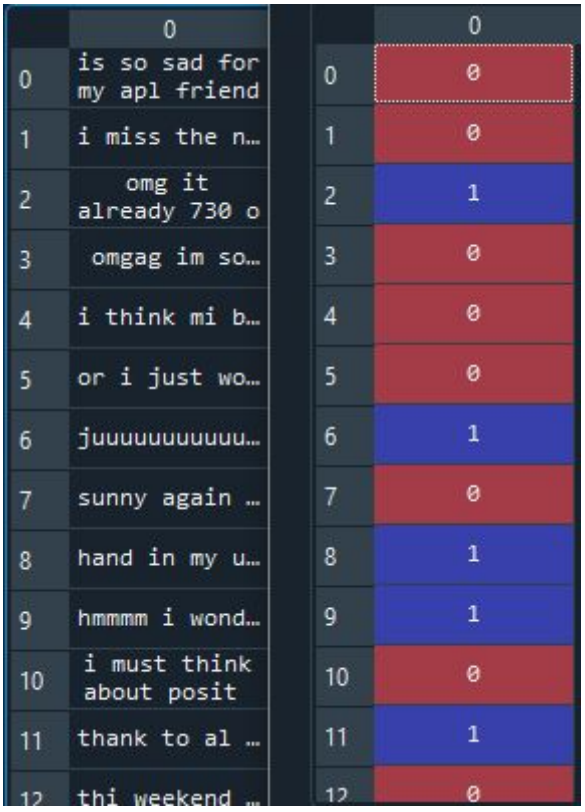
Naïve Bayes - 3a pràctica

Coneixement, Raonament i Incertesa



	0
0	is so sad for my apl friend
1	i miss the n…
2	omg it already 730 o
3	omgag im so…
4	i think mi b…
5	or i just wo…
6	juuuuuuuuuuu…
7	sunny again …
8	hand in my u…
9	hmmmm i wond…
10	i must think about posit
11	thank to al …
12	thi weekend …

	0
0	0
1	0
2	1
3	0
4	0
5	0
6	1
7	0
8	1
9	1
10	0
11	1
12	0

Josep Bravo Bravo - 1526453

Índex

Introducció

En aquesta pràctica s'ha implementat l'algorisme de classificació de Naïve Bayes sobre una base de dades real. L'objectiu principal d'aquesta pràctica s'aplica Naïve Bayes per determinar si un Tweet és positiu o negatiu.
Per a poder fer-ho haurem d'implementar des de zero tota la lectura del csv (BD) i tractament d'aquesta. També caldrà crear de manera òptima tant els arrays com els diccionaris per a fer la predicció, aplicar Laplace Smoothing per a eliminar la indecisió, i avaluar diferents configuracions del model.
En aquesta pràctica tenim com a objectiu principal assimilar el classificador de Naïve Bayes après en les classes de teoria en un exemple real.

Explicació BD

La nostra base de dades té com a objectiu predir si un tweet és positiu o negatiu. Aquesta BD conté aproximadament 1,5 M files. Cada fila té el següent format:

16; I fell in love again; 02/12/2015; 1

La primera columna és l'identificador, la segona el tweet que volem classificar, la tercera la data d'aquest tweet i la quarta la classificació real.

Solució proposada (Exercicis C, B i A)

Per a fer aquest informe de la pràctica explicaré la solució proposada per a crear el classificador Naïve Bayes sense seguir estrictament l'ordre dels exercicis, ja que he fet tots alhora al ritme que creava el codi.

Per a realitzar el nostre classificador, necessitarem obtenir l'array X i Y, on X són els tweets a classificar i Y la seva classificació real. Aquests dos arrays els dividirem en train i test per a fer les prediccions.

L'estructura principal necessària per a fer aquesta pràctica serà dos diccionaris. Cada diccionari tindrà les paraules utilitzades en el tweet d'un sentiment en concret i la seva freqüència d'aparició.

Necessitem aquests dos diccionaris, tant el que conté paraules de tweets amb sentiment negatiu, com el que conte paraules de tweets amb sentiment positiu, per a fer la predicció.

El classificador Naïve Bayes es basa en el principi de classificar un tweet segons si les seves paraules tenen més probabilitat de ser de tweets negatius o de tweets positius, o el que és el mateix, si les seves paraules tenen més probabilitats d'aparèixer en un diccionari de paraules negatives o en un diccionari de paraules positives. D'aquí la necessitat de tenir aquests dues diccionaris.

Per explicar aquesta pràctica i el codi generat per a resoldre-la començaré descrivint les 3 funcions creades per aquesta pràctica i el main:

- open_fitxer(): Aquesta funció és l'encarregada de llegir la BD (csv), tractar la BD, i retornar l'array numpy X que conté els tweets a classificar, i l'array numpy Y que conté les seves classificacions reals.
  Per a fer això utilitzem la funció read_csv de panda per a transformar el csv a un dataset de panda. Després utilitzem la funció dropna per eliminar totes les files que continguessin nulls (sols són 24 de 1,5M).
  Finalment convertim tot a una matriu numpy i retornem senzillament els arrays X i Y.

- create_dicts(X, Y, n): Aquesta funció li entra l'array X i Y (equivalents al x_train i y_train) i una dimensió de diccionari n, i serà l'encarregada de generar els dos diccionaris necessaris per a fer aquesta pràctica.
  Per a fer això primer de tot crearem el array X de tweets de sentiment negatiu (np_X_sentiment0) i el mateix per sentiment positiu (np_X_sentiment1).
  El següent pas serà crear una llista (X_flat) on ara cada posició és una paraula i no un string (tweet).
  Finalment crearem els dos diccionaris mitjançant dos bucles, on per cada paraula de X_flat, l'afagirem al diccionari, i en cas de ja estar en el diccionari, sumarem 1 a la seva freqüència d'aparició.

Finalment retornarem les n paraules del diccionari més frequents. Per a fer això ordenarem el diccionari per valor (freqüencia d'aparicio) convertintlo en una llista de tuples i agafant els n primers valors (valors més freqüents).
En cas que n sigui -1 retornarem el diccionari de la mida per defecte.

- prediccio(x_test, dict_sentiment_0, dict_sentiment_1): En aquesta funció apliquem les fórmules de Naïve Bayes que ja hem vist a classes de teoria i en entregues de problemes, però en l'exemple amb una BD real.
  Per a fer aquesta classificació guardarem en variables com la dimensió dels diccionaris, la suma dels valors dels diccionaris... fora del bucle per no haver de calcular aquest número en cada iteració.
  Crearem també el array y_pred (el que conté les nostres prediccions produïdes pel classificador Naïve Bayes).
  Després fem el bucle que implementa la fórmula. Per cada fila del test, agafarem les seves paraules del tweet i calcularem la probabilitat que aquell tweet sigui negatiu o positiu analitzant la probabilitat de cada paraula de ser d'un diccionari amb paraules positives o negatives. També aplicarem Laplace Smoothing per a eliminar la indecisió. Finalment fem la predicció mirant quina d'aquestes dues probabilitats calculades és més gran, i un cop acabat el bucle, retornem la predicció feta pel classificador Naïve Bayes (y_pred).

- main: El main és l'encarregat de cridar a les funcions realitzades anteriorment per a fer les prediccions. Primer de tot inicialitzarem el dataset amb open_fitxer(), després utilitzem la funció train_test_split de sklearn per a dividir X i Y en train i test, elegint les files de manera aleatoria amb la proporció que li indiquem.
  Seguidament creem els diccionaris, determinant la seva mida en el valor n i finalment fem les prediccions i avaluem la prediccio amb la funció accuracy_score de sklearn (funció simple que compara el array de la predicció i el real per trobar TP, TN... i aplicar la fórmula d'accuracy). Utilitzant aquest main farem les diferents proves que ens demanen a l'enunciat, modificant aquest model del classificador Naïve Bayes.

Per a fer l'avaluació de les prediccions que realitza el classificador utilitzo la mètrica accuracy, ja que és la que ens aporta un resultat més real i fàcil d'entendre quan tenim un dataset balencejat com en el nostre cas.
En tenir una base de dades de 1,5 M de files, i agafar el 30% d'aquestes files de manera aleatòria per a crear el test, considero que no fa falta implementar cap mètode de cross-validació, ja que com hem vist en moltes ocasions (en diferents treballs i diferents assignatures), en tenir un test tan gran de files agafades aleatòriament, les accuracys que optindrem en les diferents parts del cross-validation seria molt similar a la que obtenim realitzant-ho només una vegada.

Resultats i anàlisi (Proves exercici B i A)

Apartat B

1. Què passa en el primer cas, quan es va ampliant el nombre de tweets i el diccionari per fer el train?

70% de train i un 30% de test:

```
Accuracy predita: 0.7291043376718576
Temps total de execucio: 13.600338697433472
```

80% de train i un 20% de test:

```
Accuracy predita: 0.7301282379113714
Temps total de execucio: 12.152130126953125
```

90% de train i un 10% de test:

```
Accuracy predita: 0.7311670544915233
Temps total de execucio: 11.283085346221924
```

Com podem veure en les proves anteriors en augmentar el conjunt de train també augmenta una mica el accuracy. Com podem veure augmenta molt poc, ja que el model de per si ja té moltíssimes dades per entrenar i un 10% més o menys de 1,5 M files no genera gaire diferència.

2. Com afecta la mida del diccionari?

Totes les proves les faig amb un 70% de train i un 30% de test.

Diccionari mida n=10:

```
Accuracy predita: 0.5539204405008481
Temps total de execucio: 11.333119869232178
```

Diccionari mida n=1000:

```
Accuracy predita: 0.6953976696414111
Temps total de execucio: 12.542370319366455
```

Diccionari mida n=10000:

```
Accuracy predita: 0.7312778615934061
Temps total de execucio: 13.446136713027954
```

Diccionari mida n=100000:

```
Accuracy predita: 0.7319427042047033
Temps total de execucio: 12.945123195648193
```

En les anteriors proves podem veure que per diccionaris molt petits no aconseguim predir correctament la classificació real dels tweets, i en canvi en tenir moltes paraules en el diccionari augmentem el accuracy. Això és pel fet que en un tweet podem utilitzar moltíssimes paraules diferents i necessitarem un diccionari molt gran per poder trobar aquella paraula i tenir un criteri bo per classificar.

3. Com afecta la mida del conjunt de tweets de test?
Totes les proves les faig amb una mida de diccionaris de 1000.
70% de train i un 30% de test:

```
Accuracy predita: 0.6953976696414111
Temps total de execucio: 12.528980731964111
```

80% de train i un 20% de test:

```
Accuracy predita: 0.6961445521262178
Temps total de execucio: 11.619808673858643
```

90% de train i un 10% de test:

```
Accuracy predita: 0.6970235507709617
Temps total de execucio: 11.699665307998657
```

Com podem veure en les proves anteriors si fixem la mida del diccionari el accuracy varia amb menys interval en canviar el % de train i test.
La conclusió que extraiem d'aquest apartat és que no utilitzar Laplace Smoothing no és una bona idea, ja que per diccionaris petits l'accuracy és molt baixa, quasi igual que el cas aleatori, de manera que no té sentit el classificador, i en general hi ha moltes files que no predirà bé si ens trobem amb paraules probabilitat 0, de manera que l'accuracy total baixa bastant. Per altra banda podem apreciar que el criteri d'agafar els valors més freqüents d'un diccionari no ens millora el accuracy.

Apartat A

1. Què passa en el primer cas, quan es va ampliant el nombre de tweets i el diccionari per fer el train?
70% de train i un 30% de test:

```
Accuracy predita: 0.7759544327102564
Temps total de execucio: 12.509253978729248
```

80% de train i un 20% de test:

```
Accuracy predita: 0.7762005523307848
Temps total de execucio: 11.72850489616394
```

90% de train i un 10% de test:

```
Accuracy predita: 0.7765681335822231
Temps total de execucio: 11.19225788116455
```

Com podem veure en les proves anteriors passa el mateix que abans, a l'augmentar el conjunt de train també augmenta una mica el accuracy. Com podem veure augmenta molt poc, ja que el model de per si ja té moltíssimes dades per entrenar i un 10% més o menys de 1,5 M files no genera gaire diferència.

2. Com afecta la mida del diccionari?
Totes les proves les faig amb un 70% de train i un 30% de test.
Diccionari mida n=10:

```
Accuracy predita: 0.5252768046641266
Temps total de execucio: 11.82775354385376
```

Diccionari mida n=1000:

```
Accuracy predita: 0.7375043683569011
Temps total de execucio: 12.026530742645264
```

Diccionari mida n=10000:

```
Accuracy predita: 0.7686901748195123
Temps total de execucio: 12.4005126953125
```

Diccionari mida n=100000:

```
Accuracy predita: 0.7759693490508945
Temps total de execucio: 13.181177854537964
```

En les anteriors proves podem veure el mateix que abans, que per diccionaris molt petits no aconseguim predir correctament la classificació real dels tweets, i en canvi en tenir moltes paraules en el diccionari augmentem el accuracy. Això és pel fet que en un tweet podem utilitzar moltíssimes paraules diferents i necessitarem un diccionari molt gran per poder trobar aquella paraula i tenir un criteri bo per classificar.

3. Com afecta la mida del conjunt de tweets de test?

Totes les proves les faig amb una mida de diccionaris de 1000.

70% de train i un 30% de test:

```
Accuracy predita: 0.7375043683569011
Temps total de execucio: 12.451145648956299
```

80% de train i un 20% de test:

```
Accuracy predita: 0.737828266039328
Temps total de execucio: 12.090969800949097
```

90% de train i un 10% de test:

```
Accuracy predita: 0.7386337484337843
Temps total de execucio: 12.631916046142578
```

Com podem veure en les proves anteriors si fixem la mida del diccionari el accuracy varia amb menys interval al canviar el % de train i test.

Conclusions

A partir de totes les proves i resultats extrets podem extreure dues conclusions:

- El criteri de fixar els diccionaris amb les n paraules més freqüents no és un criteri que millori el resultat. Una millor idea seria aplicar mètodes avançats com BERT on tenim en compte el context de les paraules del costat, però són mètodes molt més avançats i complexos.
- Aplicar Laplace Smoothing és molt important, ja que en cas de no fer-ho a la que tinguem una paraula del tweet desconeguda la probabilitat és 0 i aquella fila del test la crearem a l'atzar, de manera que el accuracy total baixa bastant.

Gràcies a aquesta pràctica he après a implementar el classificador Naïve Bayes des de 0, estudiat a les classes de teoria, i he assimilat a la perfecció els coneixements ensenyats. També hem acabat d'assolir i entendre perfectament les mètriques, classificadors i tot el que significa crear un model.

El format de la pràctica m'ha permès entendre més els conceptes i veure com s'aplicarien a un problema real. Haver de fer-ho en una base de dades tan gran ha ajudat a veure la complexitat dels problemes reals.