UNIVERSITAT POLITÈCNICA DE CATALUNYA
BARCELONATECH
Facultat d'Informàtica de Barcelona

FIB

Detector de polaritat en un text d'opinió

Pràctica 1

Processament del llenguatge humà

Intel·ligència artificial

Autors

Llum Fuster Palà

Artur Aubach Altes

Grup 11

Salvador Medina Herrera

Quadrimestre Primavera 2022/2023

CONTINGUTS

1. INTRODUCCIÓ

En aquest treball, es presenten dos enfocaments per a la detecció de la polaritat positiva i negativa en textos d'opinió. L'objectiu principal consisteix a explorar i comparar el rendiment dels mètodes supervisats i no supervisats en l'anàlisi de sentiments, així com analitzar les diferents tècniques de processament del llenguatge natural (NLP) en la llengua anglesa. Les dades utilitzades provenen del Movie Reviews Corpus del NLTK.

En la primera part del treball (part A), es desenvolupa un detector d'opinions basat en l'aprenentatge supervisat, utilitzant diferents algoritmes de la llibreria scikit-learn. Aquest model supervisat empra un 80% de dades per a train i 20% per a test. Per a garantir la fiabilitat del model, es dissenya i aplica un protocol de validació creuada, i es realitza un preprocés de les dades, incloent l'eliminació de nombres, signes de puntuació, d'stop words i el tractament de l'apòstrof. La representació de la informació es duu a terme mitjançant el CountVectorizer, que converteix les dades en matrius de comptador de paraules enfront de documents.

En la segona part del treball (part B), es proposa un detector d'opinions no supervisat. Aquest enfocament empral Lesk de l'NLTK per a obtenir els synsets de les paraules i els valors SentiWordnet per a puntuació positiva, negativa i objectiva de cada synset. Es fan servir les mateixes dades de test que en la part A, i es proposen diverses estratègies per a combinar els valors obtinguts, tenint en compte les possibles limitacions com l'absència de synsets a SentiWordnet. Adicionalment, es consideren diverses categories per a l'anàlisi: únicament adjectius; noms, adjectius i adverbis; i noms, adjectius, verbs i adverbis.

A través d'aquest treball, es pretén proporcionar una visió completa i comparativa dels dos enfocaments en l'anàlisi de sentiments.

2. OBTENCIÓ DE DADES

En aquest treball, s'utilitza el corpus "movie_reviews" de la llibreria NLTK per obtenir les dades necessàries. Inicialment, extraiem els identificadors de cada document que correspon a una opinió de pel·lícula escrita en anglès. Mitjançant la funció `obtenir_ids`, aconseguim els identificadors de les opinions negatives i positives, separadament.

```
def obtenir_ids(x): # x és l'etiqueta pos o neg per obtenir les dades dels documents d'opinió positiva i negativa
    lst = []
    for i in mr.fileids(x):
        lst.append(i)
    return(lst)
```

Seguidament, procedim a dividir les dades en conjunts de train i test. La funció `dividir` retorna dues llistes: una amb 800 identificadors per al traini l'altra amb 200 identificadors per al test. Si es desitja canviar el percentatge de divisió, només cal modificar el valor `0.2`.

```
def dividir(dad,semilla):
    random.seed(semilla)
    split_index = int(len(dad) * 0.2)
    shuffled = random.sample(dad, len(dad))
    train = shuffled[split_index:]
    test = shuffled[:split_index]
    return train, test
```

Finalment, obtenim les llistes de paraules de cada document mitjançant el mètode words. La funció `obtenir_paraules` retorna una llista de llistes de paraules corresponents a cada document. Apliquem aquesta funció tant per als conjunts d'entrenament com de prova, en els textos positius i negatius.

```
def obtenir_paraules(ids): # ids és la llista d'identificadors de la qual es volen obtenir les paraules
    lst = []
    for i in ids:
        lst.append(mr.words(i))
    return(lst)
```

Així doncs, les nostres dades quedarien en el format següent:

```
[[paraula1_doc1, paraula2_doc1, paraula3_doc1],[paraula1_doc2, paraula2_doc2]]
```

3. MODEL SUPERVISAT

En aquesta secció, es desenvolupa un model supervisat per a la detecció de polaritat en textos d'opinió, seguint els objectius principals del treball. Aquest model es basa en l'ús d'algoritmes de la llibreria scikit-learn per a l'aprenentatge supervisat. La metodologia del model supervisat consisteix en quatre etapes: preprocés (4.1), countvectorizer (4.2), entrenament de models (4.3) i anàlisi de resultats (4.4).

Per a desenvolupar el model supervisat, primer realitzarem un preprocés de les dades obtingudes a l'apartat 3 (obtenció de dades). Aquest preprocés consisteix en l'eliminació de caràcters no desitjats (4.1.1) i l'eliminació de stopwords (4.1.2). A continuació, transformarem les dades preprocessades en una representació numèrica utilitzant la tècnica CountVectorizer (4.2), que crea matrius de comptador de paraules enfront de documents.

Un cop les dades estiguin preprocessades i vectoritzades, procedirem a l'entrenament de diferents models supervisats utilitzant la llibreria scikit-learn (4.3). Per a optimitzar els hiperparàmetres dels models i seleccionar el millor model per a la tasca, realitzarem una cerca en quadrícula (Grid Search) (4.3.1) i compararem els resultats obtinguts per a la tria del model més adequat (4.3.2).

Finalment, analitzarem el resultats del model selecionat (4.4), avaluant el seu rendiment en termes de precisió, exhaustivita.

Amb aquesta metodologia, buscarem assolir els objectius plantejats pel model supervisat, així com proporcionar una base sòlida per a la comparació amb el model no supervisat, que es desenvoluparà en l'apartat 5.

3.1 PREPROCESING

El preprocés és una fase crucial en qualsevol treball que impliqui l'anàlisi i processament de dades de text, especialment en tasques com la detecció de polaritat d'opinions. Aquesta etapa té com a objectiu preparar i netejar les dades per facilitar el treball dels algoritmes de classificació i millorar la seva eficiència i precisió. En el nostre cas, el preprocés s'aplica tant en els caràcters de les opinions com en la filtració de paraules no rellevants a través de l'ús de stopwords.

El preprocés és important per diverses raons:

1. Reducció de soroll: Eliminar elements no desitjats, com caràcters especials, signes de puntuació o números, que poden afegir soroll i distorsionar els resultats del model.
2. Consistència de les dades: Assegurar que les dades estiguin en un format estandarditzat i consistent, facilitant així la seva interpretació i processament pels algoritmes de classificació.
3. Reducció de la dimensionalitat: Filtrar paraules no rellevants, com ara stopwords, permet reduir la quantitat de característiques a considerar pels algoritmes de classificació, optimitzant el temps de processament i millorant la precisió dels resultats.

En el nostre treball, el preprocés s'ha aplicat en dues etapes: eliminació de caràcters no desitjats (4.1.1) i filtració de stopwords (4.1.2). A través de l'eliminació de caràcters, es treuen elements com signes de puntuació, números i símbols que podrien afectar negativament els resultats del model. D'altra banda, aplicar un filtre de stopwords permet excloure paraules com articles, preposicions o conjuncions que, tot i ser freqüents en els textos, no aporten informació rellevant per a la detecció de polaritat d'opinions.

En resum, el preprocés és una etapa indispensable per assegurar que les dades estiguin en un format adequat i net de soroll, facilitant així el treball dels algoritmes de classificació i millorant la qualitat dels resultats obtinguts.

3.1.1 CARÀCTERS

En aquest apartat, abordem el preprocés dels caràcters en els textos de les opinions utilitzant dos tipus diferents de preprocés. En apartats posteriors, analitzarem quin dóna millors resultats. Aquests preprocesos s'apliquen tant en les dades d'entrenament com en les de prova, per a les opinions positives i negatives.

A. El primer preprocés utilitza la funció `nomes_lletres_nbsp`, que aplica l'expressió regular `[^a-zA-Z]|nbsp` per conservar només les lletres de l'alfabet en majúscules o minúscules i eliminar la paraula "nbsp", que hem observat que apareix freqüentment en les opinions i no té cap significat.

```
def nomes_lletres_nbsp(text):
    lst = []
```

```
    for i in range(len(text)):
        lst.append([])
        for j in range(len(text[i])):
            x = re.sub('[^a-zA-Z]|nbsp', '', text[i][j])
            if x != '':
                lst[i].append(x)
    return lst
```

B. El segon preprocés, implementat per les funcions `grup_apostrofs` i `nomes_lletres_nbsp_apostrof`, aplica un preprocés que té en compte també els apòstrofs. Primer, utilitza la funció `grup_apostrofs` per mantenir els apòstrofs en conjunció amb les lletres adjacents. Després, aplica l'expressió regular `[^a-zA-Z']|nbsp` amb la funcio `nomes_lletres_nbsp_apostrof` per conservar les lletres de l'alfabet en majúscules o minúscules, eliminar la paraula "nbsp" i mantenir els apòstrofs .

```
def grup_apostrofs(words):
    new_words = []
    for w in words:
        new_w = []
        j = 0
        while j < len(w):
            if w[j] == "'":
                if j == 0:
                    new_w.append(w[j] + w[j+1])
                    j += 2
                elif j == len(w)-1:
                    new_w[-1] += w[j]
                    j += 1
                else:
                    new_w[-1] += w[j] + w[j+1]
                    j += 2
            else:
                new_w.append(w[j])
                j += 1
        new_words.append(new_w)
    return new_words


def nomes_lletres_nbsp_apostrof(text):
    lst = []
    for i in range(len(text)):
        lst.append([])
        for j in range(len(text[i])):
            x = re.sub("[^a-zA-Z']|nbsp", '', text[i][j])
```

```
            if x != '':
                lst[i].append(x)
    return lst
```

Ambdós mètodes de preprocés serveixen per eliminar caràcters no desitjats i preparar les dades per al processament posterior. En les següents seccions, analitzarem quin d'aquests mètodes de prepreprocés ofereix millors resultats en el nostre treball. Serà important analitzar l'impacte d'aquestes diferents estratègies de preprocés en les opinions per determinar quina és més eficaç a l'hora de millorar la precisió i la qualitat dels resultats obtinguts en la classificació de les opinions de pel·lícules.

3.1.2 STOPWORDS

En aquest apartat, abordarem el procés d'eliminació de stopwords, paraules que no aporten significat rellevant per a la classificació de polaritat de les opinions. Hem utilitzat dos enfocaments diferents per a l'aplicació de stopwords als dos tipus de preprocés previament mencionats.

A. En primer lloc, creem un conjunt personalitzat de stopwords basat en la llista proporcionada per nltk.download('stopwords'). D'aquesta llista, eliminem algunes paraules que considerem rellevants per a la detecció de polaritat, com ara `"couldn't", "doesn't", "don't", "wasn't"`, entre d'altres. Aquesta eliminació es realitza mitjançant la funció `remove_set2_from_set1`, que rep com a paràmetre la llista de stopwords i retorna una nova llista sense les paraules específiques que es volen conservar.

```
def remove_set2_from_set1(list1):
    important_words = ["couldn't", "doesn't", "don't", "wasn't", "wouldn't",
"shan't", "needn't", "aren't", "shouldn't", "hadn't", "mustn't", "isn't",
"won't", "haven't", "weren't", "mightn't", "didn't", "hasn't"]
    new_list = list1.copy()
    for elem in important_words:
        if elem in new_list:
            new_list.remove(elem)
    return new_list
```

B. El segon enfocament per aplicar stopwords es realitza en l'apartat 4.2, utilitzant la funció CountVectorizer de la llibreria sklearn. En aquest cas, farem servir els

stopwords per defecte que proporciona aquesta funció, sense eliminar les paraules rellevants previament esmentades. Aquesta opció ens permetrà comparar l'efectivitat dels dos enfocaments en la classificació de polaritat de les opinions de pel·lícules i determinar quin dóna millors resultats.

3.2 COUNTVECTORIZER

En aquest apartat, es presenta el procés de transformació de les dades preprocessades en una representació numèrica utilitzant la tècnica CountVectorizer. Aquesta tècnica es basa en la creació de matrius de comptador de paraules enfront de documents, que permeten quantificar la importància de les paraules en les opinions i facilitar el treball dels algoritmes de classificació.

El CountVectorizer és una funció de la llibreria scikit-learn que converteix una col·lecció de documents de text en una matriu de comptadors de paraules. Per aplicar el CountVectorizer, primer es crea un objecte vectoritzador amb els paràmetres desitjats, com ara la llista de stopwords a excloure (3.1.2) i els límits de freqüència mínima i màxima de les paraules (0.15 i 0.85 respectivament).

Primerament hem definit les funcions de tokenització i analyzer. El tokenitzador ha de ser una funció que retorna les mateixes dades ja que en el nostre cas, ja estan tokenitzades i l'analyzer únicament fica el text en una llista:

```
def tokenize(dades):
    return dades
def character_analyzer(text):
    return list(text)
```

Seguidement hem creat la funció `CountVector`, que ajusta el vectoritzador als documents de training i després els aplica als de test. La funció retorna les matrius de dades de train, les de test i els noms de les columnes, que serien les paraules.

```
def CountVector(docs, docs_tst, stop_w, prepro=r'(?u)\b\w\w+\b'):
    # crear un objeto CountVectorizer
    vectorizer = CountVectorizer(stop_words=stop_w, min_df=0.15, max_df=0.85,
token_pattern=prepro, preprocessor=None, analyzer=character_analyzer)
    """ docs = [' '.join(words) for words in dades]
    docs_tst =[' '.join(words) for words in test]"""
    # ajustar el vectorizador a los documentos y transformar los documentos en
una matriz de recuento de palabras
```

```
    X = vectorizer.fit_transform(docs)
    T = vectorizer.transform(docs_tst)
    # imprimir la matriz de recuento de palabras
    columnes = vectorizer.get_feature_names_out()
    sol = X.toarray()  # passa de
    sol_t = T.toarray()
    # imprimir el vocabulario
    return sol, sol_t, columnes
```

Per aplicar la funció a totes les dades, utilitzarem la funció `treure_dataframess` que, a partir del train i test positius i negatius i el tipus d'stop words utilitzats, retornarà les dades de train i test en format dataframe de pandas i també una llista amb els noms de les columnes.

```
def treure_dataframes(neg,pos,test_neg,test_pos,stop_w):
    dades = neg+pos
    test = test_neg+test_pos
    dad,ts,cols = CountVector(dades,test,stop_w)

    #obtenir train
    train = pd.DataFrame(dad, columns=cols)
    etiquetes = ['neg'] * len(neg) + ['pos'] * len(pos)
    train['polaritat'] = etiquetes

    #obtenir test
    testing = pd.DataFrame(ts,columns=cols)
    etiquetes_test = ['neg'] * len(test_neg) + ['pos'] * len(test_pos)
    testing['polaritat'] = etiquetes_test

    return train,testing,cols
```

Apliquem aquesta funció a les dades amb els dos tipus de preprocessat diferents i amb les tres llistes de stopwords diferents. Així doncs, tindríem 6 tipus de dades diferents. A continuació, les comparem per veure si hem obtingut resultats molt diferents. La funció `dataframes_iguales` retorna una llista de conjunts on els elements dins un mateix conjunt representen que els dataframes als quals es refereix són iguals.

```
def dataframes_iguales(lista_dataframes):
    iguales = []
    visitados = set()

    for i, df1 in enumerate(lista_dataframes):
        if i in visitados:
            continue

        iguales_temp = {i}
        for j, df2 in enumerate(lista_dataframes[i + 1:]):
```

```
            if df1.equals(df2):
                iguales_temp.add(i + j + 1)
                visitados.add(i + j + 1)

        if len(iguales_temp) > 1:
            iguales.append(iguales_temp)

    return iguales
```

Aplicant aquesta funció obtenim: [{0,1,2},{3,4,5}]. Així podem concloure que la utilització de diferents llistes de stopwords no influeix en la preparació de les dades amb el CountVectorizer ja que veiem que els tres dataframes amb mateix preprocessat i diferents llistes de stopwords ens surten com a dataframes iguals. En canvi aquests 3 difereixen amb els tres que tenen aplicat l'altre tipus de preprocessat que inclou el tractament de l'apòstrof. Per tant, finalment continuarem únicament amb 2 dataframes: train_a0 i train_a3.

3.3 ENTRENAMENT DE MODELS

En aquest apartat, procedirem a la construcció i entrenament dels models supervisats per a la classificació de polaritat de les opinions de pel·lícules. Utilitzarem les dades preprocessades i transformades amb les tècniques descrites en els apartats anteriors, incloent el preprocesament i l'aplicació del CountVectorizer.

Per trobar el millor model possible, realitzarem un "grid search", que consisteix en explorar una combinació de paràmetres i algoritmes per determinar quina configuració ofereix la millor precisió en la predicció de polaritat. Això ens permetrà triar el model més adequat per a la tasca i optimitzar el seu rendiment.

Els apartats següents detallaran el procés de "grid search", la tria del model guanyador i l'anàlisi dels resultats obtinguts.

3.3.1 GRID SEARCH

En aquest apartat, implementarem el grid search per explorar les diferents combinacions de paràmetres i algoritmes en la cerca del millor model per a la classificació de polaritat de les opinions de pel·lícules. Abans de començar, utilitzarem la funció `desordenar` per barrejar les dades de train de manera aleatòria, assegurant-nos que no hi hagi cap biaix en l'ordre de les dades.

```
def desordenar(dades):
    # desordenar las filas en train
    filas_desordenadas = np.random.permutation(dades.index)
    desordre = dades.reindex(index=filas_desordenadas)
    return desordre
```

A continuació, hem preparat el grid search mitjançant la funció `grid_search` i hem definit els paràmetres de cada model que volem provar (svm_param_grid, rf_param_grid, logreg_param_grid, knn_param_grid).

```
def grid_search(train, model, param_grid, model_name):
    X_train = train.drop('polaritat', axis=1)
    y_train = train['polaritat']

    grid_search = GridSearchCV(model, param_grid, scoring='accuracy', cv=5,
return_train_score=True, n_jobs=-1)

    start_time = timeit.default_timer()
    grid_search.fit(X_train, y_train)
    elapsed_time = timeit.default_timer() - start_time

    return grid_search.cv_results_, elapsed_time
```

Finalment, amb la funció `run_grid_search`, executem el grid search per a cada conjunt de dades i model, obtinguent els resultats en un DataFrame que inclou la puntuació mitjana de validació creuada, la desviació estàndard de la puntuació i el temps transcorregut per a cada combinació de paràmetres.

```
def run_grid_search(datasets, models, grid_search):
    resultados = []

    for dataset_name, dataset_list in datasets.items():
        for dataset in dataset_list:
            for model, param_grid, model_name in models:
                cv_results, elapsed_time_total = grid_search(dataset, model,
param_grid, model_name)
                for mean_score, std_score, params, elapsed_time in
zip(cv_results["mean_test_score"], cv_results["std_test_score"],
cv_results["params"], cv_results["mean_fit_time"]):
                    resultados_temp = pd.DataFrame({
                        "Dataset": [dataset_name],
                        "Model": [model_name],
                        "Params": [params],
                        "Mean CV Score": [mean_score],
                        "Std CV Score": [std_score],
                        "Elapsed Time": [elapsed_time]
                    })
                    resultados.append(resultados_temp)
```

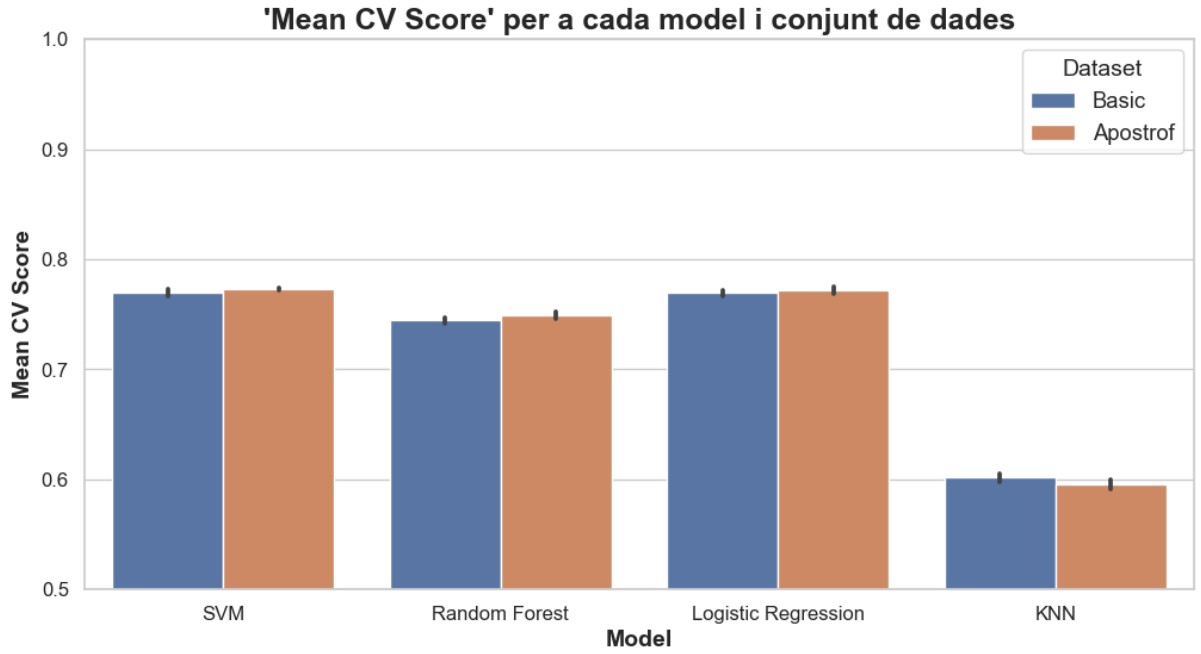
```
    # Concatenar los resultados en un solo DataFrame
    resultados = pd.concat(resultados, ignore_index=True)

    # Redondear los valores numéricos a 4 decimales
    resultados["Mean CV Score"] = resultados["Mean CV Score"].round(4)
    resultados["Std CV Score"] = resultados["Std CV Score"].round(4)
    resultados["Elapsed Time"] = resultados["Elapsed Time"].round(4)

    return resultados
```

Els resultats obtinguts ens permetran analitzar quin model i configuració ofereixen el millor rendiment en la predicció de polaritat de les opinions de pel·lícules.

3.3.2 TRIA DE MODEL

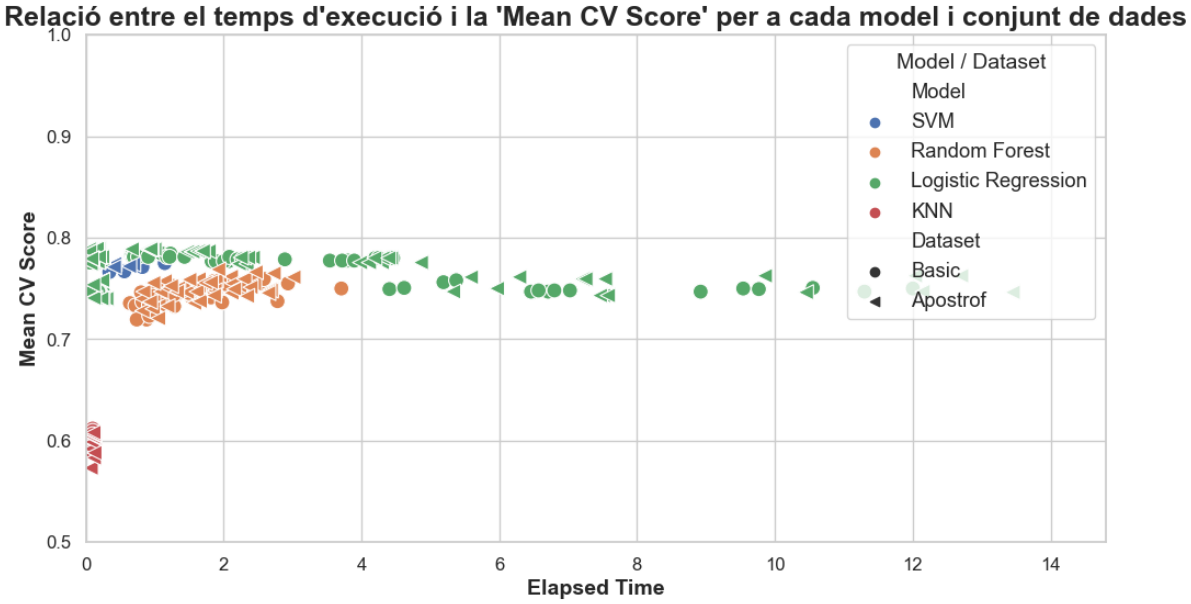
En aquest apartat, triarem el millor model basant-nos en els resultats obtinguts en l'apartat anterior. Per fer-ho, utilitzarem gràfics per comparar el rendiment i el temps d'execució de cada model.

En primer lloc, presentem un gràfic de barres que mostra la puntuació mitjana de validació creuada (Mean CV Score) per a cada model i conjunt de dades. En aquest gràfic, podem veure que els datasets amb apòstrof aconsegueixen millors resultats en tots els models excepte en KNN, tot i que la diferència és molt petita.



En segon lloc, presentem un gràfic de dispersió que mostra la relació entre el temps d'execució i la puntuació mitjana de validació creuada per a cada model i conjunt de dades.

Observem que el model de Regressió Logística té el temps d'execució més elevat, tant amb el dataset "bàsic" com amb el d'apòstrof. En canvi, KNN és el més ràpid, però té la puntuació més baixa.



Per triar el millor model de manera més precisa que amb les gràfiques, hem seguit els següents passos:

En primer lloc, hem seleccionat els millors models que representen el quantil 0,98 dels resultats, per obtenir una llista dels models amb les puntuacions més altes.

	Dataset	Dataset Type	Model	Params	Mean CV Score	Std CV Score	Elapsed Time
190	Apostrof	Desordenat	Logistic Regression	{'C':0.01, 'max_iter': 1200, 'solver': 'newton-cg'}	0.7894	0.0120	0.1817
194	Apostrof	Desordenat	Logistic Regression	{'C':0.01, 'max_iter': 1200, 'solver': 'newton-cg'}	0.7894	0.0120	0.1723
198	Apostrof	Desordenat	Logistic Regression	{'C':0.01, 'max_iter': 600, 'solver': 'newton-cg'}	0.7894	0.0120	0.1533
202	Apostrof	Desordenat	Logistic Regression	{'C':0.01, 'max_iter': 1200, 'solver': 'newton-cg'}	0.7894	0.0120	0.1566

272	Apostrof	Desordenat	Logistic Regression	{'C':0.01, 'max_iter': 1800, 'solver': 'newton-cg'}	0.7894	0.0120	0.1877
276	Apostrof	Desordenat	Logistic Regression	{'C':0.01, 'max_iter': 2400, 'solver': 'newton-cg'}	0.7894	0.0120	0.1784
280	Apostrof	Desordenat	Logistic Regression	{'C':0.01, 'max_iter': 1200, 'solver': 'newton-cg'}	0.7894	0.0120	0.1598
284	Apostrof	Desordenat	Logistic Regression	{'C':0.01, 'max_iter': 1200, 'solver': 'newton-cg'}	0.7894	0.0120	0.1597

Finalment, ja que els Mean CV Scores eren molt similars, hem triat el model més ràpid d'aquesta llista, que ens proporcionarà un bon rendiment i un temps d'execució raonable.

	Dataset	Dataset Type	Model	Params	Mean CV Score	Std CV Score	Elapsed Time
198	Apostrof	Desordenat	Logistic Regression	{'C':0.01, 'max_iter': 600, 'solver': 'newton-cg'}	0.7894	0.0120	0.1533

Així, tenim el millor model i el conjunt de dades seleccionats, que utilitzarem per a la classificació de polaritat de les opinions de pel·lícules.

3.4 ANÀLISI DE RESULTATS

En aquest apartat, analitzarem els resultats obtinguts amb el model seleccionat. Recordem que el model més òptim que hem triat és el següent: {Dataset: Apòstrof, Dataset Type: Desordenat, Model: Logistic Regression, Params: {'C': 0.01, 'max_iter': 600, 'solver': 'newton-cg'}}.

Per fer aquesta anàlisi, utilitzarem la funció `testejar` per entrenar el model seleccionat sense fer validació creuada i aplicar-lo al nostre conjunt de dades de prova. A continuació, obtindrem la matriu de confusió i les puntuacions d'exactitud.

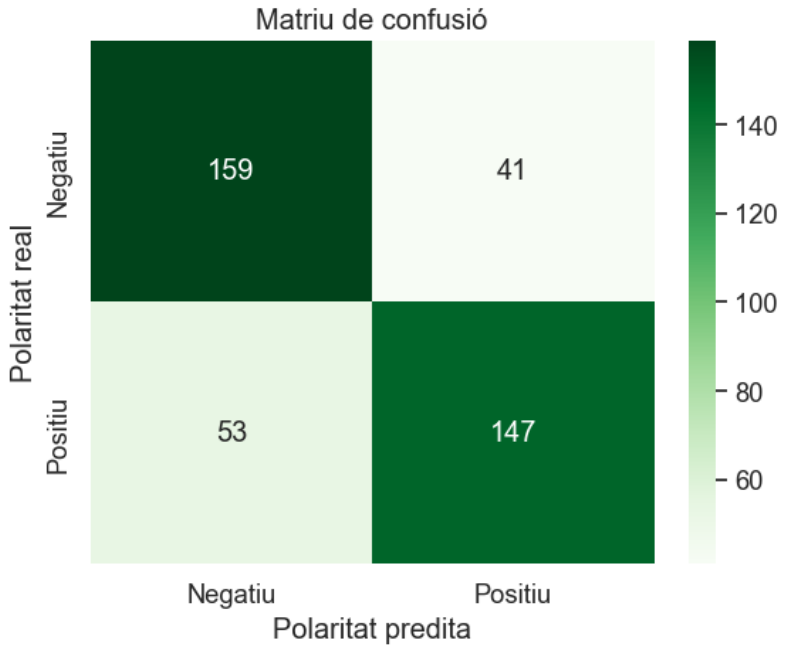
```python
def testejar(train,test,model):
    inici = timeit.default_timer()
    #crear el model de random forest
    X_train = train.drop('polaritat',axis=1)
    y_train = train['polaritat']
    model.fit(X_train, y_train)
    X_test = test.drop('polaritat',axis=1)
    y_test = test['polaritat']
    y_pred = model.predict(X_test)
    matriu_conf = confusion_matrix(y_test, y_pred)
    report = classification_report(y_test, y_pred)
    accuracy = accuracy_score(y_test, y_pred)
    # Evaluar el rendimiento del modelo
    print("Confusion Matrix:\n", matriu_conf)
    print("Classification Report:\n", report)
    print("Accuracy Score: ", accuracy)
    return matriu_conf
```

Després d'executar la funció "testejar", analitzem el "Classification Report" obtingut, que ens mostra les puntuacions de precisió, exhaustivitat (recall) i puntuació F1 per a cada classe, així com la mitjana ponderada d'aquestes puntuacions.

	precision	recall	f1-score	support
neg	0.75	0.80	0.77	200
pos	0.78	0.73	0.76	200
accuracy			0.77	400
macro avg	0.77	0.77	0.76	400
weighted avg	0.77	0.77	0.76	400

A partir d'aquest informe, podem observar que el nostre model té una precisió i exhaustivitat similars per a les classes "neg" i "pos", amb una puntuació F1 lleugerament superior per a la classe "neg". L'exactitud total del model és del 77%.

Per visualitzar millor els resultats, es pot crear una gràfica de la matriu de confusió per veure la distribució de les prediccions del model en comparació amb les etiquetes reals.



En resum, el nostre model de Regressió Logística seleccionat presenta un rendiment raonable per a la classificació de polaritat de les opinions de pel·lícules, amb una exactitud del 77% en el conjunt de dades de prova.

4. MODEL NO SUPERVISAT

En aquesta segona part del treball, proposem una alternativa per a la detecció de polaritat en textos d'opinió. A diferència de l'enfocament supervisat, aquesta proposta es basa en un model no supervisat que utilitza les puntuacions positives, negatives i objectives extretes de SentiWordnet. Per calcular aquestes puntuacions, primer hem de determinar la categoria gramatical de cada paraula per tal d'obtenir el seu synset corresponent utilitzant l'algoritme Lesk de NLTK.

En comptes de fer servir algoritmes supervisats per a l'entrenament del model, aquest enfocament utilitza eines lingüístiques per analitzar els textos d'opinió. Examinem diverses categories gramaticals per a l'anàlisi i suggerim diferents estratègies per combinar les puntuacions obtingudes, tenint en compte les possibles limitacions, com l'absència de synsets a SentiWordnet. L'objectiu principal és comparar el rendiment d'aquest model no supervisat amb el del model supervisat, per això, utilitzem les mateixes dades de test que en l'apartat A.

4.1 PREPROCESING

En aquesta secció, realitzarem el preprocessament de les dades per al model no supervisat. Aquest preprocessament inclou l'obtenció de dades específiques (5.1.1) i la neteja de caràcters no desitjats (5.1.2). El principal objectiu d'aquesta etapa és preparar les dades per a l'aplicació dels algoritmes de NLP. Tindrem només una versió de preprocés i és l'òptima segons l'anàlisi realitzat sobre els apòstrofs.

Cal mencionar, que no realitzem una eliminació d'stopwords ja que seran útils per a obtenir la categoria gramatical de cada paraula. Les stopwords seran eliminades en el filtre de les categories gramaticals amb els diccionaris (5.2.2)

4.1.1 OBTENCIÓ DE DADES ESPECÍFIQUES

Per al model no supervisat, utilitzarem les mateixes dades de test que en l'apartat 4 (MODEL SUPERVISAT). No obstant això, per a l'anàlisi no supervisada, necessitem les dades estructurades en frases a més de paraules individuals. Per a aconseguir això, aplicarem la funció obtenir_frases, que obté les dades en llistes de frases corresponents a cada document:

```
def obtenir_frases(ids):
    lst = []
```

```
    for i in ids:
        lst.append(mr.sents(i))
    return(lst)
```

Aquesta funció ens permetrà treballar amb les dades de manera més adequada per a l'enfocament no supervisat, tenint en compte les relacions entre les paraules dins de cada frase. Les frases seran utilitzades com al context per a obtenir els synsets de les paraules (5.2.3)

4.1.2 CARÀCTERS

Un cop les dades estiguin organitzades en frases, procedirem a eliminar els caràcters innecessaris o no desitjats. La funció utilitzada, fa ús de l'espressió regular per eliminar únicament els espais en blancs que estan mal codificats i en el text apareixen com nbsp (non-breaking space). Els altres caràcters i signes de puntuació els eliminarem en el moment que extraiem les categories gramaticals (5.2.1).

```
def nomes_nbsp(text):
    lst = []
    for i in range(len(text)):
        lst.append([])
        for j in range(len(text[i])):
            x = re.sub(r'\bnbsp', '', text[i][j])
            if x != '':
                lst[i].append(x)
    return lst
```

A més d'això, realitzem un anàlisi sobre l'apòstrof per veure quina és la millor forma de tractar-lo.

4.1.3 ANÀLISI DE L'APÒSTROF

A continuació, es presenta una anàlisi de l'ús de l'apòstrof en la tokenització i el seu impacte en l'etiquetatge gramatical (POS tagging) amb la llibreria NLTK. Es descriuen cinc casos diferents per demostrar com afecta la forma de tractar l'apòstrof en la tokenització.

Aquest cas tracta la contracció "you've" com un sol token. El POS-tagger no pot identificar correctament el pronom i l'auxiliar separats i, en canvi, ho etiqueta com un substantiu plural (NNS), la qual cosa és incorrecta.

```
word = ["It", "is", "like", "you've", "witnessed", "movie", "magic", "."]
```

```
nltk.pos_tag(word)
>>>[('It', 'PRP'),
 ('is', 'VBZ'),
 ('like', 'IN'),
 ("you've", 'NNS'),
 ('witnessed', 'VBN'),
 ('movie', 'NN'),
 ('magic', 'NN'),
 ('.', '.')]
```

En aquest cas, l'apòstrof es separa com un token individual. Tot i que el POS-tagger reconeix correctament "you" com un pronom (PRP) i "ve" com un substantiu plural (NNS), la puntuació intermèdia (""") rep una etiqueta incorrecta com a cometes simples (``). Això fa que l'estructura de la frase sigui menys clara i menys útil per a l'anàlisi posterior.

```
word2 = ["It", "is", "like", "you", "'", "ve", "witnessed", "movie", "magic",
"."]
nltk.pos_tag(word2)
>>>[('It', 'PRP'),
 ('is', 'VBZ'),
 ('like', 'IN'),
 ('you', 'PRP'),
 ("'", "''"),
 ('ve', 'NNS'),
 ('witnessed', 'VBD'),
 ('movie', 'NN'),
 ('magic', 'NN'),
 ('.', '.')]
```

Aquí, es col·loca l'apòstrof al costat de "you". El POS-tagger etiqueta incorrectament "you'" com un substantiu (NN) i "ve" també com un substantiu (NN). Això resulta en una estructura gramatical incorrecta i poc útil per a l'anàlisi posterior.

```
word3 = ["It", "is", "like", "you'", "ve", "witnessed", "movie", "magic", "."]
nltk.pos_tag(word3)
>>>[('It', 'PRP'),
 ('is', 'VBZ'),
 ('like', 'IN'),
 ("you'", 'NN'),
 ('ve', 'NN'),
 ('witnessed', 'VBD'),
 ('movie', 'NN'),
 ('magic', 'NN'),
 ('.', '.')]
```

Aquest cas separa correctament el pronom "you" i la contracció "'ve". El POS-tagger identifica "you" com un pronom personal (PRP) i "'ve" com una forma auxiliar del verb "have" (VBP). L'assignació d'etiquetes POS és correcta i reflecteix adequadament l'estructura gramatical de la frase. Separant les contraccions en dos tokens, es facilita l'anàlisi de l'estructura de l'oració i la comprensió del seu significat.

```
word4 = ["It", "is", "like", "you","'ve", "witnessed", "movie", "magic", "."]
nltk.pos_tag(word4)
>>>[('It', 'PRP'),
 ('is', 'VBZ'),
 ('like', 'IN'),
 ('you', 'PRP'),
 ("'ve", 'VBP'),
 ('witnessed', 'VBN'),
 ('movie', 'NN'),
 ('magic', 'NN'),
 ('.', '.')]
```

En aquest cas, s'elimina l'apòstrof completament i es combinen "you" i "ve" en un sol token. El POS-tagger etiqueta "you" com un pronom personal (PRP) i "ve" com una forma auxiliar del verb "have" (VBP), però la contracció es perd en aquest procés. L'eliminació de l'apòstrof pot dificultar la identificació de les contraccions i disminuir la precisió de l'anàlisi gramatical.

```
word5 = ["It", "is", "like", "you","ve", "witnessed", "movie", "magic", "."]
nltk.pos_tag(word5)
>>>[('It', 'PRP'),
 ('is', 'VBZ'),
 ('like', 'IN'),
 ('you', 'PRP'),
 ('ve', 'VBP'),
 ('witnessed', 'JJ'),
 ('movie', 'NN'),
 ('magic', 'NN'),
 ('.', '.')]
```

Millora, Cas 4. En aquest cas, separas el pronom "you" i la contracció "'ve" (de "have") en dos tokens diferents. Això permet que el POS-tagger de NLTK identifiqui correctament el pronom i la contracció com a dues entitats separades i els assigni les etiquetes POS correctes.

Finalment apliquem ambdues funcions a les dades en paraules i també a les dades separades pre frases. Amb aquest preprocessament, les dades estan preparades per a l'extracció de

característiques i l'aplicació dels mètodes no supervisats en l'apartat 5.2 (SYNSETS) i posteriors.

4.2 SYNSETS

En aquest apartat, abordem l'obtenció dels synsets per a poder construir posteriorment el nostre model no supervisat per a la detecció de polaritat en els textos d'opinió. Un synset és un conjunt de sinònims que representen un concepte únic en el llenguatge. En el context de l'anàlisi de sentiments, aquests synsets ens permeten capturar el significat semàntic de les paraules, proporcionant una base per a la detecció de polaritat.

4.2.1 CATEGORIA GRAMATICAL

Per a l'extracció de synsets de SentiWordNet, és essencial conèixer la categoria gramatical de cada paraula. Aquesta informació és important perquè ens permet accedir als synsets apropiats segons el rol que té la paraula en el context del text. Per a obtenir aquesta informació, farem ús de la funció de pos_tag de NLTK.

En la funció `categories` , utilitzem el pos (part-of-speech) tagging de NLTK que, donat un context (una llista de paraules) i una paraula d'aquest context, retorna una tupla que conté la paraula i la seva categoria gramatical associada. D'aquesta manera, podem identificar la categoria gramatical de cada paraula en el nostre text, facilitant la posterior extracció dels synsets de SentiWordnet.

```
def categories(dades):
    lst = []
    for doc in dades:
        clas1 = nltk.pos_tag(doc)
        lst.append(clas1)
    return lst
```

4.2.2 DICCIONARIS

En aquest apartat es creen diccionaris destinats a modificar les etiquetes de les categories gramaticals obtingudes a partir del pos tagging de NLTK ja que Lesk requereix d'unes etiquetes diferents. Crearem diccionaris pertanyents als 3 conjunts de categories gramaticals diferents que volem comparar: només adjectius; adjectius, adverbis i noms; i adjectius, adverbis, noms i verbs.

Primer de tot hem definit una funció per a crear un diccionari per a cada categoria gramatical:

```
def create_adjective_dict():
    d = {
        'JJ': 'a',
        'JJR': 'a',
        'JJS': 'a'
    }
    return d
def create_adverb_dict():
    d = {
        'RB': 'r',
        'RBR': 'r',
        'RBS': 'r',
        'WRB': 'r'
    }
    return d
def create_verb_dict():
    d = {
        'VB': 'v',
        'VBD': 'v',
        'VBG': 'v',
        'VBN': 'v',
        'VBP': 'v',
        'VBZ': 'v',
        'MD': 'v'
    }
    return d
def create_nom_dict():
    n = {
        'NN': 'n',
        'NNS': 'n',
        'NNP': 'n',
        'NNPS': 'n'
    }
    return n
```

Les claus dels diccionaris representen les etiquetes de les categories gramaticals obtingudes amb NLTK i els valors expressen la seva equivalència per a Lesk. Així doncs hem creat els diccionaris dels 3 conjunts diferents de categories gramaticals:

```
adjective_dict = create_adjective_dict()

adjective_adverb_dict=create_nom_dict()
adjective_adverb_dict.update(create_adjective_dict())
adjective_adverb_dict.update(create_adverb_dict())

adjective_adverb_verb_dict=create_nom_dict()
```

```
adjective_adverb_verb_dict.update(create_adjective_dict())
adjective_adverb_verb_dict.update(create_adverb_dict())
adjective_adverb_verb_dict.update(create_verb_dict())
```

Finalment hem aplicat la traducció dels diferents 3 diccionaris a les nostres dades amb la funció `traducir_lista_de_tuplas`. Aquesta reb les dades i el diccionari d'un dels 3 conjunts i retorna l'entrada amb les categories d'aquest conjunts traduïdes a les etiquetes de Lesk.

```
def traducir_lista_de_tuplas(lista_de_listas, diccionario):
    lista_traducida = []

    for sublista in lista_de_listas:
        sublista_traducida = []

        for tupla in sublista:
            palabra, etiqueta = tupla
            if etiqueta in diccionario:
                traduccion = diccionario[etiqueta]
                tupla_traducida = (palabra, traduccion)
                sublista_traducida.append(tupla_traducida)
            else:
                sublista_traducida.append(tupla)
        lista_traducida.append(sublista_traducida)

    return lista_traducida
```

A partir d'ara tindrem 3 conjunts de dades diferents, segons el conjunt de categories gramaticals elegides.

4.2.3 OBTENCIÓ

Finalment, procedim a l'obtenció dels synsets amb l'algorisme Lesk. Utilitzarem les dades agrupades per frases extretes a l'apartat 5.1.1 com a context per a veure quin és significat de cada paraula. La funció `treure_sinsets` reb les dades per paraules i la seva agrupació en frases i retorna únicament els synsets de les dades amb categoria gramatical vàlida (minúscula).

```
def treure_sinsets(dades,frases): # dades és un conjunt de documents
representats per llistes d'strings
    dad_sinsets =[]
    for i in range(len(dades)):
        doc = dades[i]
        doc_fr = frases[i]
        lens = []
```

```
        for f in range(len(doc_fr)):
            llarg = len(doc_fr[f])
            if not f == 0:
                llarg += lens[f-1]
            lens.append(llarg)
        z = 0
        j = 0
        sinsets = []
        while j < len(doc):
            while j < lens[z]:
                context = [paraula[0] for paraula in doc_fr[z]]
                # si ses frases tenim nomes paraules i no tuples, que es
probable:
                # context = doc_fr[z]
                if doc[j][1].islower():
                    sins = nltk.wsd.lesk(context,doc[j][0],doc[j][1])
                    sinsets.append(sins)
                j+=1
            z += 1
        dad_sinsets.append(sinsets)
    return dad_sinsets
```

Apliquem aquesta funció a les dades amb diferents categories gramaticals trauides i així obtenim 3 conjunts de dades amb els synsets de les categories corresponents gràcies al filtre de categoria gramatical en minúscula.

Fent algunes proves hem vist que per algunes paraules no es troben synsets, degut a que les dades de Lesk són limitades. Hem fet una visualització de la quantitat de Nones que apareixen. En mitjana de tots els documents trobem un 27.26% de Nones, cosa que no ens preocupa ja que és necessari reduir el nombre de paraules per al rendiment del model. Aquests Nones seran tractats en l'apartat següent.

4.3 MODELS

En aquest apartat, descriurem els models utilitzats en l'enfocament no supervisat per a la detecció de polaritat en els textos d'opinió. A diferència dels models supervisats, aquests models no requereixen un conjunt d'entrenament per aprendre a classificar les opinions com a positives o negatives. En lloc d'això, utilitzen informació lingüística, com ara els synsets de SentiWordNet, per analitzar el contingut dels textos. Els models descrits en aquest apartat són el resultat de la combinació de diferents estratègies, basades en les puntuacions de polaritat obtingudes dels synsets.

4.3.1 CREACIÓ

Per crear el model no supervisat, utilitzem la funció `graus`. Aquesta rep com a entrada un synset i retorna les puntuacions de positivitat, negativitat i objectivitat associades a aquest synset segons SentiWordNet. Aquestes puntuacions s'utilitzen per determinar la polaritat de cada paraula dins del text.

```
def graus(syn):
    synset = wn.synset(syn.name())
    # getting the sentiwordnet synset
    sentiSynset = swn.senti_synset(synset.name())
    grau = (sentiSynset.pos_score(), sentiSynset.neg_score(),
sentiSynset.obj_score())
    return grau
```

Per a combinar aquestes puntuacions positives, negatives i objectives utilitzarem diferents models. Aquests són algoritmes que analitzen un document i determinen si el seu contingut és positiu o negatiu. Hem creat 4 models diferents, cada un amb diferents ponderacions i formes de calcular la puntuació final per decidir si un text és positiu o negatiu.

Tots segueixen la mateixa estructura, només canvien en la forma de calcular la positivitat i la negativitat total del document. Tots els models reben com a entrada un document i utilitzen les puntuacions de polaritat obtingudes de la funció `graus` per determinar si el document és positiu o negatiu. Utilitzen diferents formes per a calcular la positivitat i negativitat del document sencer i retornen la funció retorna 1 si el document té una positivitat superior i 0 si té major negativitat.

El `model_1` augmenta el valor de positiu i negatiu per a cada synset amb una objectivitat inferior a 0,75. Se suma la positivitat del synset a la variable positiu i la negativitat a la variable negatiu.

```
def model_1(document):
    positiu=0
    negatiu=0
    for syn in document:
        if syn is None:
            pass
        else:
            positividad,negatividad,objetividad = graus(syn)
            if objetividad<0.75:
                negatiu+=negatividad
                positiu+=positividad
```

```
    if positiu>negatiu:
        #print("Es positiu")
        return 1

    elif positiu<negatiu:
        #print("Es negatiu")
        return 0

    else:
        return 1
```

El `model_2`, a diferència del primer model, suma positivitat o negativitat depenent de quin valor sigui més gran. Si la positivitat és major que la negativitat, s'incrementa el valor positiu; si la negativitat és major, s'incrementa el valor negatiu.

```
def model_2(document):
    positiu=0
    negatiu=0

    for syn in document:
        if syn is None:
            pass
        else:
            positividad,negatividad,objetividad = graus(syn)
            if positividad<negatividad:
                negatiu+=negatividad
            elif positividad>negatividad:
                positiu+=positividad

    if positiu>negatiu:
        #print("Es positiu")
        return 1
    elif positiu<negatiu:
        #print("Es negatiu")
        return 0
    else:
        #print("m'agraden les tetas")
        return 1
```

El `model_3` considera la subjetivitat (1 - objectivitat) per pesar la diferència entre positivitat i negativitat. Si la positivitat és major que la negativitat, s'incrementa el valor positiu ponderat per la subjetivitat; si la negativitat és major, s'incrementa el valor negatiu ponderat per la subjetivitat.

```
def model_3(document):
```

```
    positiu = 0
    negatiu = 0

    for syn in document:
        if syn is not None:
            positividad,negatividad,objetividad = graus(syn)
            subjetividad = 1 - objetividad

            if positividad < negatividad:
                negatiu += (negatividad - positividad) * subjetividad
            elif positividad > negatividad:
                positiu += (positividad - negatividad) * subjetividad

    if positiu > negatiu:
        return 1
    elif positiu < negatiu:
        return 0
    else:
        return 1
```

L'últim model, `model_4`, igual que el tercer, utilitza la subjetivitat per pesar la positivitat i negativitat, però en comptes de sumar les diferències com en el model 3, calcula la mitjana ponderada de la positivitat i negativitat. Si la mitjana ponderada de la positivitat és major que la de la negativitat, retorna 1 (positiu); en cas contrari, retorna 0 (negatiu).

```
def model_4(document): #document que és una llista de sinsets
    positiu = 0
    negatiu = 0
    total_subjetividad = 0

    for syn in document:
        if syn is not None:
            positividad, negatividad, objetividad = graus(syn)
            subjetividad = 1 - objetividad
            total_subjetividad += subjetividad

            positiu += positividad * subjetividad
            negatiu += negatividad * subjetividad

    if total_subjetividad == 0:
        return 1  # Si no hay subjetividad en el texto, devuelve 1 por defecto

    positiu /= total_subjetividad
    negatiu /= total_subjetividad

    if positiu > negatiu:
        return 1
    else:
```

```
        return 0
```

En resum el `model_1` no pondera els valors de positivitat i negativitat, el `model_2` compara directament la positivitat i negativitat per sumar-les, el `model_3` utilitza la subjetivitat per pesar les diferències entre positivitat i negativitat i el `model_4` calcula la mitjana ponderada de la positivitat i negativitat utilitzant la subjetivitat.

Per a l'aplicació d'aquests models necessitem una funció `entrenar_validar`. Aquesta és la responsable d'avaluar el rendiment del model. Rep com a entrada el model i les dades de test, i calcula la matriu de confusió, l'informe de classificació, la precisió i el temps d'execució del model. Aquesta informació és útil per comparar el rendiment dels diferents models.

```
def entrenar_validar(model,dades):
    inici = timeit.default_timer()
    resultats = []
    etiquetes = 200*[0] + 200* [1]
    for doc in dades:
        resultats.append(model(doc))
    matriu_conf = confusion_matrix(etiquetes, resultats)
    report = classification_report(etiquetes, resultats)
    accuracy = accuracy_score(etiquetes, resultats)
    final = timeit.default_timer()
    temps = final - inici
    # Evaluar el rendimiento del modelo
    print("Confusion Matrix:\n", matriu_conf)
    print("Classification Report:\n", report)
    print("Accuracy Score: ", accuracy)
    print("Temps d'execució:", temps)
    return accuracy,temps,matriu_conf
```

4.3.2 TRIA DE MODEL

L'objectiu d'aquest apartat és comparar el rendiment i el temps d'execució dels diferents models de classificació de text per determinar quin model ofereix una millor combinació de precisió i eficiència. A més, també es pretén analitzar com la selecció de diferents categories gramaticals de les paraules afecten el rendiment dels models.

```
models = [model_1, model_2, model_3, model_4]
dad = [sinsets_a,sinsets_a_r_n,sinsets_a_r_n_v]
cat = ['adjectius','adj + adv + nom','adj + adv + nom + verb']
matrius = []
resultats = []
for model in models:
```

```
    for i in range(len(dad)):
        acc, temp,matr = entrenar_validar(model,dad[i])
        resultats.append([model.__name__,cat[i],acc,temp])
        matrius.append(matr)
```

En el codi utilitzat, apliquem la funció `entrenar_validar` per a cada un dels possibles models i cada un dels possibles conjunts de categories gramaticals. Amb això extraiem els resultats de totes les combinacions de models i categories i les emmagatzemam en un dataframe.

Per obtenir el millor model comencem per agafar el quartil 0.7 de l'accuracy.

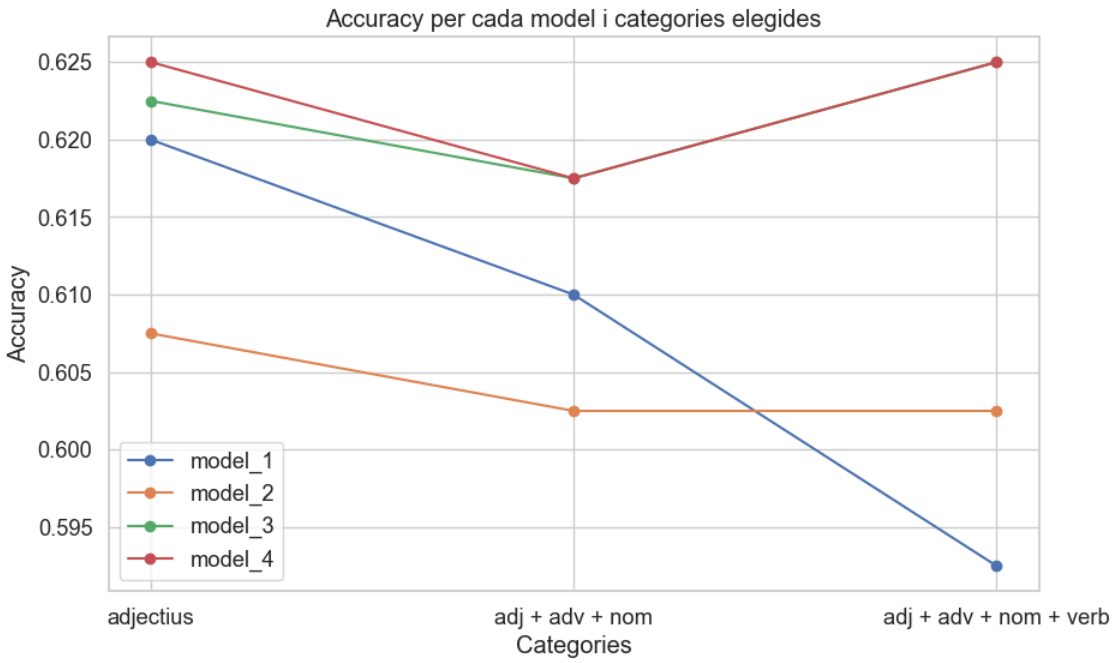
	Model	Categories	Accuracy	Temps d'execució
6	model_3	adjectius	0.6225	0.289425
8	model_3	adj + adv + nom + verb	0.6250	1.976858
9	model_4	adjectius	0.6250	0.185227
11	model_4	adj + adv + nom + verb	0.6250	2.373888

Per aquests models, elegirem el que té el temps d'execució més baix:

	Model	Categories	Accuracy	Temps d'execució
9	model_4	adjectius	0.6250	0.185227

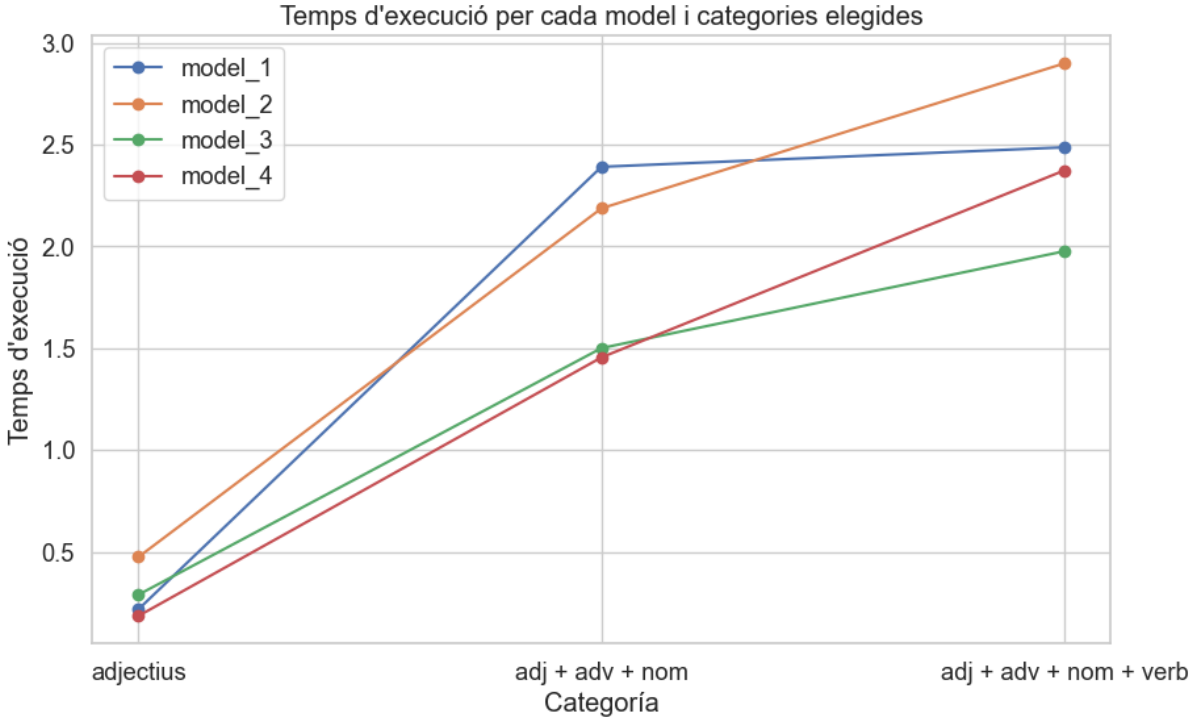
Així doncs el millor model de tots els models no supervisats entrenats ha sigut el model 4 incloent únicament els adjectius.

Per a reafirmar l'elecció del nostre model visualitzarem dos gràfics. Ambdós tenen representades 4 línies, un per cada model i a l'eix y trobem tres etiquetes; una per cada conjunt de categories gramaticals utilitzades.



Aquest primer plot plasma les accuracies de tots els models diferents. Veiem que els millors són edel model 4 i del 3. El conjunt de categories amb únicament adjectius funciona molt bé pel 4 i una mica menys pel 3. El model amb el conjunt de les 4 categories funciona igual de bé per aquests dos millors models.

El rendiment baixa quan fem ús del conjunt de 3 categories en tots els models i el de 4 categories només disminueix, i de forma molt pronunciada, en el model 1.

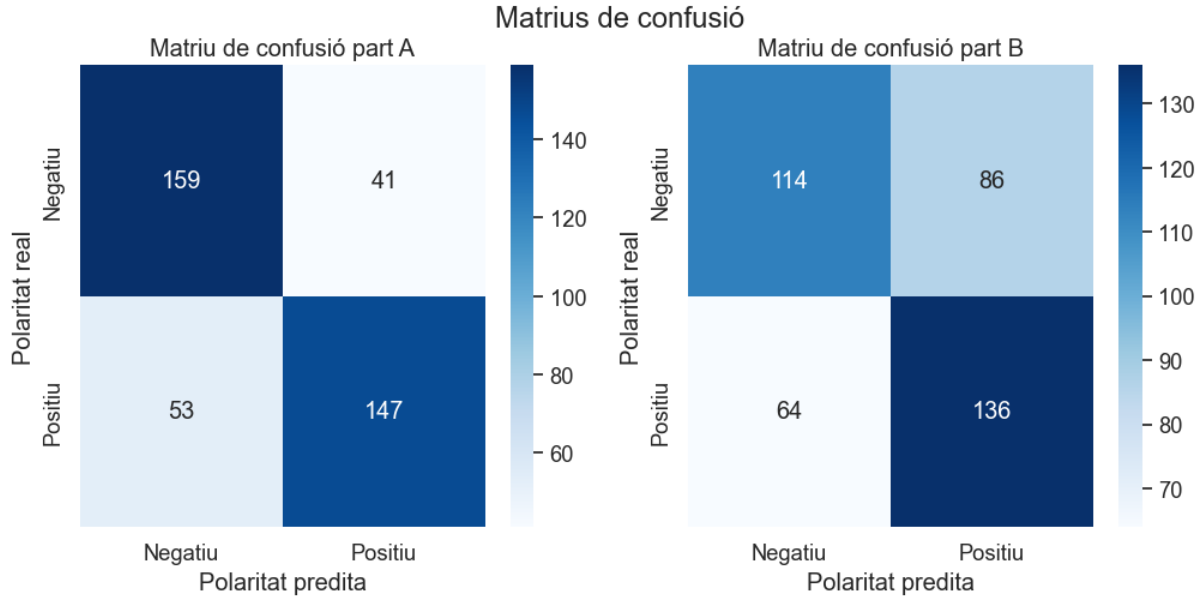


Podem veure una notòria diferència entre els temps d'execució entre els diferents conjunts de categories. Això té molta lògica ja que el conjunt de dades que s'ha d'executar és més gran quantes més categories s'inclouen.

Com hem vist que el conjunt de categories més eficient és només adjectius, que també coincideix amb un dels millors models en quant a accuracy confirmem que el model obtingut en l'anàlisi dels dataframes és la millor opció.

5. COMPARACIÓ

En aquest apartat, hem realitzat una comparació entre els models dels apartats A (supervisat) i B (no supervisat) per determinar quin d'aquests ha obtingut millors resultats. Per fer això, hem utilitzat una matriu de confusió que ens permet observar i analitzar el rendiment de cada model.



A partir d'aquests resultats, podem concloure que el model supervisat ha obtingut un millor rendiment en comparació amb el model no supervisat. Això es pot observar en el nombre més gran de prediccions correctes en ambdues categories (positiva i negativa) per al model supervisat. Aquesta diferència en el rendiment pot ser deguda a la forma en què cada model ha estat entrenat i la seva capacitat per generalitzar a partir de les característiques del text.

En resum, encara que tant els models supervisats com els no supervisats han demostrat ser útils per a la classificació de les opinions, el model supervisat sembla ser més precís i efectiu en aquesta tasca.

5. 1 PROVA LA TEVA PRÒPIA OPINIÓ

En aquest apartat, hem creat una petita interfície que permet a l'usuari introduir la seva pròpia opinió i seleccionar quin dels dos millors models (supervisat o no supervisat) utilitzar per analitzar si aquesta opinió és positiva o negativa.

Aquesta funcionalitat permet als usuaris comparar els resultats obtinguts amb cada model i determinar quin model s'ajusta millor a les seves necessitats. A partir de les proves realitzades, hem observat que el model No Supervisat funciona millor amb textos més curts que el Supervisat. No obstant això, aquesta conclusió pot variar depenent del tipus de text i les preferències de l'usuari.

Per implementar aquesta interfície, hem fet ús de la llibreria "IPython.display". A partir de les proves realitzades amb opinions pròpies, hem observat que el model no supervisat funciona millor amb textos curts en comparació amb el model supervisat. Això podria ser degut a la diferència en la manera com els dos models han estat entrenats i la seva capacitat per generalitzar a partir de les característiques del text.

6. CONCLUSIONS

En resum, aquest treball presenta un estudi exhaustiu sobre la detecció de polaritat en opinions de pel·lícules utilitzant tècniques de processament del llenguatge natural. Hem desenvolupat dos enfocaments diferents: un model supervisat i un model no supervisat.

Per al model supervisat, hem analitzat diferents preprocesos de caràcters i l'ús de stopwords, i hem aplicat una cerca de quadrícula per triar el millor model entre SVM, Regressió Logística, Random Forest i KNN. El model escollit, basat en la Regressió Logística, ha obtingut bons resultats en la classificació de polaritat. Hem observat que el tractament de l'apòstrof en el preprocessament ha millorat lleugerament els resultats.

Per al model no supervisat, hem utilitzat SentiWordNet per a l'anàlisi de sentiments i l'algoritme Lesk per obtenir els synsets corresponents a cada paraula en el text. Hem comparat diferents conjunts de categories gramaticals per a l'anàlisi i suggerit diferents estratègies per combinar les puntuacions obtingudes.

Les conclusions principals d'aquest treball són les següents:

El preprocessament és un pas clau en la detecció de polaritat, i el tractament de l'apòstrof pot millorar lleugerament els resultats.

L'ús de models supervisats com la Regressió Logística ofereix bons resultats en la classificació de polaritat en opinions de pel·lícules.

El model no supervisat basat en SentiWordNet i l'algoritme Lesk proporciona una alternativa interessant per a la detecció de polaritat, encara que potser no tan potent com els models supervisats.

L'anàlisi de diferents categories gramaticals i estratègies de combinació de puntuacions en l'enfocament no supervisat pot influir en els resultats i pot ser útil per a la detecció de polaritat en altres tipus de textos d'opinió.

Aquest treball proporciona una base sòlida per a futures investigacions en la detecció de polaritat utilitzant tècniques de processament del llenguatge natural. Pot ser útil explorar altres models i enfocaments, així com aplicar aquestes tècniques a altres tipus de textos d'opinió o idiomes.