CLASSIFICACIÓ TWEETS AMB NAÏVE BAYES

Òscar Urenda Moix 1639392

Coneixement, raonament i incertesa.

OBJECTIUS

L'objectiu d'aquesta pràctica és classificar la intencionalitat dels tweets en funció de si son tweets negatius o positius.
Per conseguir aquesta classificació aplicarem un classificador que utilitzarà un aprenentatge bayesià.

L'aprenentatge Bayesià en el context de la nostra classificació de tweets, consisteix en calcular les probabilitats de que les paraules corresponguin a un tweet positiu o un negatiu.

Per a fer això partiem de dos possibles documents amb les dades d'entrenament del nostre classificador: FinalStemmedSentimentAnalysisDataset.csv i SentimentAnalysisDataset.csv, pero només he pogut utilitzar la primera degust a que la segona tenia files amb masses arguments i no he sapigut arreglarla per tal de poder practicar amb ella.

EXERCICI 1

La primera versió de l'algorisme va ser la següent:

```
import pandas as pd
from sklearn.model_selection import train_test_split
from collections import defaultdict
print('Lectura de la BD')
X =
pd.read_csv('FinalStemmedSentimentAnalysisDataset.csv',sep=';')
print(X)
print('Comprovacio de nans')
print(X.isna().sum().sort_values() / len(X) * 100.)

print('Eliminació de nans')
X=X.dropna()
print(X.isna().sum().sort_values() / len(X) * 100.)

print('Creacio de train i test, separacio de les dades')
y = X['sentimentLabel']
X_train, X_test, y_train, y_test = train_test_split(X, y,
test_size=0.2, random_state=0)
print('longituds: X:',len(X), 'X_train: ', len(X_train), 'X_test',
len(X_test), 'SUMA DE TEST I TRAIN', len(X_train)+ len(X_test))

truePos = 0
trueNeg = 0
falsePos = 0
falseNeg = 0

print('Clasificació paraules (negatiu,positiu)')
frases = X['tweetText']
pos_neg = X['sentimentLabel']

cont_paraules_valor = defaultdict(lambda: [0, 0])
for tweetText, valor in zip(frases, pos_neg):
   palabras = tweetText.split()
   for palabra in palabras:
            cont_paraules_valor[palabra][valor] += 1
print('Algorisme de Bayes:')

probabilitats  ={}
tweets = X_train

tweetsNegatius = tweets[tweets['sentimentLabel'] == 0]
tweetsPositius = tweets[tweets['sentimentLabel'] == 1]
probNegatius = len(tweetsNegatius)/len(tweets)
probPositius = len(tweetsPositius)/len(tweets)
```

```
probs = [probNegatius,probPositius]
numParaules = [0,0]
for tweet in tweetsPositius['tweetText']:

   numParaules[1] += len(tweet.split())

for tweet in tweetsNegatius['tweetText']:
   numParaules[0] += len(tweet.split())

for paraula in cont_paraules_valor.keys():
   numPositiu = cont_paraules_valor[paraula][1]
   numNegatiu = cont_paraules_valor[paraula][0]
   auxPos = numPositiu/numParaules[1]
   auxNeg = numNegatiu/numParaules[0]

   if paraula not in probabilitats:
       probabilitats[paraula] = [0,0]
   probabilitats[paraula] = [auxPos,auxNeg]
print('Probabilitats fetes')
for tweet,valor in
zip(tweets['tweetText'],tweets['sentimentLabel']):
   probabilitatActual = [1, 1]
   paraules = tweet.split()
   for paraula in paraules:
       if paraula in probabilitats:
           probabilitatActual[0] += probabilitats[paraula][0]
           probabilitatActual[1] += probabilitats[paraula][1]
       else:
           probabilitatActual[0] += probNegatius
           probabilitatActual[1] += probPositius
       probabilitatActual[0] += probs[0]
       probabilitatActual[1] += probs[1]
   if probabilitatActual[0] < probabilitatActual[1]:
       resultat = 1
   else:
       resultat = 0

   if resultat == 0 and resultat == valor:
       trueNeg += 1
   elif resultat == 1 and resultat == valor:
       truePos += 1
   elif resultat == 0 and resultat != valor:
       falseNeg += 1
   else:
       falsePos +=1
print("Accuracy:", (trueNeg + truePos)/(trueNeg+truePos+falseNeg +
falsePos))
print("Precision:", truePos/(truePos+trueNeg))
```

```
print("Recall:", truePos/(truePos+falseNeg))
```

Els resultats d'aquest model no eren gaire bons:
Accuracy: 0.4411174789679597
Precision: 0.9626393732167927
Recall: 0.8486298821278627

Per tal de poder implementar l'algorisme he necessitat fer 2 diccionaris diferents però molt relacionats entre ells:

El primer diccionari es diu cont_paraules_valor. Aquest diccionari actua com a un comptador. Controla la quantitat de vegades que una mateixa paraula apareix en un tweet positiu o un tweet negatiu.

El segon diccionari es diu probabilitats. Aquest diccionari guarda les probabilitats de que una paraula pertanyi a un tweet positiu o negatiu.

Els diccionaris inicialment suposaven un problema de rendiment degust a que un diccionari normal de python es molt lent i allargava molt el temps d'execució de l'algorisme, especialment el diccionari cont_paraules_valor.

Per solucionar-ho he inicialitzat els valors per defecte de tots els valors cosa que ha millorat molt el temps d'execució de l'algorisme.

Respecte als resultats obtinguts, he fet una validació bastant simple degust a que ja sabia d'antelació quie aquest model faria falta millorar-lo per tant hem vaig conformar en calcular la accuraccy, la precisió i recall dels resultats obtinguts pel classificador. Aquests parametres han estat testejats amb un 20% de les dades que no es troben al conjunt de train

Respecte els resultats, es pot veure que quan el clasificador determina que un tweet es positiu, tenim una confiança molt alta en que aquest tweet sigui realment positiu, pero no passa el mateix amb els tweets negatius.

EXERCICI 2

Tal i com podem veure en el exercici anterior el sets de train i de test es poden modificar a partir de la funció del train_test_split:

```
train_test_split(X, y, test_size=0.3, random_state=0)
```

En aquesta funció es pot modificar el número del test_size, on determinem quantes dades son del test i alhora es redueixen el nuemro de files del test.

Modificant aquest valor, sempre obtenim una accuraccy al voltant de 0.35. A mesura que augmentem el valor, redueix es redueix significativament tant la precissió com el recall. Si fem que el test_size sigui 0.2, obtenim una accuracy de 0.44 i obtenim un maxim de precisio del 0.96 i un recall de 0.84.

Respecte el tamany del diccionari, afegint la variable de longitud maxima tal que:

```
max_len = len(X_train)*0.2
```

i afegint la condició de que la longitud del diccinoari sigui menor que el tamany maxim permés:

```
if len(cont_paraules_valor) < max_len:
```

Podem veure que conforme el que multipliquem al len(X_train) es fa més petit, mes augmenta tant la accuracy com el recall o precisió, és a dir. Com menys paraules tenim en el diccionari més correcte és la predicció que realitza.

EXERCICI 3

Per augmentar la accuracy he implementat una millora del tractament de les paraules no vistes anteriorment en el diccinoari, el LaplaceSmoothing. Per aplicar-lo he fet els següents canvis:

```
paraules = list(cont_paraules_valor.keys())
paraulesDesc = [1 / (numParaules[0] + len(paraules)), 1 /
(numParaules[1] + len(paraules))]

for paraula in cont_paraules_valor.keys():
   numPositiu = cont_paraules_valor[paraula][1]
   numNegatiu = cont_paraules_valor[paraula][0]

   auxPos = (numPositiu + 1) / (numParaules[1] + len(paraules))
   auxNeg = (numNegatiu + 1) / (numParaules[0] + len(paraules))

   if paraula not in probabilitats:
       probabilitats[paraula] = [0, 0]
   probabilitats[paraula] = [auxNeg, auxPos]

print('Probabilitats fetes')

truePos = 0
trueNeg = 0
falsePos = 0
falseNeg = 0

for tweet, valor in zip(X_test['tweetText'], y_test):
   prob_pos = 1
   prob_neg = 1

   palabras = tweet.split()
   for palabra in palabras:
       if palabra in probabilitats:
           prob_pos *= probabilitats[palabra][1]
           prob_neg *= probabilitats[palabra][0]
       else:
           prob_pos *= paraulesDesc[1]
           prob_neg *= paraulesDesc[0]
```

Aplicant aquesta millora, tenint un X_train amb el 80% de les dades, obtenim els següents resultats:
Accuracy: 0.833741401795075
Precision: 0.858861734460219
Recall: 0.7962354212635655

Com es pot veure, els resultats han millorat molt, tenim un classificador molt fiable.

Si decidim reduir la quantitat de dades sobre les quals els nostre model entrenarà, els resultats de l'accuraccuracy, precision i recall es mantenen bastant constants fins que reduim el conjunt a la meitat del orginal, aleshores ja comença a perdre fiabilitat conforme reduim les dades sobre les que s'entrena, tot i que el seu recall augmenta molt.

Respecte el limitar el tamany del diccionari, en el moment que limitem una mica el màxim numero de paraules que pot tindre, l'acuraccy i la precisió baixen de la següent manera:
Accuracy: 0.6084816748189035
Precision: 0.5619366208572493
Recall: 0.9837878815556327

a mesura que reduim el numero, el error es mante mes o menys constant.