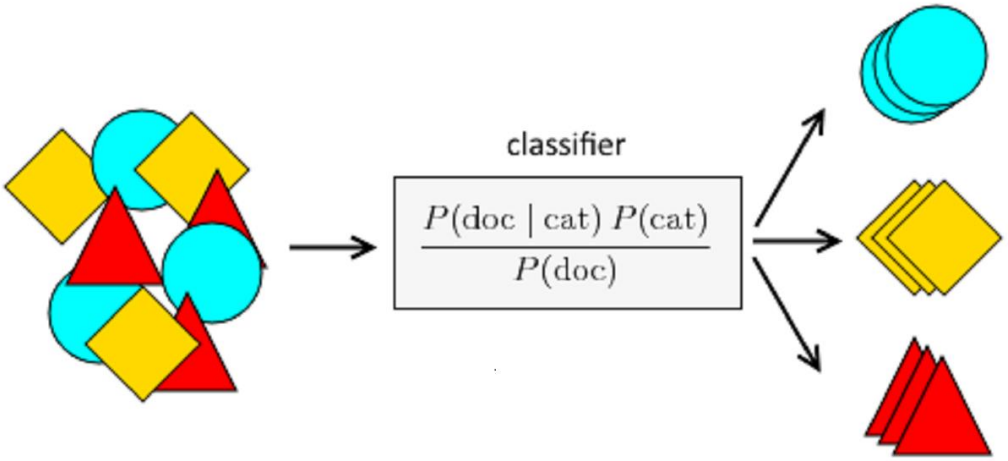
Coneixement, Raonament i Incertesa

Practica 3:

Naive Bayes



Juan Carlos Martínez Moreno

1566936

Índex

Introducció al problema

El problema que a solucionar en aquesta practica consisteix en crear un programa que sigui capaç de classificar tweets (missatges d'una xarxa social anomenada Twitter) en funció de si el tweet es positiu, es a dir, si el sentiment que vol transmetre aquest tweet es positiu, o si per contra el que vol transmetre es un sentiment negatiu.

Per fer la classificació, primer es necessita una base de dades prou gran com per donar-li al programa molts exemples i que aprengui de forma efectiva. El dataset que tenim conté, aproximadament, 1,6M de tweets que han sigut recollits de la xarxa social, i tots son en angles.

El programa es basarà en un model d'aprenentatge computacional que, donant-li un conjunt de tweets per entrenar, després serà capaç de classificar nous tweets que li arribin amb una certa probabilitat, a vegades molt alta, d'encertar. Aquest model classificador es diu classificador bayesià o Naive Bayes, i es basa bàsicament en la teorema de les probabilitats de Bayes per fer les classificacions.

$$P\ (A/B) = \frac{P(B/A) * P(A)}{P(B)}$$

Aquest teorema ens diu com podem saber la probabilitat d'un esdeveniment sabent que succeeix un altre event. Podem notar com aquest teorema també utilitza les probabilits condicionades.

En aquesta practica necessitem adaptar el classificador bayesià a un classificador de textos. El nostre classificador funcionarà de la seguent forma: l'entrenament del model el farem amb un conjunt de train, i per cada classe que existeixi a la base de dades hem de veure amb quina frequencia apareix cada paraula del conjunt de tweets de traint. En el nostre dataset existeixen 2 classes: 0 negatiu i 1 positiu. Per tant, classe binaria. Amb les freqüències d'aparició de cada paraula, el que aconseguirem més tard es tenir la probabilitat de cada paraula d'apareixer en qualsevol de les dues clases. Notem que, indirectament, s'estan separant els tweets segons si son de la classe 0 o de la classe 1, ja que la probabilitat de cada paraula s'ha de calcular per la classe 0 i després per la classe 1.

També necessitem calcular la probabilitat de que apareixi la classe 0 i la probabilitat de que apareixi la classe 1. Aquestes probabilitats son fàcils de calcular: simplement hem de contar quants tweets tenim de la classe 0 i dividir entre el total de tweets. El mateix amb la classe 1.

Un cop tenim les probabilitats de cada paraula diferent del conjunt de traint d'apareixer en qualsevol de les dues classes:

$$* P(w_k|v_j) \leftarrow \frac{n_k+1}{n+|Vocabulary|}$$

on nk es la frequencia d'aparició de la paraula i n es el numero de paraules totals que hi ha a la classe j (comptant repeticions, NO comptem paraules úniques) i també tenim la probabilitat d'aparicio de cada classe:

$$P(v_j) \leftarrow \frac{|docs_j|}{|Examples|}$$

Llavors ja tenim llest el diccionari amb cada paraula i les seves probabilitats i ja tenim entrenat el model.

Per fer la classificació de nous tweets que arriben al model, s'ha de fer una assumpció bastant irreal però que ens simplifica la tasca de classificar: **assumir que cada paraula d'un tweet nou es independent de la resta**.

Com aquest classificador es basa en probabilitats d'aparició de cada paraula, el que haurem de fer serà separar les paraules d'un tweet nou i veure quina probabilitat té cada paraula en cadascuna de les classes. En el nostre problema en concret, com nomes tenim 2 classes, haurem de veure les probabilitats de que una paraula qualsevol del tweet aparegui en la classe 0 i en la classe 1. Per tant, la probabilitat de que el tweet sigui de la classe x serà el màxim de la probabilitat de que el tweet sigui de la classe 0 i que sigui de la classe 1.

La probabilitat de que el tweet sigui d'una classe x es calcula multiplicant les probabilitats d'aparició de cada paraula del tweet en la classe x i al final multiplicant per la probabilitat d'aparició de la classe x. La formula seria aquesta:

$$v_{NB} = \underset{v_j \in V}{\operatorname{argmax}} P(v_j) \prod_{i \in positions} P(a_i|v_j)$$

On vj seria la classe x i a seria cadascuna de les paraules del tweet.

Proposta de solució al problema de classificació de tweets

El problema que volem tractar necessita algunes estructures de dades, mes o menys "simples", que ens permeten guardar la informació que necessitem en tot moment per fer la classificació. Les estructures de dades que es poden utilitzar son moltes i molt variades, però en aquest problema en concret podem utilitzar estructures molt simples com diccionaris, on podem indexar per la paraula que volguem i trobarem la informació ràpidament.

Per implementar el classificador de Naive Bayes he creat una classe anomenada **NaiveBayes** on guardaré els diccionaris i la informació numèrica que faci falta, a més de les funcions que ens calcularan tota aquesta informació. Els mètodes i els atributs que conté aquesta classe son (tots els atributs son públics):

- **total_words**: numero que conté totes les paraules que hi ha en el conjunt de tweets de train (comptant repeticions i tot). total_words = words_classN + words_classP
- **probability_classes**: tupla que guarda la probabilitat d'aparició de cada classe (tant 0 com 1).
- **words_classN**: numero que conté totes les paraules que hi ha en el conjunt de tweets de train **que pertanyen a la classe negativa (0)**
- **words_classP**: numero que conté totes les paraules que hi ha en el conjunt de tweets de train **que pertanyen a la classe positiva (1).**
- **smoothing**: valor que utilitzem per aplicar la tècnica de laplace smoothing.
- **probabilities_classN:** diccionari que conté les freqüències d'aparició de cada paraula dins el conjunt de tweets que pertanyen a la classe negativa.
- **probabilities_classP:** diccionari que conté les freqüències d'aparició de cada paraula dins el conjunt de tweets que pertanyen a la classe positiva.

Al separar un possible diccionari que podria contenir les freqüències de les dues classes en una tupla per cada entrada, si el separem en dos diccionaris, un per cada classe, podem utilitzar els diccionaris **Counter()** de la llibreria **collections**, que estan especialitzats en comptar objectes que anem ficant al diccionari amb el mètode Update(). Cada cop que llegim un tweet del conjunt de traint, podem directament separar les paraules i ficar-les al diccionari amb el mètode Update(), i el diccionari ja afegeix el recompte automàticament, cosa que ens facilita a nosaltres el treball i el codi es veu més clar.

Els mètodes que tenim a la classe son:

- **fit(x,y,size_dict = -1) (públic):** funció que rep el conjunt de traint dividit en un numpy array x (els tweets) i un numpy array y (les classes de cada tweet). Opcionalment podem passar el tamany del diccionari que volem, i ens pot servir per limitar la longitud d'aquest. Aquesta funció crida a una funció privada que entrena el model creant els 2 diccionaris de freqüències i calculant els altres atributs (total de paraules, paraules classe N, paraules classe P).

- **predict(x) (públic):** funció que classifica els tweets que es passen en el array numpy x. Utilitza els diccionaris creats en l'entrenament per fer la classificació, i retorna una llista amb les classificacions.
- **__limit_size_dict(n) (private):** funció que ens retalla el tamany del diccionari original creat al entrenament. Agafa les n entrades amb més freqüència de cadascun dels dos diccionaris, i crea 2 nous diccionaris amb tamany n.

També necessitem una sèrie de funcions auxiliars que ens ajudin a preparar les dades per poder entrenar el model, i, clarament, funcions per evaluar el model.

Les funcions implementades son:

- **train_test_split(x, y, partició):** funció que ens divideix el dataset original en un conjunt per entrenar i altre per validar el model. La funció rep el dataset original dividit en un array numpy x amb els tweets i un array numpy y amb les classes dels tweets, i per ultim el percentatge que volem que pertanyi al conjunt de train. Exemple: si volem que el 80% del dataset original sigui per entrenar, llavors passarem per parametre 0.8.
- **kfold(dataset, cv):** funció que implementa la tècnica de validació creuada o cross-validation per evaluar el model. La funció rep el dataset original i el numero de particions que volem fer.
- **Test_one_model(x, y, smoothing, partition):** funció que prova un sol model amb el valor de laplace smoothing i la partició del conjunt de train i test que especifiquem per parametre, i mostra per pantalla les mètriques, la matriu de confusió i les corbes ROC i PR.

Problemes sorgits i propostes de solució

El procés de codificació del programa que utilitzo per solucionar el problema d'aquesta practica no ha sigut molt difícil, però si que he tingut uns petits problemes en un punt del procés.

Per fer la part de crear el diccionari de les freqüències d'aparició de cada paraula es pot fer amb un bucle for i separant les paraules de cada tweet i afegint-les manualment al diccionari, o també existeix la possibilitat d'utilitzar **numpy** per fer de forma automàtica la separació de les paraules (np.char.split()) de cada tweet, ajuntar totes aquestes paraules (np.concatenate() i np.ravel()) i després, amb la funció np.unique(), obtenir un array amb les paraules úniques i el recompte de cadascuna, i amb així podem crear els diccionaris en poques línies.

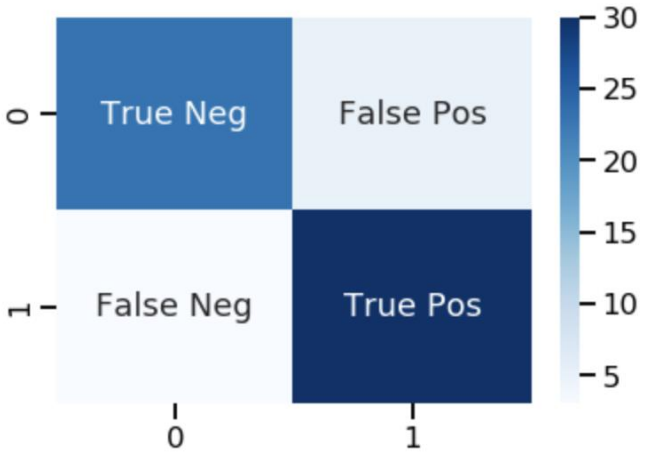
El principal problema que m'he trobat amb això es el temps d'execució, ja que triga molt en executar aquestes funcions, i per tant, he hagut d'implementar-ho manualment amb un bucle for, que triga molt menys. Crec que el temps elevat de numpy es degut a la gran quantitat de tweets que ha de separar les seves paraules, i després ajuntar totes aquestes paraules en un sol array.

Un altre petit problema seria el tema de les **probabilitats zero**. Quan a un tweet nou hi ha una paraula que no s'ha vist en l'entrenament, la probabilitat d'aparició d'aquesta paraula es 0, i per tant, tota la probabilitat del tweet es converteix en 0 (perquè es multiplica). Però, aquest problema es soluciona al exercici 3 amb laplace smoothing.

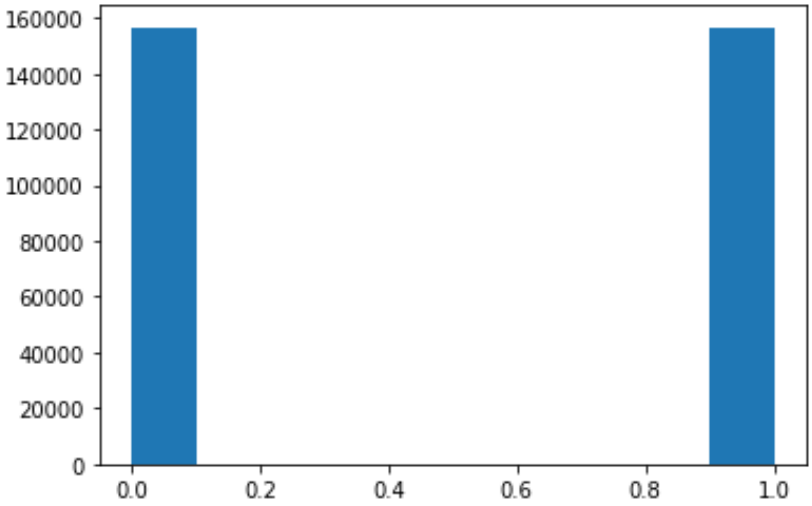
Per últim, un problema que podria sorgir seria l'arrosegament de moltíssims decimals, ja que les probabilitats que acabem tenint per cada paraula son molt petites, i si tenim un tweet amb moltes paraules, la multiplicació pot acabar acumulant molts decimals i hi ha risc de patir overflow. Una possible solució a aquest problema seria utilitzar **logaritmes** en comptes de probabilitats, i sumar logaritmes.

Resultats de la pràctica

En aquest apartat veurem els resultats que hem obtingut executant el nostre classificador Naive Bayes amb el dataset de tweets que tenim, i avaluarem el seu rendiment a partir de mètriques com per exemple, calcular l'accuracy, precision, recall, i veure matrius de confusió i corbes ROC i PR. Les nostres matrius de confusió tindran la següent forma:



Respecte a les mètriques utilitzades, ens podríem quedar nomes amb l'accuracy, ja que funciona perfectament amb conjunts de dades balancejats com el nostre, com es podrà veure al histograma que mostrem amb el numero de dades que tenim per cada classe (160k aprox per cada classe, observat sobre el conjunt de test!).



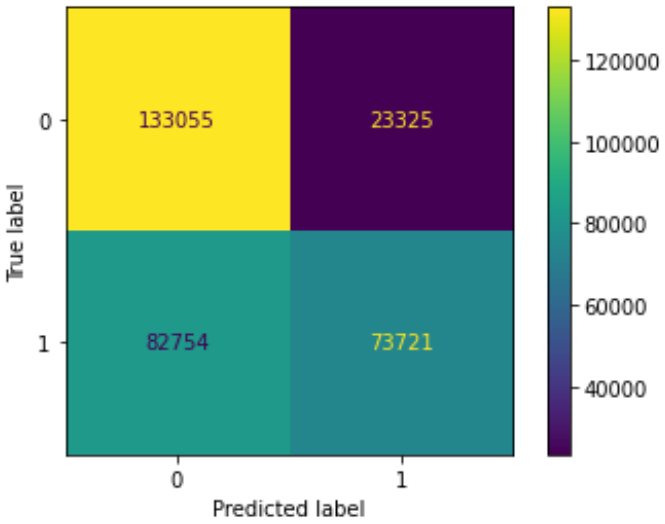
Exercici 1

En aquest exercici s'ha d'implementar el classificador Naive Bayes basic i fer proves amb el conjunt dividit en train i test i després validar-lo amb la tècnica de validació creuada.

Amb una divisió al atzar del conjunt en train i test, obtenim les següents dades:

Mètrica	Dades obtingudes
Accuracy	0.66
Precision	0.75
Recall	0.47

Obtenim els dos diccionaris, tant el que fa el recompte de les paraules que apareixen als tweets tant de la classe negativa com positiva, i el tamany es de: 341.512 paraules i 411.375 paraules, respectivament. Obtenim la següent matriu de confusió:



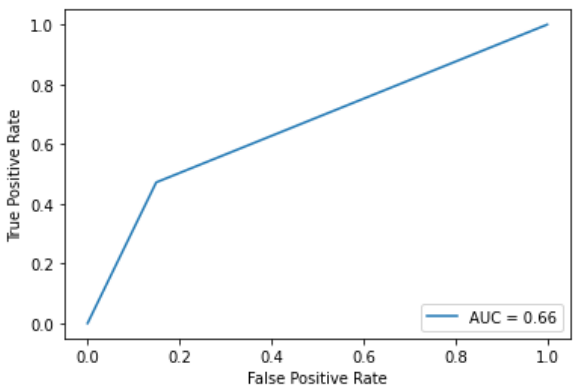
Com es pot observar a la matriu de confusió, moltes de les dades les classifica erròniament de la classe 0, es a dir, la classe negativa. El motiu d'això es el problema que hem comentat anteriorment de les probabilitats a 0. Quan no trobem una paraula al diccionari, automàticament la probabilitat es converteix a 0, i si això passa amb les dues classes, llavors la funció np.argmax() que agafa la probabilitat més gran agafa la primera posició per defecte, es a dir, la primera, ja que les dues probabilitats son iguals, zero.

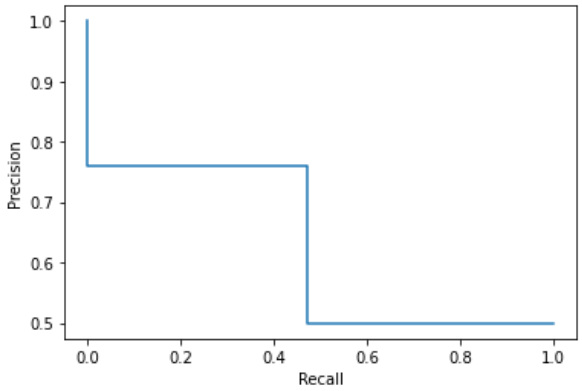
Ara anem a analitzar el resultat de l'execució del cross-validation, dividint el conjunt en 5 subconjunts (cv=5):

Iteració	Accuracy
1	0.6619
2	0.6603
3	0.6615
4	0.6613
5	0.6624
Mitjana de les iteracions	0.6615

El resultat del cross-validation es pràcticament igual que una sola execució, per tant, obtenir un 66% d'accuracy amb el problema de les probabilitats zero es un bon resultat.

A continuació mostrem les corbes ROC i Precision-Recall:





Exercici 2

A l'exercici 2 ens dedicarem a fer diferents proves amb el model, i aquestes proves consisteixen en anar modificant el tamany dels diccionaris i/o del conjunt de train, i veurem com evolucionen els resultats de rendiment quan anem fent les variacions.

Primer anirem ampliant el conjunt de train. Notarem que la mida dels dos diccionaris, a mesura que creix el conjunt de train, també creix la mida dels dos diccionaris. Això es degut a que quant més tweets agafem per entrenar, més paraules diferents podrem encontrar, i per tant, més paraules diferents tindrà el diccionari. En la següent taula es recullen els resultats obtinguts:

Partició de train	Mida diccionari classe 0	Mida diccionari classe 1	Accuracy
0.2	118803	143409	0.616
0.3	162727	195624	0.629
0.4	202346	244664	0.638
0.5	239549	290277	0.646
0.6	275484	331923	0.652
0.7	308796	372600	0.656
0.8	340833	411700	0.662

Els resultats mostren el que esperàvem, quant més gran es el conjunt de train, més paraules als diccionaris, i per tant, millors resultats.

Ara provarem a fixar un tamany pel conjunt de train, però anirem variant el tamany dels diccionaris. En la següent taula es mostren els resultats obtinguts:

Tamany dels dos diccionaris	Accuracy
20.000	0.59
50.000	0.61
100.000	0.63
200.000	0.64
300.000	0.65

El tamany del diccionari, i per tant, la varietat de paraules fa que la classificació sigui més o menys bona, segons es pot veure als resultats obtinguts. Quant més tamany tinguin els diccionaris, més varietat de paraules, i per tant, es classifiquen bé més tweets.

Per últim, fixarem un tamany pels dos diccionaris, de 100.000, i anirem variant el tamany del conjunt de train. He escollit un tamany de 100.000 ja que quan el tamany de train es de 0.2, el mínim, el tamany dels dos diccionaris està al voltant de 100.000 paraules, i per no tenir problemes, poso aquest numero. En la següent taula es mostren els resultats obtinguts:

Partició de train	Accuracy
0.2	0.607
0.3	0.613
0.4	0.619
0.5	0.624
0.6	0.628
0.7	0.632
0.8	0.636

Quan fixem la mida del diccionari però variem el conjunt de train, el que estem fent es entrenar amb més varietat de paraules. En el meu problema he fet que la mida del diccionari es limiti a les paraules més freqüents, i per tant, aquesta selecció de les paraules més freqüents fa que el rendiment augmenti una mica, però no es tant significatiu com variar el tamany del diccionari o el conjunt de train sense restringir el tamany del diccionari.

Exercici 3

En aquest exercici aplicarem la tècnica de laplace smoothing, que ens soluciona el problema de les probabilitats zero, comentat anteriorment. Aquest problema suposava una penalització en el rendiment bastant important, però ara, amb el laplace smoothing, podrem finalment prendre decisions en tots els tweets, ja que quan la probabilitat era 0 en les dues classes, agafàvem automatcament la classe 0. Ara, com no ens passarà mai que tinguem probabilitats zero, tenim mes possibilitats de que les probabilitats per una classe i per l'altra siguin diferents (ja que quan es multiplica un 0 per altres probabilitats, automàticament quedava 0. Ara ja no passa això).

La nova formula per calcular cadascuna de les probabilitats quedaria així:

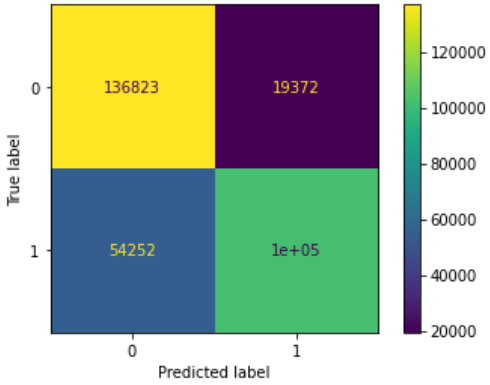
$$P(w'|positive) = \frac{\text{number of reviews with w' and y = positive } + \ \alpha}{\text{N } + \ \alpha * K}$$

on alfa es el valor del laplace smoothing, N es el numero de paraules de la classe n, i K hauria de ser el numero de paraules diferents que hi trobem a tots els tweets, però utilitzaré el numero total de paraules per no augmentar més el temps d'execució trobant el numero de paraules diferents.

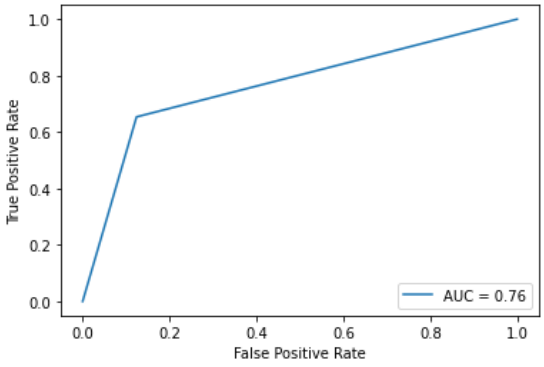
Si provem un valor de laplace smoothing, com per exemple **l = 1,** obtenim els següents resultats:

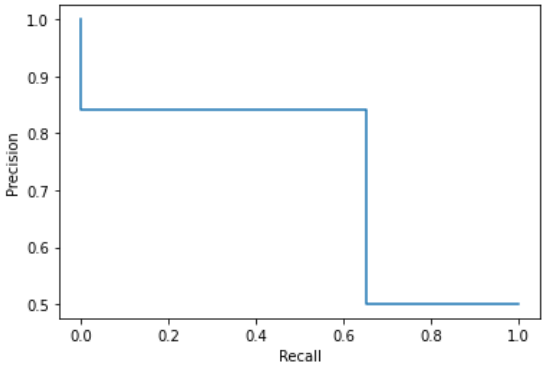
Mètrica	Dades obtingudes
Accuracy	0.76
Precision	0.84
Recall	0.65

Amb una matriu de confusió:



I les següents corbes ROC i PR:





Ara que hem aplicat el laplace smoothing amb l = 1, podem veure a la matriu de confusió com ha classificat correctament uns 236k tweets de 300k que hi ha, aproximadament. El accuracy ha pujat en un 10%, per tant, ha fet efecte i molt bó.

Conclusions

La resolució del problema proposat de la classificació de tweet ha estat, en major o menor mesura, assequible i no he tingut problemes molt complexos de resoldre. El classificador que hem implementat a la practica, el Naive Bayes, es un classificador força simple i amb un potencial i un rendiment bastant bo malgrat la simplicitat de la idea que segueix aquest model.

Sobretot es un bon classificador perquè es capaç de processar milions de tweets en segons, cosa que el fa molt ràpid i una molt bona opció quan no es compta amb moltíssims recursos. També dona resultats bastant bons malgrat la assumpció que fa de que les paraules son independents entre si. Això, a la realitat, no es veritat, però per simplicitat s'assumeix això, i a canvi d'arriscar resultats més bons, s'aconsegueix un classificador molt ràpid i simple que funciona perfectament per gent inexperta que comença en aquest mon de la IA.

Tot i que es podria millorar molt el model, considero que ha estat una practica molt satisfactòria, on he après molt i he posat en practica els coneixements adquirits en la teoria, i ara podem veure la part practica i el sentit que te tot això. També es milloren molt les tècniques de programació, ja que s'ha de buscar unes estructures de dades el més eficient possible per intentar minimitzar el temps d'execució al màxim, tot i que processar milions de tweets en segons, en sembla bastant interessant.