

Spamizer

Marc Sàncchez, Francesc Xavier Bullich, Gil Gassó

5/8/2019

Contents

Naive Bayes	2
Assumpcions.	3
Punts forts i febles del mètode de Naive Bayes.	4
Aplicació.	4
Tecnologies escollides.	4
Manual de l'aplicació.	5
Implementació i exemples d'execució.	6
Lectura de fitxers.	6
Mètode de selecció.	6
Filtre i abstracció del filtratge.	7
Entrenament.	7
Validació.	8
Compute.	8
Fase Experimental.	9
Càcul del TCR (Total Cost Ratio)	9
Anàlisi de la PHI i la K.	10
Què passa quan lemmatitzem.	14
Comparativa amb un altre programa	15
Referències	17

```

# x és el nom del fitxer que volem carregar
loadFormattedData <- function(x){

  tmp = read.csv(x)
  names(tmp) <- c("id", "phi", "k", "tp", "tn", "fp", "fn", "nham", "nspam")

  #Calculem els tcr dels valors
  # BASE : (NSPAM) / (50 * NHAM + NSPAM)
  base <- tmp$nspam / (50 * tmp$nham + tmp$nspam)
  # WERR: (50 * FP + FN)/(50 * NHAM + NSPAM) + 0.000001 -> per que no sigui 0
  werr <- (50 * tmp$fp + tmp$fn) / (50 * tmp$nham + tmp$nspam) + 0.000001
  # TCR : BASE / WERR
  tcr <- base/werr

  # Calculem l'accuracy
  accuracy <- (tmp$nspam + tmp$nham - tmp$fp - tmp$fn)/(tmp$nspam + tmp$nham) * 100

  # Generar una matriu que permeti representar els resultats en funció de k i phi
  values <- data.frame(accuracy, tcr)
  names(values) <- c("accuracy", "tcr")
  head(values)

  tmp <- cbind(tmp, values)

  # Ordenem els valors
  tmp <- tmp[order(-tmp$tcr), ]

  return(tmp)
}

```

Naive Bayes

En aquest apartat s'especifica com s'adapta el mètode de naive bayes al filtratge de correu.

Descripció del codi amb el qual implementem el mètode Naive Bayes.

Codi implementat:

```

public boolean isSpam(MemDB memDB, Collection<String> message, double k, double phi) {
    double spamProbability=0;
    double hamProbability=0;

    int hamAlphabet = memDB.getCountAlphabet(TableEnumeration.Table.HAM);
    int spamAlphabet = memDB.getCountAlphabet(TableEnumeration.Table.SPAM);

    List<String> words = new ArrayList<>(message);
    hamProbability = memDB.calculateProbability(words, TableEnumeration.Table.HAM,k,hamAlphabet);
    spamProbability = memDB.calculateProbability(words, TableEnumeration.Table.SPAM,k,spamAlphabet)

    double pTotalSpam = memDB.getMessageProbabylity(MemDB.Column.SPAM,k);
    double pTotalHam = memDB.getMessageProbabylity(MemDB.Column.HAM,k);

    //Tenim la probabilitat sumada de cada una de les paraules del missatge en forma de logaritme

```

```

    //sumem les probabilitats de que els missatges siguin spam o ham en forma de logaritme
    hamProbability += pTotalHam;
    spamProbability += pTotalSpam;

    //comparam les probabilitats obtingudes aplicant el parametre phi.
    return (spamProbability) > ((hamProbability)+ Math.log(phi));

}

```

Descripció pas a pas de les variables i funcions implementades.

Primer de tot el que fem és declarar dues variables que ens servirà per guardar tan la suma de probabilitats de cada paraula de que sigui spam o ham.

```

double spamProbability=0;
double hamProbability=0;

```

Aquí el que fem és declarar dos enters que ens guarden el total de paraules que tenim registrades en la base de dades en memòria com a spam i com a ham.

```

int hamAlphabet = memDB.getCountAlphabet(TableEnumeration.Table.HAM);
int spamAlphabet = memDB.getCountAlphabet(TableEnumeration.Table.SPAM);

```

Assignem els valors a les probabilitats de ham i spam. Sumem la probabilitat de cadascuna de les paraules del missatge de que siguin ham o spam.

```

List<String> words = new ArrayList<>(message);
hamProbability = memDB.calculateProbability(words, TableEnumeration.Table.HAM,k,hamAlphabet);
spamProbability = memDB.calculateProbability(words, TableEnumeration.Table.SPAM,k,spamAlphabet);

```

Declarem dos variables que contindran els valors de la probabilitat total de que un missatge sigui spam o sigui ham.

```

double pTotalSpam = memDB.getMessageProbabylity(MemDB.Column.SPAM,k);
double pTotalHam = memDB.getMessageProbabylity(MemDB.Column.HAM,k);

```

Tan la suma de probabilitats de cada paraula del missatge com la probabilitat del missatge de ser spam o ham, estàn logaritzades, i per tant el que fem al final és sumar aquestes dades.

```

//Tenim la probabilitat sumada de cada una de les paraules del missatge en forma de logaritme
//sumem les probabilitats de que els missatges siguin spam o ham en forma de logaritme
hamProbability += pTotalHam;
spamProbability += pTotalSpam;

```

Finalment apliquem la fórmula amb phi quan ens permet donar importància al fet de no classificar missatges Ham com a Spam, per tant com que volem estar segurs d'això fem el següent: mirem que per classificar un missatge com a Spam, la probabilitat de ser spam ha de ser més gran que phi vegades la probabilitat de que sigui ham.

```
(spamProbability) > ((hamProbability)+ Math.log(phi));
```

Assumpcions.

El mètode de Naive Bayes assumeix que les característiques dels elements que analitza són independents entre si.

En el cas de processament de textos assumeix que cada una de les paraules d'un mateix bloc són independents de la resta. Això obviament no és cert. Les frases es construeixen a partir d'una gramàtica i tenen una

semàntica que relaciona cada una de les paraules que conté amb les altres. Per tant el fet de trobar X paraula al costat de Y és degut a una dependència entre elles.

Si consideréssim les dependències entre els elements, segurament només podríem classificar exemples idèntics als que ja tenim classificats, per tant seria totalment inútil.

D'altra banda si considerem que les característiques són independents fa que puguem comptabilitzar més cops elements que de forma conjunta (depenent) només comptabilitzaríem 1 sol cop.

En l'exemple dels correus:

- Si tenim en compte que les paraules d'una frase són dependents, trobar 1 frase que només surti a ham ens incrementa la probabilitat que el missatge sigui ham
- Si pel contrari assumim la independència entre paraules. Comptabilitzem cada una de les paraules de la frase, incrementant la probabilitat que sigui ham (aquí es pot veure la ingenuïtat del mètode).

Punts forts i febles del mètode de Naive Bayes.

Punts forts:

- L'algoritme és prou fàcil de construir i d'entendre.
- És un algoritme prou ràpid i dóna prou bons resultats.
- Es pot entrenar amb un conjunt relativament petit d'exemples.
- En els casos que es compleixi l'assumpció d'independència el mètode funciona millor.

Punts febles:

- L'assumpció d'independència dels exemples pot fer que la classificació no sigui correcte si no té una bona base d'exemples (ben diferenciats)
- El conjunt d'exemples ben diferenciat és difícil de trobar fora del món acadèmic.
- Les característiques que no surtin al conjunt d'entrenament no es poden classificar correctament, ja que la probabilitat que assigna el mètode és 0.
- Si s'entrena malament la màquina, la màquina acaba enganyant.
- Si hi ha molta diferència en nombre d'elements entre els diferents bag of words podem trobar inconsistències en els resultats.

Applicació.

S'ha preparat el programa per que sigui una instrucció en java i per que el seu us sigui mitjançant un java -jar.

Consulteu el github del projecte.

Tecnologies escollides.

L'aplicació està feta en java en un format de comanda c. Per veure com s'executa l'aplicació es pot consultar l'apartat manual de l'aplicació. La idea és generar un paquet jar i que rebi paràmetres.

S'han aplicat patrons de disseny de software tals com el patró strategy i el patró singleton. El patró strategy s'aplica en general a totes les possibles operacions que poden ser canviades en temps d'execució mentre que el patró singleton s'aplica als accessos a les bases de dades.

L'aplicació conté 2 bases de dades on es desa la informació relativa al filtre. Una base de dades en memòria i una base de dades desada en local en un o diversos fitxers.

A causa del nombre de dades que es processen a casa execució s'han replantejat les tecnologies escollides.

Versió 1

La primera versió de l'aplicació i després de consultar diferents projectes relacionats amb el machine learning es va plantejar mitjançant bases de dades SQL, concretament una base de dades feta amb HSQLDB en memòria i una base de dades també HSQLSB desada en un fitxer.

Versió 2

Per culpa de la gran quantitat de dades que s'havia de processar i localitzant un coll d'ampolla en les operacions de logarimitzar que s'havien de fer sobre les dades extretes dins de les mateixes sentències SQL, s'ha implementat una segona versió mitjançant hashmaps i mitjançant fitxers .csv (Comma separated values).

La versió 2 corregeix la lentitud de l'accés a les dades. Utilitza un mapa clau valor per les aparicions de les paraules de ham i un mapa clau valor per les aparicions de les paraules d'spam. Així mateix els valors del total de missatges de ham i spam són comptadors i l'alphabet es representa amb un Set, és a dir, amb un conjunt de paraules.

Manual de l'aplicació.

L'aplicació és un programa en java executable amb la instrucció java -jar. El seu manual es llista a continuació :

```
usage: main/java/spamizer
-c <arg>  Usage : -c <spamDir> <hamDir> [-n <int>]
           Receives 2 parameters, A directory with spam mails and a
           directory with ham mails. A calculation for values phi and k
           will be done using a selection for the mails set. By default
           the selection will be random based on k-fold cross-validation
           and the heuristic method used to calculate phi and k values
           will be random
-d          Flag that indicates that data must be loaded from local
           database, this database is allocated inside project dir named
           db made by csv files
-h          Set training mails as ham, adding this argument -s must not be
           present
-l          Uses Stanford core lib as lemmatizer for all, training and
           validation.
-n <arg>   The number of iterations for -c mode execution.
-o          Uses a ordered method to select and insert mails, using this
           option you can insert same mails ordre for all
           executions. Custom K-fold cross-validation by default.
-p          Set the persistance of the memory database to a local database
-s          Set training mails as spam, adding this argument -h must not
           be present
-t <arg>   Directories where training mails in txt are stored, this or
           database argument must be present you can set a maximum of 2
           directories in this several order : -t <spamDir> <hamDir>. If
           only one dir is set the parameter -h or -s must be included
-v <arg>   Directory where validation mails in txt are stored. This
           procedure will validate mail inside validationDir with
           database loaded by default or stored inside memory. [-h | -s]
           -v <validationDir> .
```

Implementació i exemples d'execució.

En aquest apartat s'expliquen els blocs amb què s'ha dividit la part d'enginyeria del software aplicada a aquesta pràctica, en concret els següents apartats mostren els patrons strategy.

Lectura de fitxers.

La interfície reader proporciona l'obligació d'implementar el mètode que permet llegir i extreure textos d'una font de dades.

De la mateixa manera s'ha generat el DirectoryMailReader que implementa la interfície Reader i que és la classe que gestiona l'accés a un directori en concret per extreure'n un llistat de correus.

Mètode de selecció.

La interfície Selector és la que s'utilitza per separar el conjunt d'entrenament i de validació. Al ser una interfície, permet canviar el mètode de selecció en temps d'execució i fent fàcil la incorporació de nous mètodes de selecció.

Aquest mètode entraria només a l'apartat -c de les execucions. Donades dues col·leccions de correus és capaç d'extreure un subconjunt per spam i un subconjunt per ham deixant un tercer subconjunt com a correus per validar. Són objectes correu i per tant, com a objecte correu, ja s'incorpora un camp booleà que estipula si el correu és spam o ham. Aquest booleà s'utilitza al final de la validació per comptabilitzar el nombre de correus classificats com a tp, fp, tn i fn.

Adaptació del mètode K-fold cross-validation.

S'utilitza aquest mètode de selecció per defecte. Si no se li especifica la opció -o utilitza aquest.

El mètode k-fold cross-validation aplica una divisió del total d'objectes de la població en n parts. Donada aquesta divisió el mateix mètode va rotant, entrenant la BD amb n-1 parts i posteriorment aplica el mètode de validació per la part que no s'ha fet servir per entrenar.

En el nostre cas tot i anomenar-lo k-fold cross-validation, **sabem que no ho és**, l'anomenem així per què ens vam basar en ell per definir el nostre mètode de selecció.

Aquest consisteix en discriminar de la lectura un percentatge de correus (en el nostre cas entre el 5% i el 15% generat aleatoriament) del total de cada directori i posar-lo en una altra col·lecció anomenada unknown. Aquesta col·lecció serà utilitzada per a la validació mentre que la resta s'utilitzen per training. Val a dir que la selecció es fa mitjançant la generació d'un nombre aleatori. Si el nombre és inferior al % generat el classifiquem com a unknown, en canvi si és superior o igual el classifiquem com spam o ham en funció del directori que s'estigui processant.

Fixed Selection

La selecció fixada ha sigut implementada per a poder calcular més acotadament els valors de phi i k. La selecció fixada sempre processa els mateixos correus amb el mateix ordre i discrimina els mateixos. D'aquesta manera podem generar execucions idèntiques amb valors diferents de les constants phi i k. Ens és molt útil per estudiar el seu comportament.

Per utilitzar aquesta opció de selecció es fa servir el paràmetre -o.

```
java -jar spamizer.jar -o -n 1 -c
"/home/marc/Escriptori/test/all/SPAM_TRAINING"
"/home/marc/Escriptori/test/all/HAM_TRAINING"
```

D'aquesta manera s'ordenaran els correus mitjançant el nom del fitxer i es podran repetir les insercions, serveix per poder fixar els paràmetres de phi i k i veure com responen.

Filtre i abstracció del filtratge.

La interfície Filter permet generalitzar el filtratge de text procedent de cada correu electrònic. Fent que es pugui canviar de filtre en temps d'execució. També millora l'escalabilitat permetent que se'n puguin afegir de nous només implementant aquesta interfície.

Stanford Core NLP.

La gent de Stanford ens proporciona una gran llibreria per al processament de textos. Veure la referència [2]. La llibreria s'anomena Stanford Core NLP lib. Dins d'ella s'implementa un mètode de “tokenització”, és a dir generació de tokens per paraules que permeten estructurar el text i realitzar diferents comprovacions, com ara:

- Saber la categoria gramatical de la paraula (Nom, verb, determinant...)
- Posicionament de les paraules dins de paràgrafs, frases, expressions...
- Permet la **Lematització de les paraules**, és a dir cercar l'arrel d'aquestes.
- Anàlisi de sentiments de les expressions dient si una frase és positiva, negativa ...
- Extreure subjecte, verb i predicats...

Entre altres.

Ens hauria agradat explotar més aquesta llibreria però el temps per a la realització de la pràctica i el seu mateix objectiu ens allunyava de la possibilitat d'estudiar-la més a fons, concretament el nostre filtre basat en StanfordCoreNLP realitza una discriminació de totes les paraules que no són noms, verbs, adjectius, adverbis i lematitza tot el que li passen.

Més endavant hi ha conclusions sobre l'ús de la mateixa llibreria.

Per utilitzar el filtre lematitzador d'stanford s'ha d'especificar a la instrucció mitjançant el paràmetre “-l”.

```
java -jar spamizer.jar -l -n 1 -c
"/home/marc/Escriptori/test/all/SPAM_TRAINING"
"/home/marc/Escriptori/test/all/HAM_TRAINING"
```

Custom Filter.

És el filtre utilitzat per defecte, si no se li especifica la opció -l fa servir aquest.

Aquest filtre simplement deixa passar tot el que se li dóna sempre que cada paraula compleixi amb el seu invariant semàntic, és a dir, que tingui algun caràcter. Extreu també les paraules “Subject:”, “cc”, “from” i “to”, que tot i que en els correus de prova no hi són, en altres paquets de correus sí que pot ser que ens els trobem. També filtra els salt de línia i retorn de carro.

Entrenament.

S'implementa a la classe Trainer.

L'entrenament es pot efectuar de dues maneres, mitjançant la instrucció amb el paràmetre -t o bé amb la intrucció i el paràmetre -c que realitza un entrenament per cada iteració. Més endavant es parla de la fase compute que engloba l'entrenament i la validació.

A la fase d'entrenament s'emplen els bag of words i es realitza el comptatge en memòria de les paraules que s'han inserit tant per spam com per ham. Per realitzar una fase d'entrenament i desar l'entrenament per posteriors execucions es pot fer mitjançant la següent combinació.

```
java -jar spamizer.jar -p -t
"/home/marc/Escriptori/test/all/SPAM_TRAINING"
"/home/marc/Escriptori/test/all/HAM_TRAINING"
```

La opció -p serveix com a opció persist. Llegirà la base de dades local que tinguem emplenada amb les execucions anteriors i farà un merge amb l'entrenament que tingui realitzat en memòria. En cas que no hi hagi una base de dades local la generarà amb fitxers csv.

El paràmetre -t estipula que s'està entrenant i tal i com diu el manual de l'aplicació l'ordre dels directoris importa, sempre va primer el directori spam.

Validació.

S'implementa a la classe Validator.

La classe validator hereda directament de Trainer, per tant un validador també ens pot servir per entrenar, s'ha especificat així per aconseguir alleugerir el codi i per que les fases puguin ser més modulars.

Aquesta fase estipula l'ús del mètode NaiveBayes per validar si un correu és spam o ham. mitjançant uns contadors sap veure el nombre de True positive, true negative, false positive i false negative resultat de l'avaluació dels diferents correus i actualitza la variable global Results amb les dades de la validació. Per tant aquesta classe a part de fer el training fa el procés de validar els correus llençant el mètode Naive Bayes

Compute.

Durant la la fase experimental al desenvolupament ens vam adonar que necessitàvem una manera òptima de poder generar valors de k i de phi. Llavors vam inventar aquest mètode per tractar k i phi com a variables dins d'una execució del nostre filtre.

El que permet és assignar-li, mitjançant el paràmetre -n un nombre d'execucions, els resultats de les quals es desen a un fitxer csv anomenat results.csv. Aquests son els fitxers que s'utilitzen més endavant per a l'evaluació de la fase experimental.

Mitjançant el paràmetre -c proporcionant valors per el paràmetre -n i directoris d'spam i de ham, el nostre filtre de correu farà -n iteracions amb -n valors per k i -n valors per phi. Els valors de k i phi seran generats amb k entre 0.0000001 i 1 i phi entre 1 i 5 de manera aleatòria.

Un exemple d'execució seria :

```
java -jar spamizer.jar -n 10 -c
"/home/marc/Escriptori/test/all/SPAM_TRAINING"
"/home/marc/Escriptori/test/all/HAM_TRAINING"
```

Que realitza 10 iteracions amb 10 filtratges diferents i 10 execucions totalment diferents.

Es pot customitzar l'execució del paràmetre -c mitjançant els paràmetres -o i -l. El paràmetre -o estipula si l'entrenament i la separació dels correus per validar ha de tenir un ordre en concret, en cas que -o aparegui els fitxers s'ordenaran de manera alfabetica permetent generar la mateixa execució sempre amb diferents valors de k i phi. Molt útil per veure com oscil·len aquests valors.

El paràmetre -l permet lematitzar la inserció de les paraules i la lectura dels correus a validar mitjançant la llibreria StanfordCoreNLP [2].

Per defecte s'utilitza una selecció que n'anomenem Custom K-Fold cross-Validation (ja que no és un k-fold però la idea va sorgir d'allà) i la informació es filtra amb el filtre Custom, descrit anteriorment.

```
java -jar spamizer.jar -o -l -n 10 -c
"/home/marc/Escriptori/test/all/SPAM_TRAINING"
"/home/marc/Escriptori/test/all/HAM_TRAINING"
```

Fase Experimental.

En aquesta fase es presenten una sèrie de gràfics i una anàlisi (no confirmat) sobre totes les execucions que s'han anat realitzant.

Càcul del TCR (Total Cost Ratio)

Amb el Total cost ratio podem extreure un valor que pondra amb més força el valor de les aparicions dels falsos positius. El que es busca amb el TCR és el valor màxim possible. Per fer-ho hem realitzat diverses execucions i hem preparat una sèrie de conclusions per intentar esbrinar les funcions phi i k que millor s'ajusten al nostre problema mitjançant el càlcul del TCR.

Veiem com es pot generar la columna TCR

```
results = read.csv("./20000m-500n-SF.csv")
#Calculem els tcr dels valors
# BASE : (NSPAM) / (50 * NHAM + NSPAM)
base <- results$NSPAM / (50 * results$NHAM + results$NSPAM)
# WERR: (50 * FP + FN)/(50 * NHAM + NSPAM) + 0.000001 -> per que no sigui 0
werr <- (50 * results$FP + results$FN) / (50 * results$NHAM + results$NSPAM) + 0.000001
# TCR : BASE / WERR
tcr <- base/werr

# Generar una matriu que permeti representar els resultats en funció de k i phi
values <- data.frame(results$PHI, results$K, tcr)
names(values) <- c("phi", "k", "tcr")
head(values)

##          phi         k      tcr
## 1 2.103004 0.8211889 29.954564
## 2 1.140895 1.5680716 11.313981
## 3 3.032053 0.7119040 10.994228
## 4 3.627198 0.1008787 10.737433
## 5 1.033468 0.2739749  9.265549
## 6 1.647332 0.9765125  9.136871

# Ordenem els valors
values <- values[order(-values$tcr), ]
head(values, 20)

##          phi         k      tcr
## 1298 2.697174 0.22600527 33.67084
```

```
## 1120 1.236293 2.34200744 30.15416
## 1 2.103004 0.82118893 29.95456
## 1058 1.879062 0.02760826 29.90953
## 1491 3.137478 0.23219458 29.04305
## 1355 2.798966 0.09099896 27.00852
## 576 1.924919 0.38421715 26.20279
## 930 4.382425 2.86167496 25.13894
## 1332 2.300543 0.34994918 24.90058
## 1205 2.277744 2.09654442 23.51241
## 1184 4.532676 2.01806091 21.53804
## 1267 3.261305 0.99216020 20.27118
## 1179 4.653059 2.43772195 18.95931
## 1172 3.035856 1.56213014 18.69684
## 585 3.352996 1.72896730 17.93125
## 1242 4.096789 2.80409060 15.30624
## 582 2.452848 0.24901731 13.80774
## 988 4.544143 0.48137259 13.14401
## 651 2.694429 0.84360478 11.99221
## 1424 1.741707 2.31180243 11.79649
```

Anàlisi de la PHI i la K.

Carreguem les diferents simulacions en un dataframe per poder processar les dades. Utilitzem la funció loadFormattedData declarada a l'apartat de funcions del document. Aquesta funció ens afegeix les columnes calculades per l'accuracy i el TCR.

```
b1 = loadFormattedData("./m20000-n9500-SF-P-16-k-03.csv")
b2 = loadFormattedData("./m20000-n10000-P-15-K-03.csv")
b3 = loadFormattedData("./20000m-500n-SF.csv")
b4 = loadFormattedData("./2000m-1000n-phi-1.7-2.3-k-0-0.5.csv")
```

```
v <- rbind(b1, b2, b3, b4)
v <- v[order(-v$tcr), ]
head(v)
```

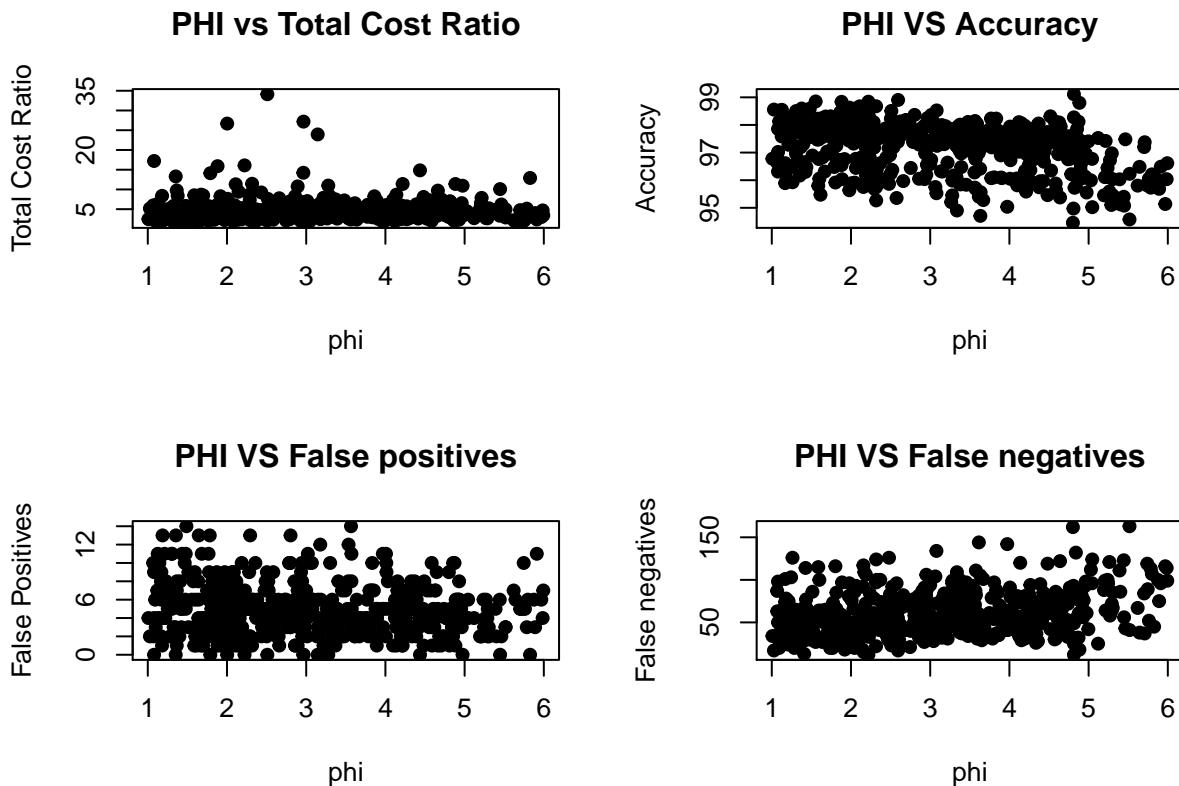
```
##      id     phi      k   tp   tn fp fn nham nspam accuracy      tcr
## 66031 8637 1.778913 0.5925286 695 625  0 11  695   636 99.17355 57.63278
## 26510 2299 1.362548 0.6460839 688 644  0 15  688   659 98.88641 43.83089
## 14114 1064 2.179125 0.2739705 607 542  0 13  607   555 98.88124 42.59106
## 38981 5932 1.734514 1.0386465 694 666  0 16  694   682 98.83721 42.53095
## 20121 4046 2.723349 0.4087232 836 740  0 18  836   758 98.87077 42.01178
## 59141 7948 4.616297 0.4507945 1075 948  0 25 1075   973 98.77930 38.83499
```

PHI

El valor mínim que té sentit assignar-li a phi és 1 i el màxim el podríem limitar a 5 com a molt o inclús a 6 si el que volem és no tenir cap correu que sigui Ham i que el consideri com Spam. Aquest paràmetre se'l pot considerar més influent que el valor de k, ja que el valor de phi està directament lligat al nombre de falsos positius i de falsos negatius. En canvi el valor de k representa un coeficient molt baix a aplicar a totes les paraules.

Veiem els següents diagrames de dispersió donada una mostra de 500 punts sobre el total de les execucions.

```
m <- v[sample(nrow(v), size = 500), ]  
  
par(mfrow=c(2,2))  
plot(m$phi, m$tcr, main="PHI vs Total Cost Ratio",  
     xlab="phi", ylab="Total Cost Ratio", pch=19)  
  
plot(m$phi, m$accuracy, main="PHI VS Accuracy",  
     xlab="phi", ylab="Accuracy", pch=19)  
  
plot(m$phi, m$fp, main="PHI VS False positives",  
     xlab="phi", ylab="False Positives", pch=19)  
  
plot(m$phi, m$fn, main="PHI VS False negatives",  
     xlab="phi", ylab="False negatives", pch=19)
```



En l'anterior grid podem veure diferents comparacions del comportament de la variable phi sobre una mostra de 500 elements dins del conjunt total de les execucions. Dels gràfics anteriors podem extreure certes conclusions a vista, tenint en compte que durant les execucions no s'ha fixat en cap moment ni un ordre de lectura de correus, ni un valor per k, ni un valor per phi i els correus per validar eren seleccionats aleatoriament. De totes maneres disposem d'un número molt elevat i amb molta varietat de resultats.

- Es pot veure que la mitjana del tcr queda entre 1 i 6.
- Es pot veure com a més valor de phi, més disminueix el nombre de fp (lentament).
- Es pot veure com a més valor de phi més augmenta el nombre de fn (més pronunciat).
- Es pot veure com l'accuracy és entre el 97 i 99 però que si el valor de phi augmenta llavors l'accuracy baixa 4 punts.

K

Quan apliquem el suavitzat hem de tenir en compte què passa si donats el bag of words de ham i el de spam una paraula no existeix. En la nostra fórmula aquest valor ens proporcionar multiplicacions per 0. Si una paraula no existís en algun dels conjunts, ens podria portar una classificació errònia.

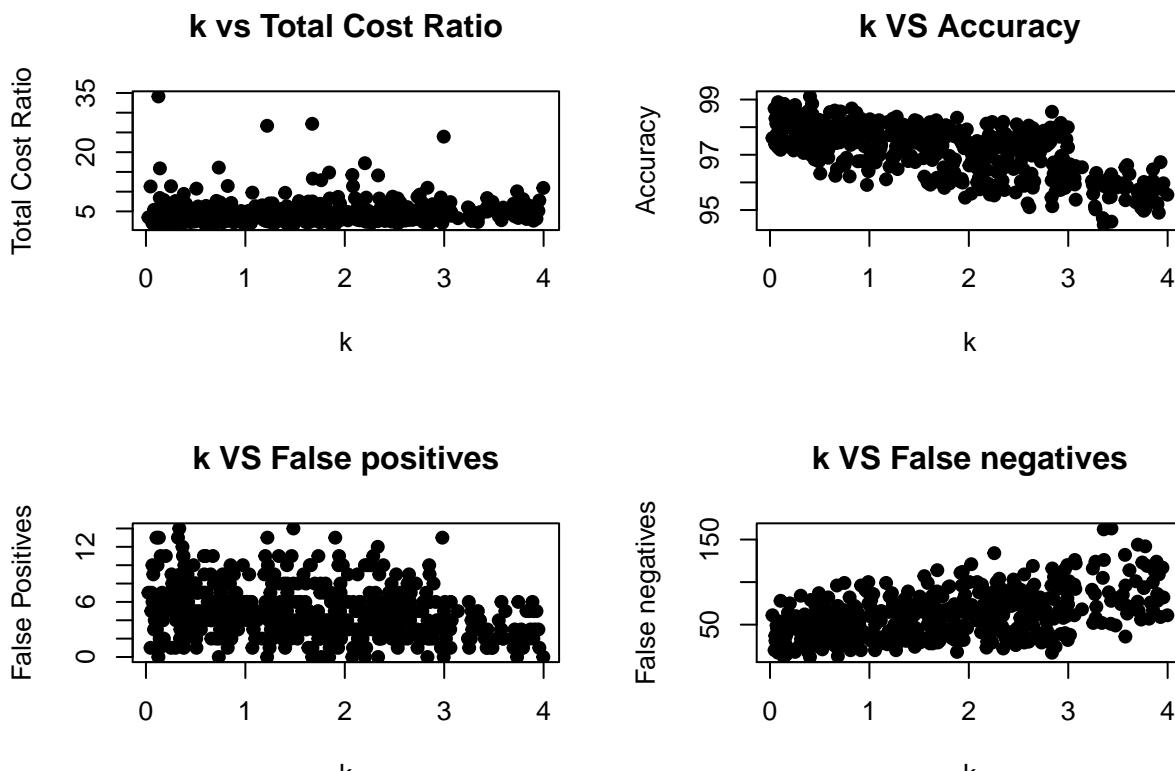
Per tant la k estipula el valor que se li assigna a una paraula quan aquesta no és present. Aquest valor no pot ser 0, però pot ser proper a zero. Si fos zero, la probalitat calculada també seria zero i per tant no podríem classificar correctament. Tanmateix no té sentit aplicar un valor molt gran a la k, ja que si ho féssim, aquest valor provocaria que les paraules que no existeixen fossin puntuades molt altes i es trauria valor de còmput a les aparicions.

```
par(mfrow=c(2,2))
plot(m$k, m$tcr, main="k vs Total Cost Ratio",
     xlab="k", ylab="Total Cost Ratio", pch=19)

plot(m$k, m$accuracy, main="k VS Accuracy",
     xlab="k", ylab="Accuracy", pch=19)

plot(m$k, m$fp, main="k VS False positives",
     xlab="k", ylab="False Positives", pch=19)

plot(m$k, m$fn, main="k VS False negatives",
     xlab="k", ylab="False negatives", pch=19)
```



Utilitzant el mateix supòsit que en la variable phi observem doncs:

- Amb els valors de k pel tcr passa quelcom molt similar als valors de phi.
- Amb els valors de k més petits l'accuracy augmenta, a mesura que es fa créixer el valor de k més

disminueix l'accuracy.

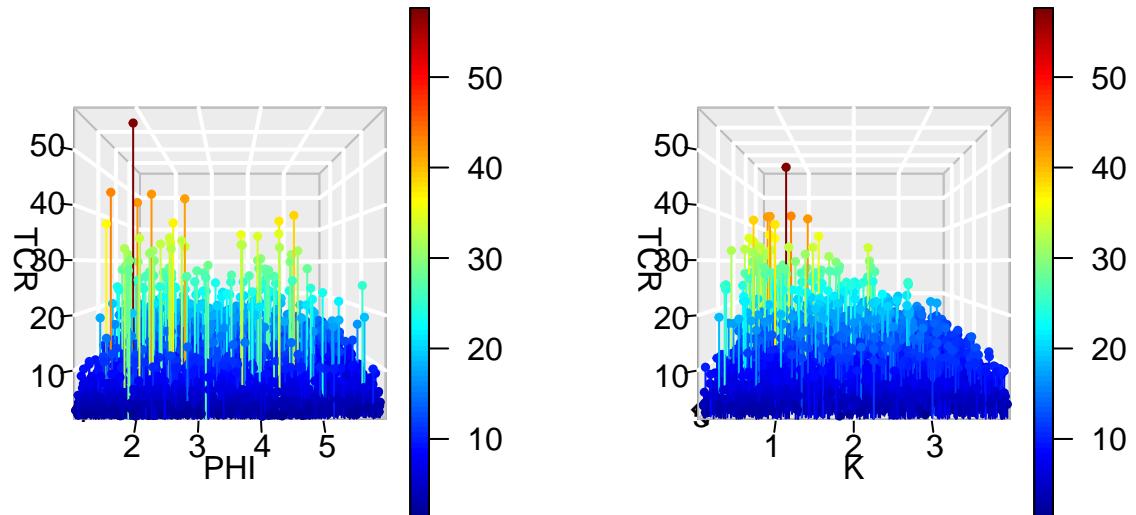
- Veiem que k no impacta molt en els falsos positius.
- Per altra banda veiem que té una relació directa amb el comportament dels falsos negatius. Limitarem els valors de k en un rang de (0 - 1].

Conclusions conjuntes entre phi i k

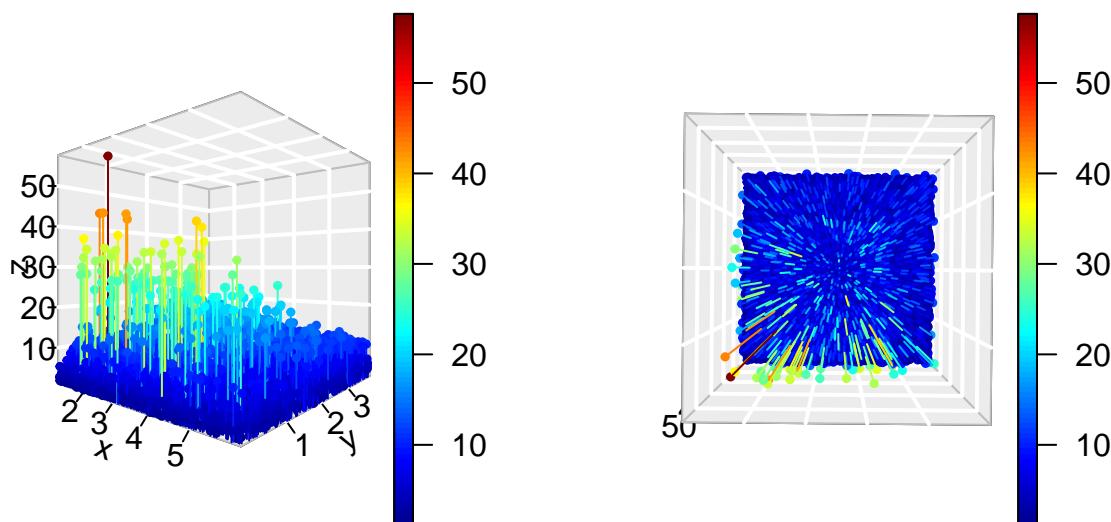
No té sentit mirar les formes dels valors de phi i k de manera independent per què són valors generats aleatoriament. El que sí que té sentit és observar si les variables es poden descriure conjuntament amb el nombre de FP o FN i finalment si es poden comprovar mitjançant el total cost ratio. La variable phi està directament lligada amb els valors FP i FN per definició.

Si fem gràfics en 3D dels valors de k i phi en funció del tcr directament sobre la població d'execucions que tenim podem veure coses interessants.

```
par(mfrow=c(1,2))
scatter3D(v$phi, v$k, v$tcr, phi = 0, theta=0, bty = "g", type = "h", ticktype = "detailed", pch = 19,
scatter3D(v$phi, v$k, v$tcr, phi = 0, theta=90, bty = "g", type = "h", ticktype = "detailed", pch = 19)
```



```
par(mfrow=c(1,2))
scatter3D(v$phi, v$k, v$tcr, phi = 0, bty = "g", type = "h", ticktype = "detailed", pch = 19, cex = 0.5)
scatter3D(v$phi, v$k, v$tcr, phi = 90, theta = 0.5, bty = "g", type = "h", ticktype = "detailed", pch = 19)
```



En els anteriors gràfics es pot distingir perfectament el rang d'actuació dels valors més alts per tcr. De la mateixa manera que s'ha extret les conclusions sobre una mostra, pensant que la mostra seria prou representativa, si observem sobre el total veiem que es confirmen les conclusions que hem anat exposant.

Donat el valor de k entre 0 i 1 i els valors de phi també entre 0.5 i 2.5 és per on es mou el tan buscat màxim global de la funció conjunta. Evidentment no hem formulat una hipòtesi concreta i no l'hem pogut verificar, ja que no hem ajustat el nostre model a cap mostra, però utilitzant el coeficient de correlació de Pearson sobre els gràfics mostrats per phi i per k en les seccions anteriors, podem intuir el comportament de les variables.

No hem d'oblidar però que el valor del tcr s'ha tret directament de l'execució amb les variables phi i k i per tant ens serveix per veure si hi ha algun patró pel qual la funció k o la funció phi de manera independent entre elles poden fer que creixi el valor del tcr. Tanmateix això no serà possible pel fet que el valor del tcr s'estreu tant de la variable phi tant com de la variable k i **S'hauria de fixar o bé la phi o bé la k per poder extreure una conclusió sobre el tema.**

Què passa quan lemmatitzem.

Com hem citat abans, l'ús de la llibreria Stanford Core NLP, ens permet distingir tipus de paraules, trobar l'arrel de les mateixes, etc. N'hem volgut fer un anàlisi tan temporal com en eficàcia, i aquestes són les conclusion a les que hem pogut arribar:

Si ens basem en el temps, hem vist que el fet d'utilitzar el filtre amb la llibreria Stanford Core NLP, augmenta considerablement, parlem de l'ordre de 100 vegades més lent que si no utilitzem un filtratge de paraules.

També podem veure que parlant d'eficàcia, l'utilització de la llibreria Stanford Core NLP, ens dóna resultats amb valors d'accuracy inferiors al 90%. Podem veure un exemple d'execució amb 22.613 missatges:

```
Execution number : 2
Spam number??? : 10678.0
Ham number : 11935.0
Unknown number??? : 2712.0
PHI : 2.239505399350125
K : 0.3731320998425923
Accuracy : 86.09882005899705 %
Tcr : 3.517241369980792
```

Missatge HAM classificat correctament com a HAM: 1386.0

Missatge HAM classificat com a SPAM : 0.0

```
Missatge SPAM classificat correctament com spam : 949.0
Missatge SPAM classificat com a HAM : 377.0
```

Comparant amb un exemple d'execució sense filtre, que ens sol donar valors d'accuracy entre un 97% i 99%, un exemple d'execució seria:

```
Execution number : 3
Spam number : 2989.0
Ham number : 3390.0
Unknwon number : 376.0
PHI : 1.917704909263815
K : 0.5433680794849531
Accuracy : 98.67021276595744 %
Total Cost Ratio : 34.999993000001396
```

```
Missatge HAM classificat correctament com a HAM : 201.0
Missatge HAM classificat com a SPAM : 0.0
Missatge SPAM classificat correctament com spam : 170.0
Missatge SPAM classificat com a HAM : 5.0
```

Comparativa amb un altre programa

Analitzats els nostres resultats, passem a veure una comparativa amb un altre filtre de spam. Utilitzarem Mallet seguint les indicacions de la pràctica.

MALLET [3] és un paquet basat en Java per al processament estadístic del llenguatge natural, classificació de documents, agrupació, modelització de temes, extracció d'informació i altres aplicacions d'aprenentatge automàtic al text.

Entre d'altres, inclou mètodes com Naïve Bayes per a la classificació de documents.

Com funciona Mallet

Mallet té moltes modalitats però només ens centrarem en aquelles que inclouen el processament necessari per realitzar una validació de filtre de spam (classificació de documents).

De forma similar al nostre programa, mallet funciona en mode de comanda des d'un terminal. Hem d'indicar quins són els directoris on hi ha els documents a classificar, i després mitjançant opcions parametritzades, indicar quin serà el mètode de classificació utilitzat.

Carregar els directoris:

```
bin\mallet import-dir --input "emailsENRON\SPAM" "emailsENRON\HAM" --output data.mallet
```

- Opció input: Quan fem import-dir hem d'indicar quins són els directoris on hi ha els documents a carregar (s'en poden indicar diversos).
- Opció output: Fitxer on es guardarà el resultat de la carrega de dades (aquest fitxer ja conte els tokens processats)

Executar validació amb Naïve Bayes:

```
bin\mallet train-classifier --input data.mallet --trainer NaiveBayes --trainingportion 0.9 --num-trials
```

- Opció input: Indica quin es el fitxer d'entrada on hi ha les dades per el proces de classificació (en aquest cas ha d'estar carregat i processat previament)
- Opció trainer: Indica quin tipus de validació s'utilitzara (s'en pot indicar més d'un)
- Opció training-portion: Indica quin percentatge de documents s'utilitzarà d'entrenament i validació. En aquest cas la relació és (90:10)
- Opció num-trials: Indica quantes iteracions de d'entrenament-validació s'executaran.

Si mirem l'apartat d'implementació, podem comprovar que les dues aplicacions funcionen de forma similar, seguint mallet més complet i amb altres funcionalitats que no entren en el context de la pràctica.

Per exemple la nostre opció d'entrenament -t amb persistència -p seria similar al import-dir -input -output.

El mètode de particionat es similiar al que tenim també. S'indica una proporció de documents d'entrenament (0.9), la resta serà pel conjunt de validació. En el nostre cas, com s'ha comentat en l'apartat d'implementació, és un percentatge variable. Mallet també permet la utilització del mètode k-fold-cross-validation, en comptes del particionat aleatori.

```
bin\mallet train-classifier --input data.mallet --trainer NaiveBayes --cross-validation 10
```

Resultats

Per que els resultats siguin similars, utilitzem el mètode de particionat aleatori per fer la comparació.

Aquests són alguns dels resultats obtinguts (No calcula el TCR, s'affageix manualment al final de cada resultat):

```
----- Trial 1 -----  
  
Trial 1 Training NaiveBayesTrainer with 17999 instances  
Trial 1 Training NaiveBayesTrainer finished  
Trial 1 Trainer NaiveBayesTrainer training data accuracy= 0.9869992777376521  
Trial 1 Trainer NaiveBayesTrainer Test Data Confusion Matrix  
Confusion Matrix, row=true, column=predicted accuracy=0.98  
label 0 1 |total  
0 SPAM 981 30 |1011  
1 HAM 10 979 |989  
  
Trial 1 Trainer NaiveBayesTrainer test data accuracy= 0.98  
TCR: 1,907547  
  
----- Trial 2 -----  
  
Trial 2 Training NaiveBayesTrainer with 17999 instances  
Trial 2 Training NaiveBayesTrainer finished  
Trial 2 Trainer NaiveBayesTrainer training data accuracy= 0.9872215123062392  
Trial 2 Trainer NaiveBayesTrainer Test Data Confusion Matrix  
Confusion Matrix, row=true, column=predicted accuracy=0.981  
label 0 1 |total  
0 SPAM 966 32 |998  
1 HAM 6 996 |1002  
  
Trial 2 Trainer NaiveBayesTrainer test data accuracy= 0.981  
TCR: 3,006024  
...  
----- Trial 6 -----
```

```
Trial 6 Training NaiveBayesTrainer with 17999 instances
Trial 6 Training NaiveBayesTrainer finished
Trial 6 Trainer NaiveBayesTrainer training data accuracy= 0.9866103672426246
Trial 6 Trainer NaiveBayesTrainer Test Data Confusion Matrix
Confusion Matrix, row=true, column=predicted accuracy=0.9825
  label  0  1 |total
  0 SPAM 974 34 |1008
  1 HAM   1 991 |992

Trial 6 Trainer NaiveBayesTrainer test data accuracy= 0.9825
TCR: 12
```

Com es pot observar els resultats dels dos filtres son molt similars. Tots dos tenen una mitjana d'accuracy del 98-99%. Pel que fa al TCR veiem que també son força similars, tinguent en compte que només hem fet 10 iteracions. En el millor dels casos de 10 execucions ha tret 1 sol fals positiu.

Les nostres millors execucions mostren un TCR de fins 57 amb accuracy del 99%, després de fer moltíssimes execucions per afinar un valor de phi i k. Mallet no mostra aquests valors per tant no podem comparar-los.

Si tenim en compte la diferència de nombre d'execucions i l'afinament dels paràmetres, comparant els dos programes podem dir que hem fet un bon classificador amb Naïve Bayes.

Referències

1. R graphics
2. Stanford lib
3. Mallet