Naïve Bayes

Coneixement, Raonament i Incertesa

Enginyeria Informàtica 2021-2022

Martí Caixal Joaniquet - 1563587

Índex

Introducció

La pràctica tracta de dissenyar, programar i testejar un aprenentatge bayesià. Més concreatament, s'ha d'utilitzar l'algorisme "Naïve Bayes" per poder classificar un tweet entre positiu o negatius. Està dividida en 3 aparts:

1. Disseny i implementació de l'algorisme.
2. Modificar mides de diccionari i conjunt d'aprenentatge.
3. Efecte del Laplace Smoothing

Dades

Les dades amb les que es treballa són un conjunt de tweets emmagatzemats a un arxiu csv. Es tracta de més d'un milió de mostres, cada una d'elles en el següent format:

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

El 16 és un identificador, la segona columna és el tweet en qüestió, la tercera la data de publicació i per últim l'etiqueta classificadora (0 si és negatiu, 1 si és positiu).

Estem davant d'un conjunt de dades balancejat, significant que es té el mateix nombre de mostres d'una classe que de l'altra.

Disseny i implementació

Per poder implementar l'algorisme es fa ús d'un seguit de variables, a continuació es mostren les principals:

- wc (diccionari): Per cada classe, emmagatzema el número de vegades que una paraula apareix. Es tracta, doncs, de diccionaris anidats. A continuació es mostra un exemple del format.

```
wc = {'positive': { 'hello' : 4,
                    'car' : 7,
                    'pencil' : 5
                  },
       'negative': { 'window': 2,
                    'paper' : 10
                  }
      }
```

- dictionary (set): emmagatzema cada paraula que es troba durant l'entrenament. Al tractar-se d'un set, ens assegurem que totes les paraules són úniques i no es repeteixen.

- log_prior_probability (diccionari): Guarda la probabilitat a priori que un tweet sigui positiu o negatiu.

- aplace_smoothing (float): Conté el valor del Laplace Smoothing que s'utilitza durant la fase de predicció.

Tot i que moltes variables podrien ser simples llistes, el fet d'utilitzar diccionaris va bé per poder accedir directament al valor que es necessita.

L'algorisme està implementat dins d'una classe anomenada "NaiveBayes". Les principals funcions que conté són "Fit" i "Predict". La classe hereda de "sklearn.base.BaseEstimator" per tal de poder utilitzar el "cross_validate" de "Sklearn".

Funció "Fit"

És la funció utilitzada durant l'entrenament del model. Primer omple les variables "tweet_num" i "log_prior_probability".

Seguidament fa els següents passos per cada tweet:

1. Separar tweet per paraules.
2. Mirar la classe a la que pertany el tweet.
3. Per cada paraula:
    a. Afegir-la al set "dictionary".
    b. Incrementar el nombre d'aparicions per classe al diccionari "wc".

Funció "Predict"

És la funció utilitzada durant les prediccions de les mostres. Per cada mostra fa els següents passos:

1. Separar tweet per paraules.
2. Posar variables positive_count i negative_count a 0.
3. Per cada paraula:
    a. Si no apareix al diccionari, continuar a la següent.
    b. Calcular $p(w_i \mid \text{'positive'})$ i $p(w_i \mid \text{'negative'})$.
    c. Sumar aquests valors a positive_count i negative_count
4. Afegir la probabilitat apriori de cada classe a positive_count i negative_count
5. Guardar com a resultat el la classe amb el valor més gran entre positive_count i negative_count

Execució

El programa es pot executar amb Python des d'una línia de comandes. Abans, però, caldrà instal·lar totes les dependències necessàries:

```
python3 -m pip install -r requirements.txt
```

Seguidament es pot executar el programa:

```
python3 ./src/main.py
```

Addicionalment, també es pot executar amb arguments. La següent línia mostra com executar només amb 50000 mostres aleatòries per tal de tenir una execució més ràpida:

```
python3 ./src/main.py --n_rows 50000
```

Es pot trobar una descripció de cada argument:

```
python3 ./src/main.py -h
```

```
usage: main.py [-h] [--smooth SMOOTH] [--n_rows N_ROWS] [--n_splits N_SPLITS]

optional arguments:

-h, --help           show this help message and exit

--smooth SMOOTH      Value for Laplace Smoothing

--n_rows N_ROWS      Amount of rows to read from csv file

--n_splits N_SPLITS  K_Fold splits
```

Mètrica i anàlisi

Al tractar-se d'un conjunt de mostres balancejat, les mètriques es simplifiquen bastant. Per exemple, el valor que s'obté d'accuracy és fiable. En unes dades no balancejades, el valor podria ser molt alt però les prediccions dolentes, i per tant s'hauria d'utilitzar F1-Score. El model que s'està creant només es vol per saber si un tweet és positiu o negatiu, sense donar més importància a un o l'alte. L'accuracy ens permet saber exactament això, el percentatge de prediccions ben fetes.

Si per altra banda es volgués filtrar els tweets tal que només es mostrin els que són positius, s'hauria d'utilitzar una altra mètrica. La precission ens seria una bona mètrica, doncs calcula el percentatge de prediccions positives correctes d'entre el total de prediccions positives. Un precission alt permetria assegurar-nos que tots els tweets que passen el filtre són positius.

D'altre banda, si l'objectiu sigués permetre tants positius com sigui possible, sense importar els negatius que es colessin, s'hauria de mirar el Recall.

De totes formes, per aquesta pràctica ambdues classes són igual d'importants i es mirarà l'accuracy.

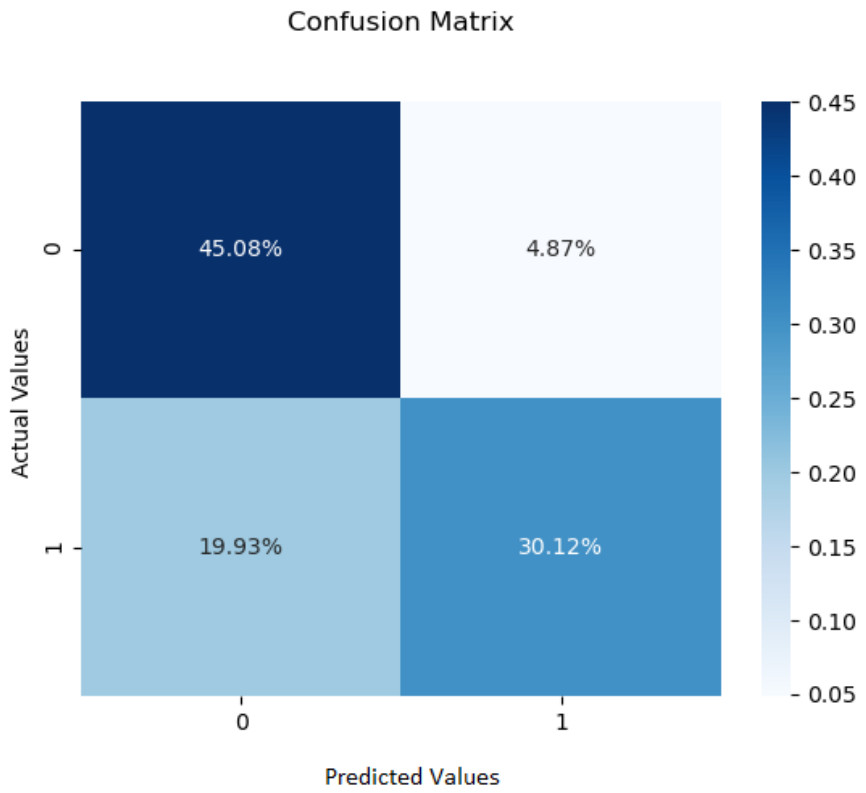
Les proves efectuades a la pràctica s'han realitzat amb un "K-Fold Cross Validation" de 10 folds. Per aconseguir la mètrica simplement es fa una mitjana de les 10 mètriques obtingudes. Al haver-hi més d'una predicció, no és suficient sabent la mitjana. Cal comprovar també que els resultats siguin consistents. Ajudant-se de la desviació estàndard de totes les mètriques es pot estar segur que són consistents.

Els resultats obtinguts són una Accuracy de 75% amb desviació estàndard de 0.001. Si bé no són excepcionalment bons, almenys les prediccions són consistents entre elles.

La matriu de confusió inferior mostra com estan distribuïdes les classificacions. Sumant els TP i TN → 45 + 30 = 75 es pot veure que dona l'accuracy prèviament mencionada.

També es pot veure que és bastant més capaç de classificar correctament els tweets negatius que els positius.

Per últim, utilitzar el "Leave One Out" no és plausible ja que es té moltes mostres i el temps d'execució triga molt.



Cal anotar que la matriu està normalitzada tant per files com per columnes, sumant 1 entre totes les cel·les.

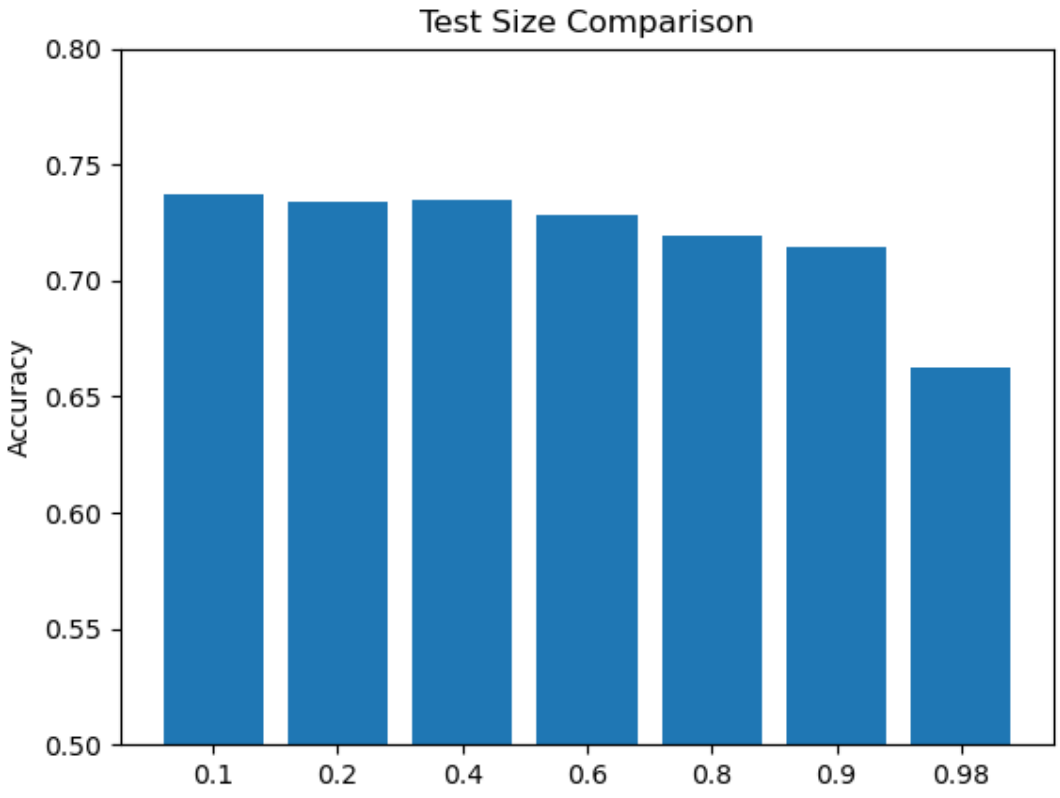
Diferents mides de Train i diccionaris

Canviant mida del conjunt test

A vegades pot ser interessant veure quantes mostres són necessàries per aconseguir un entrenament satisfactori. La prova de l'apartat anterior s'havia fet amb un 20% de les mostres pel test i el 80% restant pel training. Ara, en canvi, es realitzen diferents execucions, cada una amb una proporció traint/test diferent.

Com bé es veu al gràfic inferior, a mesura que hi ha més mostres al test, i per tant menys al training, les prediccions tendeixen a ser pitjors. És un resultat esperat, al cap i a la fi com més dades es tingui per aprendre, més fàcil serà fer una predicció d'una nova mostra.

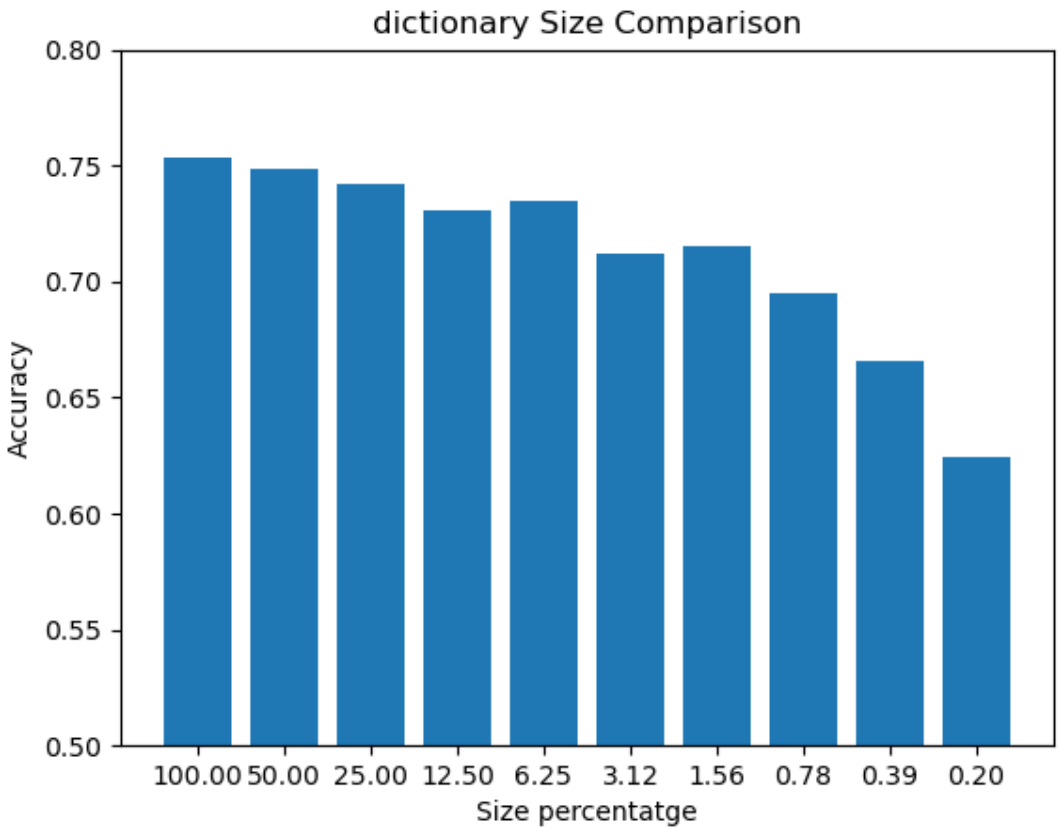
El punt interessant a comentar és que s'ha de reduir molt la mida de les mostres de training fins aconseguir unes prediccions bastant més dolentes que l'original. Es veu que destintant només el 10% de les mostres al training, es segueix aconseguint un accuracy per sobre del 70%. És quan ja es destinen només un 2% de les mostres que els resultats comencen a empitjorar.



Canviant mida diccionari

Una altra manera de veure la quantitat d'informació necessària per fer bones prediccions és limitant la mida del diccionari. És a dir, limitar la quantitat de paraules que pot aprendre el model. Com també és d'esperar, les prediccions empitjoren a mesura que hi ha menys paraules al diccionari.

Ara, si més no, s'aconsegueix mantenir uns bons resultats tot i tenir el diccionari a l'1% de la seva mida original. És només quan la mida baixa del 0.2% que l'accuracy ja és inferior al 65%.



Més mostres sense augmentar el diccionari

Una última prova que s'ha fet és augmentar la mida del conjunt traint però mantenir una mida fixa del diccionari.

Els resultats són els mateixos quan la mida del diccionari es petita. Tot i ampliar les mostres, quan el diccionari està ple, les noves mostres no es poden analitzar al complet i per tant no aporten gaire informació al model. En canvi, quan es permet una mida gran del diccionari, triga més a arribar al punt on no hi ha milora.

Laplace Smoothing

L'algorisme Naïve Bayes es basa en ajuntar probabilitats d'events per aconseguir una final. És molt probable que durant la fase d'entrenament alguna paraula no s'hagi vist i per tant tingui una probabilitat de 0. Això comporta que la probabilitat final també acabi sent 0.

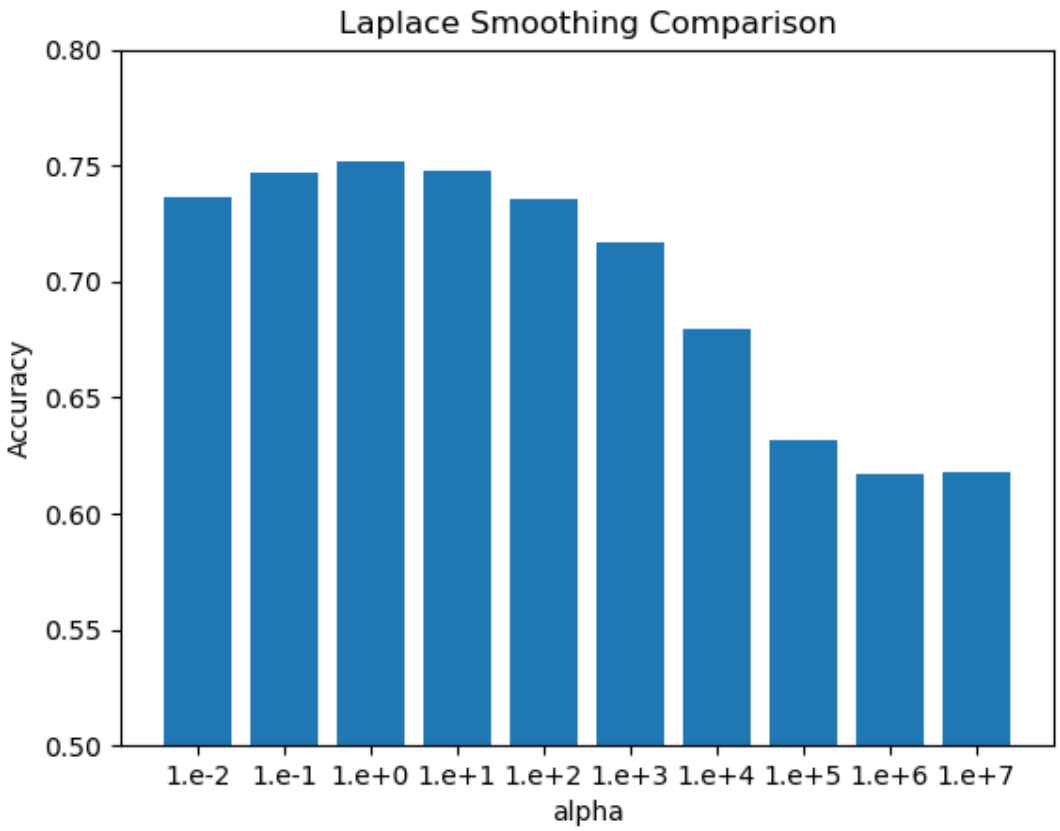
Fins ara, sempre que es trobava una situació com aquesta, on una paraula no apareix mai en tweets d'una classe, la paraula en qüestió era ignorada de la mateixa manera que s'ignoren les que no existeixen al diccionari.

Per solucionar el problema es fa ús del Laplace Smoothing, afegint un valor alpha a l'hora de calcular les probabilitats. D'aquesta manera, s'evita que una probabilitat sigui 0 encara que la paraula no existeixi durant la fase d'aprenentatge.

A mesura que augmenta el valor d'alpha, el "likelihook" es mou en direcció a una distribució uniforme (0.5). El valor òptim per la gran majoria de vegades és 1.

La imatge inferior demostra més gràficament la pèrdua d'accuracy a mesura que s'augmenta el valor d'alpha. També es veu com el valor 1 dona els millors resultats.

De totes formes, sense utilitzar Laplace Smoothing s'arribava a una accuracy del 74, mentre que ara amb prou feines es supera el 75%. Al tenir moltes dades per fer l'entrenament, segurament hi ha pocs casos on realment sigui necessari aplicar el smoothing.



Conclusions

Havent fet aquesta pràctica s'ha vist que tenir un gran nombre de mostres ajuda a tenir unes bones prediccions. Si només s'haguessin tingut 10K no s'hauria aconseguit el 75% d'accuracy. També, tot i tenir moltes mostres, és important guardar totes les paraules que es pugui per aconseguir obtenir la millor informació de cada classe.

Com a millora al futur, estaria bé modificar el programa per tal que es pogués adaptar a qualsevol tipus de Naïve Bayes, com per exemple un detector de missatges spam.