Pràctica 3 - CRI

Josep Maria Domingo Catafal - NIU 1599946

Contents

1 Introducció: explicació del problema.

La gent publica tweets constantment, i en molts casos aquests tweets reflecteixen l'estat d'ànim de la persona. Disposem d'una base de dades amb molts tweets ja classificats segons l'estat d'ànim que reflecteixen (positiu o negatiu), i l'objectiu és, a partir d'aquesta informació, entrenar un model, utilitzant ***Naive Bayes***, que permeti classificar futurs tweets.

El format del dataset es el següent:

```
ID   Contingut del tweet    Data         Sentiment (0 o 1)
-----------------------------------------------------------
16   I fell in love again   02/12/2015   1
```

2 Implementació

2.1 Lectura del dataset

Funció: `load_dataset`

El primer pas és llegir el dataset per poder treballar amb ell. Per fer-ho s'ha fet ús de la llibreria pandas. En aquest pas s'eliminen les columnes de ID i data, ja que no ens serveixen per res.

2.2 Divisió en conjunt de train i test

Funció: `train_test_split`

Un cop tenim les dades el següent pas es dividir en conjunt de train i test. Per aquest pas també s'ha utilitzat padas. Per crear el conjunt d'entrenament, el que es fa és agrupar les dades segons el seu sentiment (0 o 1) i s'agafa una mostra de la mida especificada (per defecte un 70%) de cada un dels grups. D'aquesta manera ens assegurem que tindrem la mateixa proporció de positius i negatius tant en el conjunt de train com el de test. Per tal de crear el conjunt de test és simplement agafar els elements que no han sigut seleccionant per al conjunt de train.

Les particions són numpy arrays.

2.3 Implementació del classificador

Per tal d'implementar el classificador s'ha creat una classe anomenada BayesClassifier, la qual conté totes les funcions necessàries per entrenar el model i fer prediccions.

2.3.1 Entrenament

Funció: `fit`

Per tal d'entrenar el model s'ha de cridar la funció fit, la qual rep les dades d'entrenament per paràmetre. La seva funcionalitat és cridar a la resta de funcions que ens permetran entrenar el model.

2.3.2 Calcul dels priors: P(positiu) i P(negatiu)

Funció: `__compute_target_probabilities`

Per tal de calcular els priors, és tan senzill com contar quants tweets són positius i dividir-ho entre el total i el mateix per als negatius. Per fer-ho es fa amb numpy, ja que les dades estan a un numpy array i aporta bon rendiment.

2.3.3 Creació dels diccionaris

Funció: `__extract_words`

S'han creat dos diccionaris, un amb les paraules dels tweets positius i un altre amb les paraules dels tweets negatius. Per tal de crear aquests diccionaris, es fa servir la classe 'Counter' del mòdul collections de la llibreria estàndard de Python. El que es fa és dividir els tweets segons el seu sentiment, i per cada tweet, es crea una llista amb les paraules que conté, i es passa al 'Counter' el qual genera un diccionari amb el nombre d'ocurrències de cada paraula. Hi ha dos comptadors, un per les paraules positives i un per les paraules negatives. La classe Counter ens va perfecte perquè disposa de la funció `most_common` que retorna els n elements més frequents. En cas que n sigui 'None' es retornen tots els elements. Això ens permet que a l'hora d'escollir la mida del diccionari, de cara al següent apartat, només haguem de cridar a aquesta funció.

A l'hora de contar les paraules s'exclouen les paraules buides ("stopwords" en anglès), ja que aquestes no acostumen a aportar significat sentimental, i s'obtenen millors resultats si les ometem.

El format dels diccionaris que es retornen és el següent:

```
positive_words:  {
    "word" : occurances of word given sentiment is positive,
    ...
}
negative_words:  {
    "word" : occurances of word given sentiment is negative,
    ...
}
```

2.3.4 Creació de la taula de probabilitats

Funció: `__compute_probability_table`

Per tal de poder aplicar Naive Bayes necessitem una taula amb les probabilitats condicionades de cada paraula. La taula es genera de la forma següent: Per cada

paraula en els diccionaris de l'apartat anterior, es divideix el nombre d'ocurrències d'aquella paraula entre el nombre total de paraules d'aquell sentiment. És a dir si estem recorrent el diccionari de paraules positives es dividiria el nombre d'ocurrències d'aquella paraula entre el nombre de paraules positives.

El resultat es retorna en forma de diccionari i té el format següent:

```
{
    'word' : [P(word|negative), P(word|positive)],
    ...
}
```

2.4 Predicció

Funció: `predict`

Un cop ja tenim totes les probabilitats calculades, ja podem fer prediccions aplicant Naive Bayes. El que es fa és, per cada paraula del tweet que es vol predir, multiplicar les seves probabilitats i finalment multiplicar per la probabilitat que sigui negatiu. Després fem el mateix, però multiplicant per la probabilitat que sigui positiu. El resultat que sigui major dels dos ens indicarà quin és el sentiment del tweet. En resum seria fer el següent:

$$S := \{Positiu, Negatiu\}$$

$$\arg\max_{x_i \in S} P(x_i) \cdot \prod_{i=1}^{n} P(word_i|x_i)$$

2.5 Validació dels resultats

Funció: `classification_report` i `cross_validation_score`

Per tal de validar com de bé s'estan classificant els resultats, es comparen les classificacions dels tweets de test amb les prediccions fetes i es calculen les següents metriques: accuracy, precision, negative predictive value, recall i specificity. S'han escollit aquestes metriques, ja que són les que en podem extreure de la confusion matrix, però principalment ens fixarem en l'accuracy, ja que és la que ens indica quants tweets hem classificat correctament.

A l'hora d'aplicar Cross-Validation només s'ha tingut en compte l'accuracy.

2.6 Laplace smoothing

Per tal de millorar les classificacions podem aplicar Laplace smoothing, així quan una paraula tingui probabilitat 0, no ens anularà la resta de probabilitats. Per defecte el codi aplica sempre Laplace smoothing (el valor de la constant és 1 per defecte), i si no es vol aplicar, és tan simple com assignar 0 al parametre

Laplace del constructor, d'aquesta manera, al ser la constant 0, no afecta en res l'smoothing.

3 Resultats

A continuació mostrarem els resultats de les diferents proves fetes i a posteriori els analitzarem. Els temps d'execució estan en segons.

```
==> 1. Testing Cross-Validation

Score for fold 1: 0.7337347895710875
Score for fold 2: 0.7367808707381234
Score for fold 3: 0.7484247513802736
Score for fold 4: 0.7529378109412981
Score for fold 5: 0.7568103857183587

Runtime: 31.90 seconds

==> 2.1 Testing different partition sizes

=== Train 60.0%, Test size: 40.0% ===

 == Without laplace smoothing ==
tp: 213875,    fp: 65482
fn: 98785,   tn: 247579
-------------------------------------------
accuracy:                  0.7374756480923607
precision:                 0.7655974255164538
negative predictive value: 0.7147942626831888
recall:                    0.684049766519542
specificity:               0.7908330964252973

Runtime: 6.23 seconds

 == With laplace smoothing ==
tp: 213660,    fp: 50662
fn: 99000,   tn: 262399
-------------------------------------------
accuracy:                  0.7608167218296973
precision:                 0.8083322614084336
negative predictive value: 0.7260645436207627
recall:                    0.6833621185952792
specificity:               0.8381721134219848

Runtime: 6.19 seconds
```

```
=== Train 70.0%, Test size: 30.0% ===

 == Without laplace smoothing ==
tp: 160708,    fp: 48745
fn: 73787,   tn: 186051
-------------------------------------------
accuracy:                  0.738899744508205
precision:                 0.7672747585377149
negative predictive value: 0.7160269090741154
recall:                    0.6853365743406042
specificity:               0.7923942486243377

Runtime: 6.35 seconds

 == With laplace smoothing ==
tp: 160501,    fp: 38014
fn: 73994,   tn: 196782
-------------------------------------------
accuracy:                  0.761325062700968
precision:                 0.8085081731859054
negative predictive value: 0.7267335362070494
recall:                    0.6844538263075972
specificity:               0.8380977529429803

Runtime: 6.35 seconds

=== Train 80.0%, Test size: 20.0% ===

 == Without laplace smoothing ==
tp: 107508,    fp: 32289
fn: 48822,   tn: 124241
-------------------------------------------
accuracy:                  0.7407434635300134
precision:                 0.7690293783128394
negative predictive value: 0.7178946395243351
recall:                    0.6876990980617924
specificity:               0.7937200536638344

Runtime: 6.42 seconds

 == With laplace smoothing ==
tp: 107397,    fp: 25310
fn: 48933,   tn: 131220
-------------------------------------------
accuracy:                  0.7626957744678131
```

```
precision:                 0.8092790885183148
negative predictive value: 0.7283808762551831
recall:                    0.6869890616004606
specificity:               0.8383057560850955

Runtime: 6.45 seconds

==> 2.2 Testing different dictionary sizes (Train: 70.0%, Test: 30.0%)

=== Dictinary Size 100 ===

 == Without laplace smoothing ==
tp: 136915,    fp: 69348
fn: 97580,   tn: 165448
-------------------------------------------
accuracy:                  0.6442974614897793
precision:                 0.6637884642422538
negative predictive value: 0.6290128807579421
recall:                    0.5838717243438026
specificity:               0.7046457350210396

 == With laplace smoothing ==
tp: 127739,    fp: 50668
fn: 106756,   tn: 184128
-------------------------------------------
accuracy:                  0.6645492881815335
precision:                 0.7159976906735722
negative predictive value: 0.6329945957838864
recall:                    0.5447408260304057
specificity:               0.7842041602071586

Runtime: 5.43 seconds

=== Dictinary Size 1000 ===

 == Without laplace smoothing ==
tp: 151868,    fp: 54159
fn: 82627,   tn: 180637
-------------------------------------------
accuracy:                  0.7085262662186149
precision:                 0.7371266872788518
negative predictive value: 0.6861439467606661
recall:                    0.6476385423996247
specificity:               0.7693359341726435

 == With laplace smoothing ==
```

```
tp: 146541,    fp: 37976
fn: 87954,   tn: 196820
-----------------------------------------
accuracy:                  0.7316590345862163
precision:                 0.7941869854810126
negative predictive value: 0.6911445567362189
recall:                    0.6249216401202584
specificity:               0.8382595955638086

Runtime: 5.68 seconds

=== Dictinary Size 10000 ===

 == Without laplace smoothing ==
tp: 155058,    fp: 40084
fn: 79437,   tn: 194712
-----------------------------------------
accuracy:                  0.7453158061842226
precision:                 0.7945906058152525
negative predictive value: 0.7102415110031406
recall:                    0.6612422439710868
specificity:               0.8292815891241759

 == With laplace smoothing ==
tp: 152978,    fp: 37762
fn: 81517,   tn: 197034
-----------------------------------------
accuracy:                  0.745831477697207
precision:                 0.8020236971794065
negative predictive value: 0.7073534110450151
recall:                    0.6523721188085034
specificity:               0.8391710250600521

Runtime: 5.47 seconds

=== Dictinary Size None ===

 == Without laplace smoothing ==
tp: 160708,    fp: 48745
fn: 73787,   tn: 186051
-----------------------------------------
accuracy:                  0.738899744508205
precision:                 0.7672747585377149
negative predictive value: 0.7160269090741154
recall:                    0.6853365743406042
specificity:               0.7923942486243377
```

```
 == With laplace smoothing ==
tp: 160501,    fp: 38014
fn: 73994,   tn: 196782
-------------------------------------------
accuracy:                  0.761325062700968
precision:                 0.8085081731859054
negative predictive value: 0.7267335362070494
recall:                    0.6844538263075972
specificity:               0.8380977529429803

Runtime: 6.36 seconds

==> 2.3 Testing different partition sizes with fixed size dictionary (1000)

=== Train 60.0%, Test size: 40.0% ===

 == Without laplace smoothing ==
tp: 202820,    fp: 72279
fn: 109840,   tn: 240782
-------------------------------------------
accuracy:                  0.708945360631975
precision:                 0.7372618584582278
negative predictive value: 0.6867281573888689
recall:                    0.6486918697626816
specificity:               0.7691216727730379

 == With laplace smoothing ==
tp: 195486,    fp: 50625
fn: 117174,   tn: 262436
-------------------------------------------
accuracy:                  0.7318309598047692
precision:                 0.7943001328668772
negative predictive value: 0.6913305761175944
recall:                    0.6252350796392248
specificity:               0.8382903012511939

Runtime: 5.15 seconds

=== Train 70.0%, Test size: 30.0% ===

 == Without laplace smoothing ==
tp: 151868,    fp: 54159
fn: 82627,   tn: 180637
-------------------------------------------
accuracy:                  0.7085262662186149
```

```
precision:                  0.7371266872788518
negative predictive value: 0.6861439467606661
recall:                     0.6476385423996247
specificity:                0.7693359341726435

 == With laplace smoothing ==
tp: 146541,    fp: 37976
fn: 87954,   tn: 196820
-------------------------------------------
accuracy:                   0.7316590345862163
precision:                  0.7941869854810126
negative predictive value: 0.6911445567362189
recall:                     0.6249216401202584
specificity:                0.8382595955638086

Runtime: 5.30 seconds

=== Train 80.0%, Test size: 20.0% ===

 == Without laplace smoothing ==
tp: 101438,    fp: 36102
fn: 54892,   tn: 120428
-------------------------------------------
accuracy:                   0.7091542542990475
precision:                  0.7375163588774175
negative predictive value: 0.6869039470682181
recall:                     0.6488709780592337
specificity:                0.7693605059732959

 == With laplace smoothing ==
tp: 97706,    fp: 25331
fn: 58624,   tn: 131199
-------------------------------------------
accuracy:                   0.7316531355878029
precision:                  0.7941188422994709
negative predictive value: 0.6911649273270363
recall:                     0.6249984008187808
specificity:                0.8381715964990737

Runtime: 5.40 seconds
```

A l'utilitzar Cross-Validation obtenim un accuracy entre 0,73 i 0,76. El resultat no varia excessivament entre folds, per tant, sembla que està funcionant prou bé.

Respecte a la mida de les particions de train i test, es pot observar una diferència a mesura que s'incrementa la mida del conjunt d'entrenament, tot i que no molt gran. Tot i això, és important no fer una partició de train massa gran, ja que hi

ha risc d'overfitting o que no tinguem suficients dades per fer proves.

També s'han provat diccionaris de diferent mida. Hem pogut observar que la mida del diccionari és molt important. Quan el diccionari és molt petit, les prediccions són dolentes i a mesura que augmentem la mida les prediccions van millorant. Això és degut al fet que si no tenim suficients dades, el model és molt més feble, ja que hi ha moltes paraules que no ha vist mai i, per tant, la seva probabilitat és 0.

Si deixem el diccionari a una mida fixa i anem canviant la mida de la partició de train, el canvi d'accuracy en canviar la mida de les particions passa a ser molt poc, ja que abans en canviar la mida de les particions feia canviar la mida del diccionari també, però ara en deixar-lo fixa, hi ha menys marge de millora.

Totes les proves anteriors també han sigut realitzades aplicant i sense aplicar Laplace smoothing. El fet d'aplicar-lo representa una millora substancial, incrementant un 3% l'accuracy en la majoria de casos, i la resta de mètriques també les millora substancialment. Així que pel mateix cost, val la pena aplicar Laplace smoothing sempre que es pugui.

4 Problemes trobats durant el projecte i com els heu resolt.

El principal problema trobat ha sigut treballar amb una base de dades tan gran. En primera instància ho vaig començar amb Python pur i era realment lent (uns 60 segons per entrenar i validar). Aquest problema, però, va ser fàcilment solucionat utilitzat numpy i pandas, ja que permeten vectoritzar moltes de les operacions i redueixen substancialment el temps d'execució. (Uns 6-7 segons per entrenar i fer les prediccions)

5 Conclusions i treballs futurs

Hem pogut observar que un classificador que utilitza Naive Bayes, és relativament simple d'implementar i té un cost computacional bastant baix. A més a més les prediccions que realitza són bones, així que és una molt bona opció a tenir en compte quan ens trobem amb un problema de classificació. En aquesta implementació, la precisió és millorable, ja que les dades han sigut poc processades. De cara a un futur es podria aplicar lematització o stemming per tal de tractar paraules que segurament són la mateixa, però estan escrites diferent (contenen un typo, o és una altra conjugació del mateix verb per exemple).