinar la millor manera, més endavant comentarem com vam realitzar un validation (encara que realment els mètodes no supervisats no en tenen, en aquest cas ens ajudarà a triar aquests "hiperparàmetres"donat que sí que disposem de dades etiquetades).

Les diferents opcions per determinar l'score d'una paraula són:

- **pos**: Retorna el SentiScore positiu obtingut amb SentiWordnet.

$$\texttt{pos}(s) = s.pos$$

- **neg**: Retorna el SentiScore negatiu obtingut amb SentiWordnet

$$\texttt{neg}(s) = s.neg$$

.

- **obj**: Retorna el SentiScore obj (neutral) obtingut amb SentiWordnet.

$$\texttt{obj}(s) = s.obj$$

- **max_score**: Retorna 1 quan el valor absolut de l'score positiu és major que el valor absolut de l'score negatiu, -1 si el valor absolut de l'score positiu és menor que el valor absolut de l'score negatiu, i 0 si són iguals.

$$\texttt{max\_score}(s) = \begin{cases} 1 & \text{si } |s.pos| > |s.neg| \\ 0 & \text{si } |s.pos| = |s.neg| \\ -1 & \text{si } |s.pos| < |s.neg| \end{cases}$$

- **dif**: Retorna la resta entre el l'score positiu i l'score negatiu.

$$\texttt{dif}(s) = s.pos - s.neg$$

- **dif2**: Retorna la resta dels quadrats de l'score positiu i l'score negatiu.

$$\texttt{dif2}(s) = s.pos^2 - s.neg^2$$

- **dif_threshold**: Retorna la resta de entre l'score positiu i l'score negatiu, considerant-los només si el seu valor absolut supera una llindar (*threshold*) mínim, sinó es considera el seu valor com 0 i es fa la resta igualment. Aquest llindar es determina mitjançant el paràmetre `score_threshold` de la funció `score_synsets`.

$$\texttt{dif\_threshold}(s) = \begin{cases} s.pos & \text{si } |s.pos| \geq \text{threshold} \\ 0 & \text{si } |s.pos| < \text{threshold} \end{cases} - \begin{cases} s.neg & \text{si } |s.neg| \geq \text{threshold} \\ 0 & \text{si } |s.neg| < \text{threshold} \end{cases}$$

- **dif2_threshold**: Retorna la resta de entre els quadrats de l'score positiu i l'score negatiu, considerant-los només si el seu valor absolut supera una llindar (*threshold*) mínim, sinó es considera el seu valor com 0 i es fa la resta dels quadrats igualment. Aquest llindar es determina mitjançant el paràmetre `score_threshold` de la funció `score_synsets`.

$$\texttt{dif\_threshold2}(s) = \begin{cases} s.pos^2 & \text{si } |s.pos| \geq \text{threshold} \\ 0 & \text{si } |s.pos| < \text{threshold} \end{cases} - \begin{cases} s.neg^2 & \text{si } |s.neg| \geq \text{threshold} \\ 0 & \text{si } |s.neg| < \text{threshold} \end{cases}$$

I per ajuntar-los i obtenir l'score total d'una frase es disposa dels següents mètodes ($sc$ representa la llista d'scores de totes les paraules de la frase):

- **sum**: Retorna la suma de tots els scores.

$$\texttt{sum}(sc) = \sum_{i=1}^{N} sc_i$$

  on $N$ és el nombre d'scores.

- **mean**: Retorna la mitjana dels scores.

$$\texttt{mean}(sc) = \frac{1}{N} \sum_{i=1}^{N} sc_i$$

  on $N$ és el nombre d'scores.

- **max**: Retorna el valor màxim dels scores.

$$\texttt{max}(sc) = \max(sc)$$

  on $N$ és el nombre d'scores.

- **min**: Retorna el valor mínim dels scores.

$$\texttt{min}(sc) = \min(sc)$$

- **scale_norm1_mean**: Retorna el promig de la norma-1 (suma de valors absoluts de tots els elements) dels valors escalats mitjançant la funció `min_max_scale`.

$$\texttt{scale\_norm1\_mean}(sc) = \frac{\|\text{min\_max\_scale}(sc)\|_1}{N}$$

  on $N$ és el nombre d'scores i $\texttt{min\_max\_scale}(sc)$ escala tots els elements d'$sc$ en l'interval [0,1].

- **scale_norm2_mean**: Retorna el promig de la norma-2 (arrel quadrada de la suma dels quadrats de tots els elements) dels valors escalats mitjançant la funció `min_max_scale`.

$$\texttt{scale\_norm2\_mean}(sc) = \frac{\|\text{min\_max\_scale}(sc)\|_2}{N}$$

  on $N$ és el nombre d'scores i $\texttt{min\_max\_scale}(sc)$ escala tots els elements d'$sc$ en l'interval [0,1].

Per tal d'unir els resultats de les frases entre sí, s'ha optat per simplement realitzar la mitjana, per no donar més pes a les opinions més llargues (amb més frases) ni provocar cap altre biaix.

Arribats a aquest punt, tenim una puntuació per cada opinió, però encara falta un valor a triar: a partir de on decidim que la predicció és que és positiva, ja que de moment els valors que tenim són continuus. Per tal d'escollir aquest paràmetre de discretització, també provarem amb la validation.

## 4.3 Validació

Com hem comentat, hi ha diverses decisions que s'han de prendre sobre el model. Com que poden afectar al resultat d'aquest i considerant que disposàvem de dades etiquetades, hem determinat que la millor opció era realitzar un validation. Per realitzar-la, s'han usat els synsets de lesk incloent els adjectius satèl·lit, ja que eren els més fàcils d'obtenir. Més tard, s'analitzaran els models que usen aquests.

Aquesta validació té com a objectiu determinar quina combinació de paràmetres fan que el model obtingui uns millors resultats. Per realitzar-la, s'han provat moltes combinacions de mètodes per donar score als synsets, mètodes per ajuntar els scores dels synsets, llindars (*thresholds*) per discretitzar els valors predits i tipus de paraules considerades (verbs, adjectius, adverbis, ...). Aquesta validació s'ha fet amb les dades de validació amb els synsets de Lesk, que contenen un 25% de les dades d'entrenament originals (75% del total). La mètrica utilitzada per aquest validation és l'accuracy, ja que les dues classes a predir estan balancejades.

Els paràmetres que s'han provat són els següents:

- **Conjunts de paraules**: Totes les possibles combinacions amb totes les possibles longituds del conjunt $\{n, a, s, v, r\}$. Aquestes lletres representen els POS tags assignats a cada tipus de paraula que utilitza Lesk [13], on $n$ representa noms, $a$ representa adjectius, $s$ representa adjectius satèl·lit, $v$ representa verbs i $r$ representa adverbis. Totes aquestes combinacions s'han realitzat mitjançant la llibreria `itertools`.
- **Scores**: Tots els scores permesos per la funció `score_synset`, explicats prèviament. És a dir: `pos`, `neg`, `obj`, `max_score`, `dif`, `dif2`, `dif_threshold` i `dif2_threshold`.
- **Scores thresholds**: Aquests valors s'utilitzen només per quan el mètode de score és `dif_threshold` o `dif2_threshold` i els valors provats són 0.05, 0.125, 0.25, 0.4 i 0.6. Tots són positius ja que representen el valor mínim que ha de tenir l'score positiu/negatiu per tal de ser considerat en la resta de valors absoluts (o de valors al quadrat per `dif2_threshold`).
- **Merges**: Els diferents mètodes que s'han provat per ajuntar scores de paraules són: `sum`, `mean`, `min`, `max`, `scale_norm1_mean` i `scale_norm2_mean` (tots els disponibles en la funció `score_synsets`). Cal tenir en compte que quan l'score és `max_score`, només es permet utilitzar els mètodes de `sum` i `mean`, per tal de que els resultats convergeixin a prop del valor 0 i siguin coherents.
- **Thresholds per discretitzar**: Els llindars que s'han provat per discretitzar les prediccions són -0.6, -0.4, -0.25, -0.125, -0.05, 0, 0.05, 0.125, 0.25, 0.4 i 0.6.

Cal mencionar que tots aquests valors per la validació han estat escollits en funció de prèvies execucions que s'han realitzat, que han acabat amb resultats poc satisfactoris i han permès ajustar els paràmetres que es proven per tal de que se'n puguin provar pocs i garantint que cap d'ells és prescindible en la validació.

L'execució d'aquesta validació ha tardat aproximadament 3 hores (170 minuts) i els millors resultats obtinguts es poden veure en la taula 4. En aquest cas, el conjunt de paraules que fan que el model rendeixi millor són: noms, adjectius, adjectius satèl·lit i adverbis $\{n, a, s, v, r\}$). És important notar que sorprenentment els verbs ($v$) no estan en cap dels models que obtenen millors resultats. A més, també es vol remarcar que l'accuracy obtingut amb aquests conjunts de paràmetres és de només 0.656, un valor relativament baix, de manera que no podem esperar que aquest model tingui un gran encert en el test.

| **POS tags** | **Best Parameters** | **Accuracy** |
|---|---|---|
| $\{n, a, s, v, r\}$ | {Score: `dif`, Merge: `sum`, Threshold: 0} | 0.656 |
| $\{n, a, s, v, r\}$ | {Score: `dif_threshold`, Score threshold: 0.05, Merge: `sum`, Threshold: 0} | 0.656 |
| $\{n, a, s, v, r\}$ | {Score: `dif_threshold`, Score threshold: 0.125, Merge: `sum`, Threshold: 0} | 0.656 |

Taula 4: Millors paràmetres i el seu accuracy en la validació

Cal mencionar que si els millors POS tags no fossin els mateixos en tots els casos, per determinar quin és el conjunt de paraules (POS tags) que fan que el model rendeixi millor, es desempatarien els resultats calculant la mitjana de l'accuracy d'aquell conjunt en totes les seves combinacions de

paràmetres, i si es mantingués l'empat s'escolliria el que tingués menys tipus de paraules, per tal de tenir un model més senzill (mantenint els bons resultats). Si encara hi hagués un empat, s'escolliria arbitràriament el primer conjunt de POS tags trobat.

Per tal de visualitzar el rendiment del model amb els diferents paràmetres, s'ha realitzat un *heatmap* per cada possible llindar (*threshold*) de discretització de les prediccions en el que es mostra l'accuracy del model per cada possible mètode de score i de fusió de scores (*merge*). Aquests gràfics es poden veure en les figures 4, 5 i 6 i clarament podem notar com els millors resultats es troben en els gràfics de la figura 5, on els thresholds són propers a 0. També es pot observar com la majoria de millors resultats es concentren en el mètode de merge `sum`, tot i que `mean` també té alguns valors bastant alts. En general els millors mètodes de score són `dif` i `dif_threshold` amb valors de `score_threshold` baixos. A més, també es pot apreciar com en general el pitjor mètode d'score és el `max_score`.

Veient aquests resultats, s'ha decidit fixar l'score `dif` com el definitiu degut a la seva major senzillesa i a que no requereix de cap paràmetre addicional. És a dir, els paràmetres finals per provar el model en el test són:

- **Conjunts de paraules**: $\{n, a, s, r\}$. Tots menys els verbs.
- **Score**: `dif`.
- **Score threshold**: Cap, degut a que `dif` no en requereix.
- **Merge**: `sum`.
- **Threshold per discretitzar**: 0.

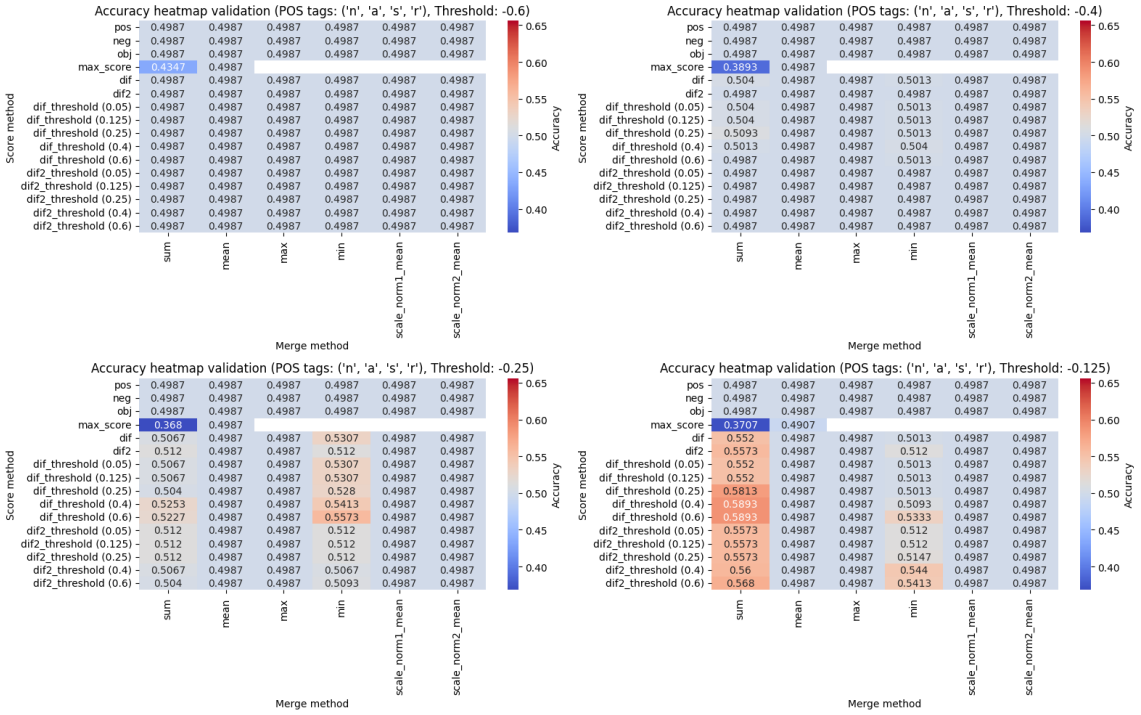


Figura 4: *Heatmap* de l'accuracy per les diferents combinacions de paràmetres amb thresholds -0.6, -0.4, -0.25 i -0.125

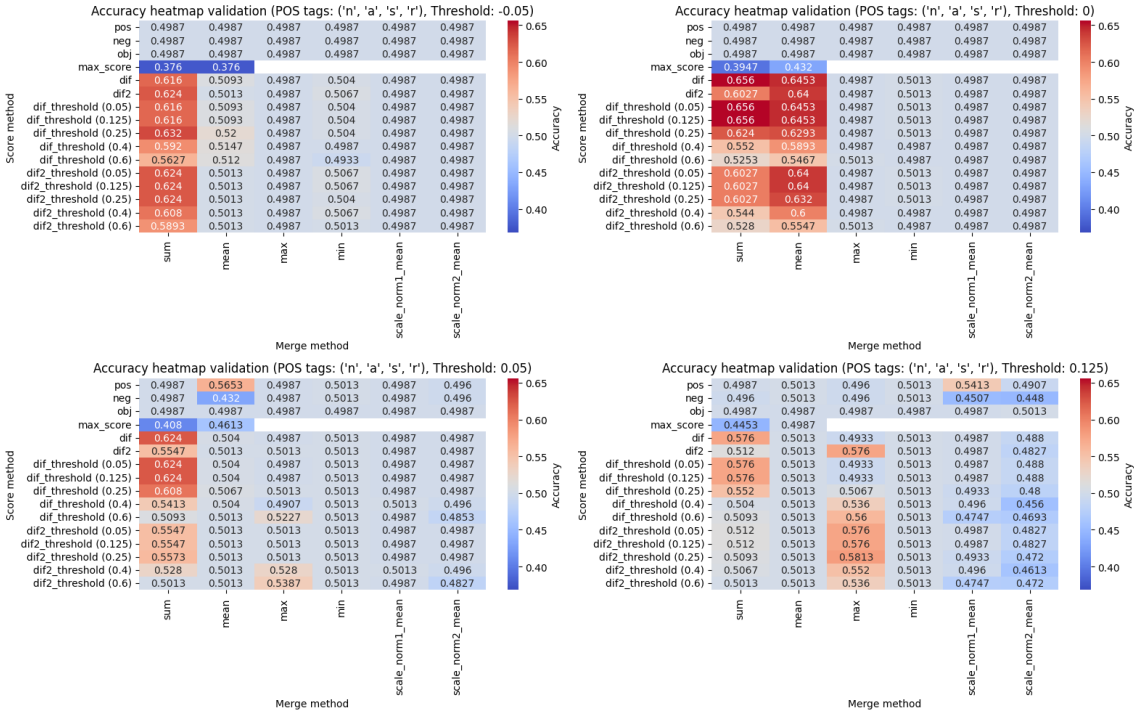


Figura 5: *Heatmap* de l'accuracy per les diferents combinacions de paràmetres amb thresholds -0.05, 0, 0.125 i 0.25

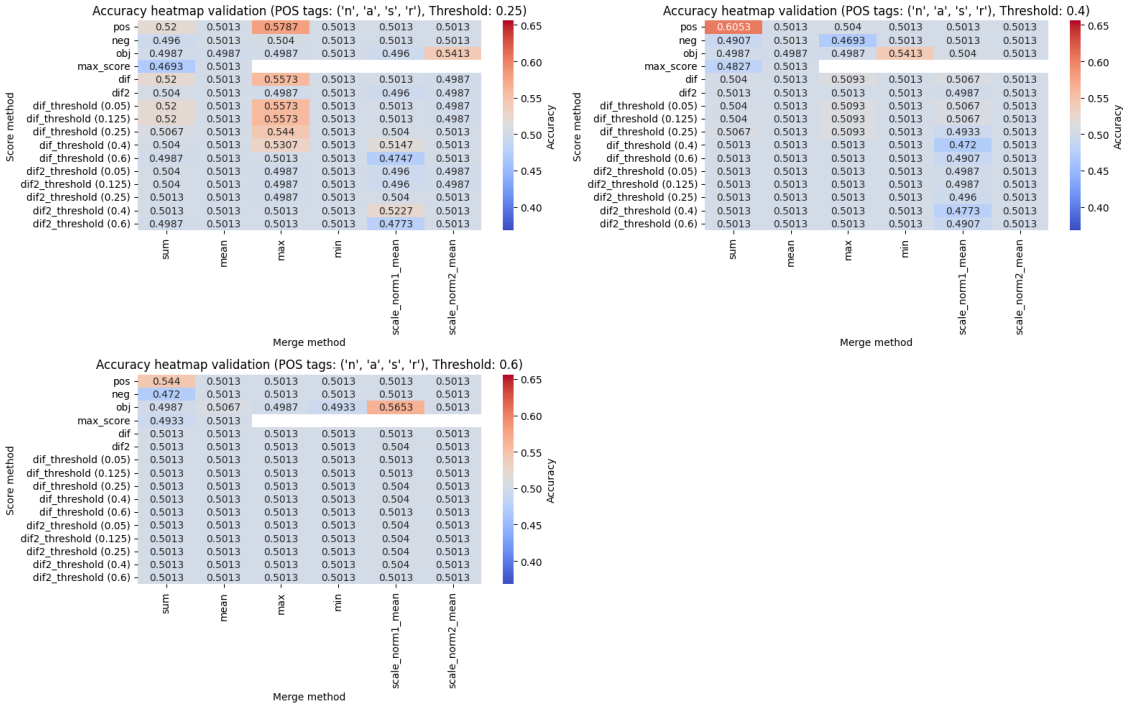


Figura 6: *Heatmap* de l'accuracy per les diferents combinacions de paràmetres amb thresholds 0.25, 0.4 i 0.6

Per altra banda, en la figura 7 es pot veure la matriu de confusió de les prediccions fetes pel model amb "millors paràmetres" en la partició de validació. Es pot apreciar com hi ha una mica més de falsos negatius que de falsos positius, tot i que la majoria de valors recauen en els valors correctes. Aquest petit biaix cap als falsos positius pot semblar "fàcil d'arreglar" si es disminueix molt lleugerament el threshold de discretització. No obstant, al provar-ho amb el threshold -0.01 (en comptes de 0) obtenim certament uns valors més balancejats, tal i com es pot veure en la figura 8, però això és a costa de que els falsos positius augmentin i ara l'accuracy sigui de 0.640 (menys que els 0.656 previs).

Per tant, s'ha decidit no modificar el threshold establert previament al realitzar la validació i mantenir el threshold 0 per fer les proves en el test.

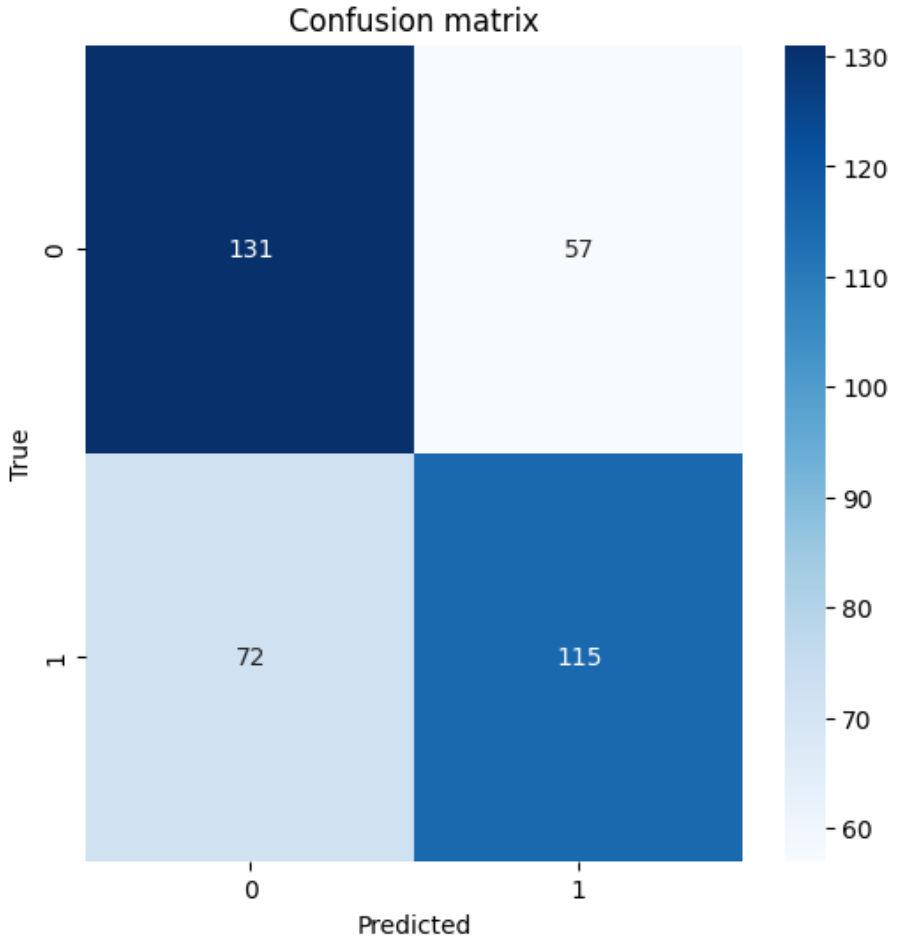


Figura 7: Matriu de confusió en la partició de validació amb threshold de discretització 0

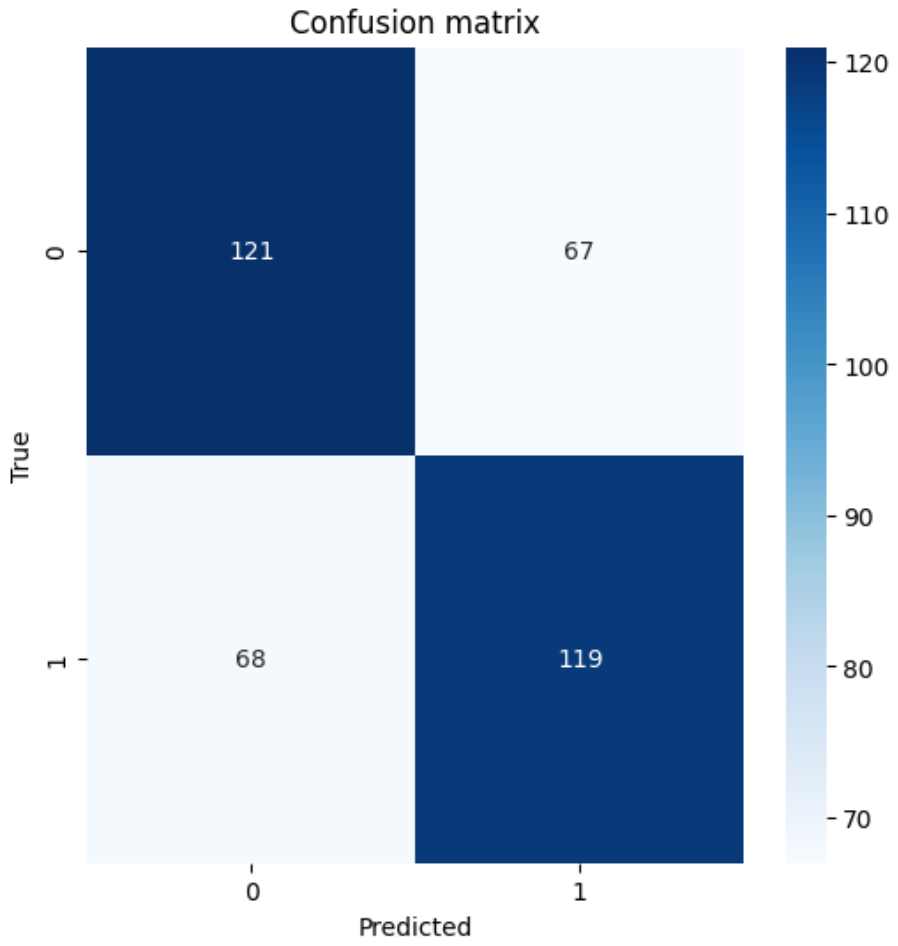


Figura 8: Matriu de confusió en la partició de validació amb threshold de discretització -0.01

## 4.4 Resultats

Una vegada determinats els millors paràmetres del nostre model, s'ha comprovat com rendeix en el test. Al fer-ho s'ha obtingut un accuracy de 0.644, molt semblant al que s'havia aconseguit en la partició de validació. En la figura 9 es pot veure la matriu de confusió del model en el test, on crida l'atenció la gran quantitat de valsos positius (predir classe positiva quan realment és classe negativa) que hi ha, a més tenint en compte que en la partició de validació hi havia més falsos negatius que falsos positius.

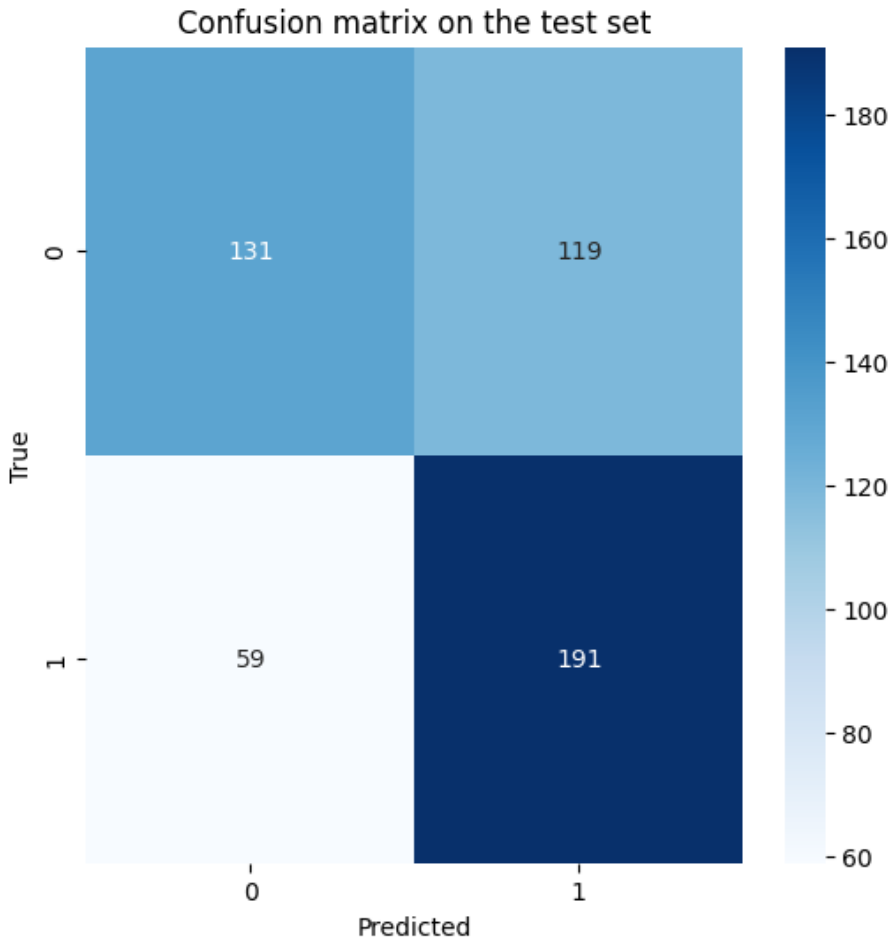


Figura 9: Matriu de confusió en la partició de test amb threshold de discretització 0

De nou, sembla que aquest biaix pugui ser arreglat mitjançant un ajust en el paràmetre de threshold. Per comprovar-ho, s'ha fet la prova d'augmentar una mica el threshold i s'ha pogut comprovar que augmentant-lo fins a 0.035 s'obtenen uns resultats molt millors, tal i com es mostra en la figura 10. Ara l'accuracy es de 0.652 (millor que el 0.644 previ); no obstant, **aquesta informació no pot ser extrapolada a cap decisió ni canvi en els paràmetres del model**, ja que això provocaria que tinguessim *data leakage*, on el model apren informació específica del conjunt de test, la qual no hauria de ser accessible durant l'entrenament.

Aquesta pràctica pot resultar en una estimació inflada de la capacitat real del model i és crucial mantenir una estricta separació entre els conjunts de dades d'entrenament i de test per evitar cap mena de fuga de dades.

A més, com a curiositat s'ha provat el treshold de 0.035 en el conjunt de validació i l'accuracy ha baixat de 0.656 a 0.616.

Per tant, el model s'ha de quedar amb el threshold inicial establert en la validació.

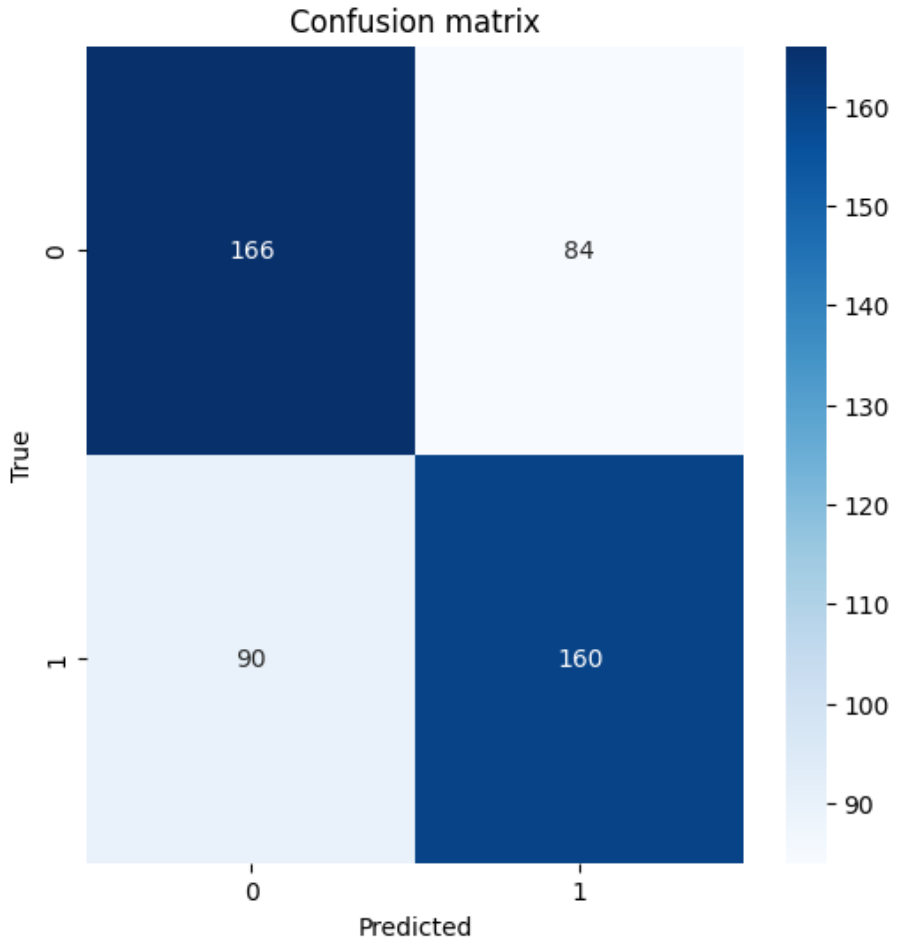


Figura 10: Matriu de confusió en la partició de test amb threshold de discretització 0.035

Com s'ha pogut veure, en general, l'accuracy d'aquests models no supervisats ronda el 0.66%. Aquest valor no és gaire alt, sobretot comparant-lo amb els de l'aprenentatge supervisat. Tot i això, probablement amb un desambiguador de sentits millor (la implementació de UKB de TextServer, per exemple, o la nostra 3) aquest resultat tindria un marge de millora, probablement arribant a 0.7.

Podem, enlloc de discretitzar els resultats, analitzar-los com una variable contínua. L'histograma es pot observar a la figura 11:

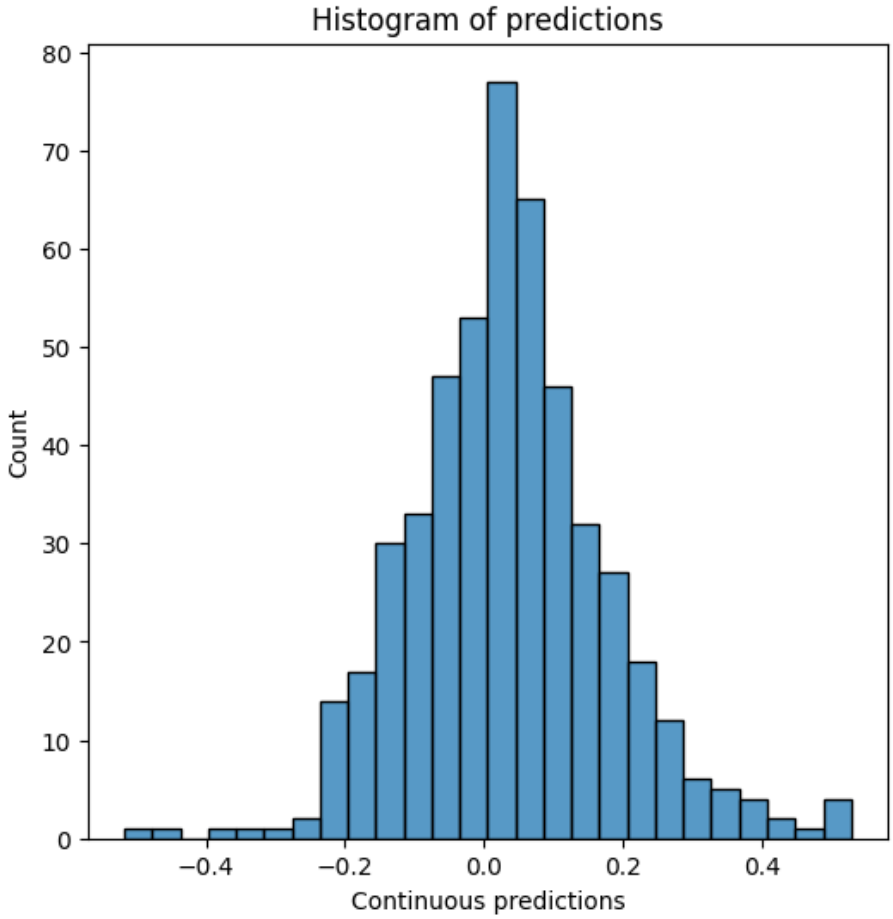


Figura 11: Histograma dels scores de cada opinió

Sorprentment, la distribució és molt normal. Aquest fet ens ha fet pensar que podiem usar la funció de distribució acumulada de la normal per tal d'obtenir unes probabilitats. Amb aquestes probabilitats, podem dibuixar la ROC curve 12.

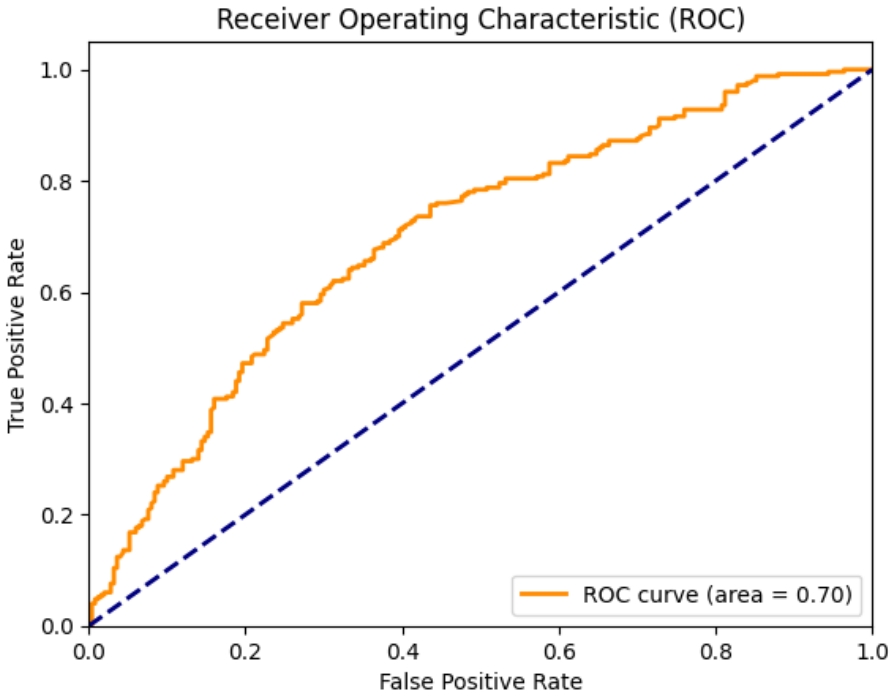


Figura 12: Curva ROC de la partició de test després d'obtenir probabilitats

En general, però, és complicat realitzar un algorisme no supervisat, i solen tenir pitjors resultats que els que sí que ho són. Tot i això, aquest ens ofereix una flexibilitat i un rang d'aplicacions més alt que no el supervisat. D'entrada, no depèn de les dades d'entrenament, o sigui que no passa res si aquestes estan esbiaixades per algun motiu o tenen algun problema. A més, els models de Sklearn que hem utilitzat són classificadors, mentre que el que ens retorna el no supervisat és una puntuació que permet una major interpretabilitat. En general, com més alta més positiva és la ressenya i com més baixa més negativa.

## 4.5 Extra

Addicionalment, vam decidir realitzar algunes proves. Per començar, com hem comentat abans disposavem d'altres desambiguacions de sentits, com per exemple la nostra implementació del ppr w2w de UKB. Aquesta tardava molt a executar, però en principi era la millor forma per desambiguar. Es va decidir executar-la, i obtenir almenys algunes opinions per tal d'analitzar com es comportava. Després d'una execució de més de 6 hores, es van obtenir les primeres 50 opinions desambiguades.

Executant, utilitzant els noms, els adjectius, els adverbis i els verbs (els adjectius satèl·lit "ss" no), sumant els positius i restant els negatius, i directament amb el threshold a 0, vam obtenir una accuracy de 0.7. Aquesta és més alta que qualsevol altra de les que hem obtingut amb el Lesk, i

tot i que pot ser que sigui degut a la mida reduïda del test, és una millora respecte al model que teniem. Per tant, aquesta desambiguació sembla prometedora, i probablement obtingui resultats millors.

En general, el comportament de la desambiguació amb UKB o Lesk provocava que alguns tipus de paraules tinguèssin més importància, o que canviessin alguns paràmetres. Per tant, és evident que afecten en gran mesura al model.

Un altre conjunt de synsets del que disposavem era el que simplement agafava el més freqüent. Podem també executar el test amb aquest, obtenint 0.574, encara que probablement provant amb més combinacions seria possible obtenir millors resultats.

Hi ha més conjunts de synsets per si es volen realitzar proves, com un que lemmatitza, un que elimina stopwords, o les altres opcions de la nostra versió de UKB.

Una altra idea que voliem provar era la detecció de les negacions, ja que evidentment influeixen a la hora de detectar la polaritat d'un text. Vam crear un preprocessament senzill, que tan sols detectava les paraules "not" i "t" (per exemple didn ' t estava separat), i buscava l'antònim màxim (el contrari més contrari) utilitzant la path similarity. Realitzat aquest canvi, es substituïa a la frase. Aquesta solució tampoc seria la millor, però és una millora respecte al que teniem (que directament no detectava negació). Amb aquest synset negat, els resultats de test són 0.634 amb un threshold de 0.05 (a 0 són 0.592).

Tot i això, no teniem cap referència de si aquests models realment eren bons o no, a part de comparar-los entre ells. Per evitar això, vam aprofitar que nltk conté una implementació de VADER, [14], per comparar-lo amb els nostres resultats. Aquest, ens permet obtenir una polaritat directament per cada frase, i després les ajuntem de la mateixa manera que fèiem amb les nostres implementacions. L'accuracy d'aquest model és 0.64 fet que ens indica que el nostre és bastant bo, ja que no varia gaire respecte aquest i de fet és millor.

# 5 Conclusions

## 5.1 Comparació de resultats

Per tal de comparar els resultats, utilitzarem els millors de la part supervisada i de la part no supervisada.

Ja s'ha vist que el model supervisat és millor en general. No obstant, es volia saber l'accuracy de cada model en les males prediccions de l'altre. En comprovar-ho, s'ha vist que el model no supervisat té un accuracy de 0.45 en les prediccions errònies del model supervisat, mentre que el model supervisat té una accuracy de 0.78 en les prediccions errònies del model no supervisat.

Això reitera la conclusio de que és millor el model supervisat.

## 5.2 Propostes de millora

Durant el procés de creació d'aquest analitzador de sentiments, s'han anat plantejant idees que probablement podrien millorar el resultat dels models però que per diversos motius, especialment falta de temps, finalment no s'han dut a terme.

En l'apartat de supervisat, es podria aplicar també l'algorisme que inverteix les paraules negades, o algun més elaborat. A més, un preprocessament més exhaustiu probablement obtindria millors resultats (encara que el que hem fet nosaltres sembla ser que anava pitjor). Una millora que creiem que seria substancial seria prescindir directament del CountVectorizer i substituïr-lo per uns *embeddings*, que tenen l'habilitat de representar millor la informació de les frases. A més, probablement algun mètode més avançat d'aprenentatge supervisat, basat en Xarxes Neuronals Recurrents per exemple, seria capaç d'interpretar millor el context de la frase i per tant probablement retornaria millors prediccions. Tot i això, el model obtingut és bastant bo, sobretot tinguent en compte que és bastant "senzill"dintre de tot.

Pel que fa l'altre apartat, hem comentat com els diferents synsets podien afectar al model. Evidentment, la implementació de UKB de TextServer obtindrà uns synsets més reals que no pas les que hem realitzat nosaltres, i aquest fet segurament milloraria la precisió del model. Tot i això, molt probablement no arribaria a l'accuracy del model supervisat.

## 5.3 Final

Els resultats obtinguts reflexen com els models d'aprenentatge supervisat obtenen resultats molt superiors als basats en no supervisat (basats en el coneixement). Aquest tipus de models de machine learning es beneficien de disposar d'una gran quantitat de dades, i és probable que amb més dades d'entrenament pugui aconseguir resultats encara millors.

Tot i això, a mesura que augmenten aquestes dades també augmentarà el temps d'entrenament d'aquest model. L'aprenentatge no supervisat es pot executar per cada opinió de manera independent,

sense dependre de cap tipus de dades. Malauradament, en el nostre cas depèn de poder disposar d'un bon desambiguador de sentits i d'escollir una manera d'ajuntar els resultats adequada.

En general, els dos models implementats han assolit bons resultast tinguent en compte les seves limitacions i les nostres. Hem determinat que encara hi ha marge de millora, però aquest també hi és en general en tot l'àmbit de l'anàlisi de sentiments i del processament del llenguatge humà.

# 6 Referències

[1] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[2] Claude Sammut and Geoffrey I. Webb, editors. *TF–IDF*, pages 986–987. Springer US, Boston, MA, 2010.

[3] Andrea Esuli and Fabrizio Sebastiani. Sentiwordnet: A publicly available lexical resource for opinion mining. 05 2006.

[4] Matthew Honnibal and Ines Montani. spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing. To appear, 2017.

[5] Steven Bird, Ewan Klein, and Edward Loper. *Natural language processing with Python: analyzing text with the natural language toolkit*. "O'Reilly Media, Inc.", 2009.

[6] Michael Lesk. Automatic sense disambiguation using machine readable dictionaries. *International Conference on Systems*, 06 1986.

[7] Eneko Agirre, Oier López De Lacalle, and Aitor Soroa. The risk of sub-optimal use of open source nlp software: Ukb is inadvertently state-of-the-art in knowledge-based wsd, 2018.

[8] Eneko Agirre and Aitor Soroa. Personalizing pagerank for word sense disambiguation, 2009.

[9] Aric Hagberg, Pieter Swart, and Daniel S Chult. Exploring network structure, dynamics, and function using networkx. Technical report, Los Alamos National Lab.(LANL), Los Alamos, NM (United States), 2008.

[10] RAPIDS. *cuGraph*, 2021. Version 24.06.00, `https://github.com/rapidsai/cugraph`, Accessed: April 9, 2024.

[11] George A Miller, Claudia Leacock, Randee Tengi, and Bunker Ross Thomas. A semantic concordance. *Human Language Technology*, 03 1993.

[12] Nltk semcor corpus documentation. Accessed on 2024-04-09.

[13] Nltk lesk algorithm documentation. Accessed on 2024-04-09.

[14] C. Hutto and Eric Gilbert. Vader: A parsimonious rule-based model for sentiment analysis of social media text. *Proceedings of the International AAAI Conference on Web and Social Media*, 8:216–225, 05 2014.