ADVANCED TEXT PROCESSING WITH SPARK

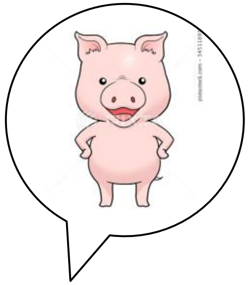
JuHyeong Kim

CONTENT

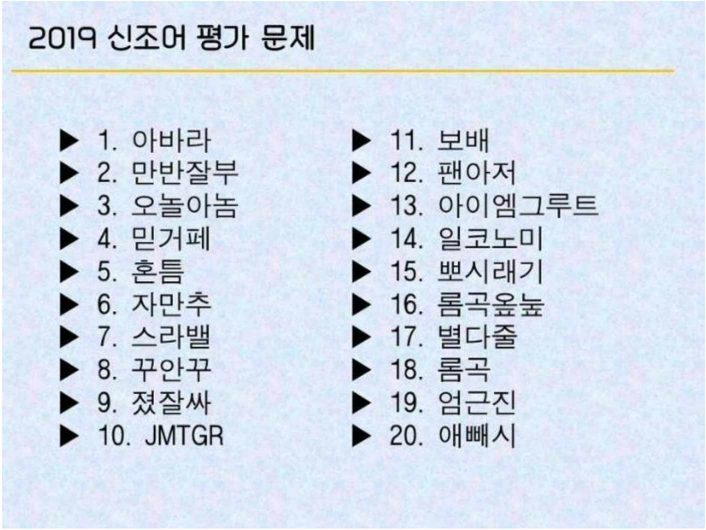
- Key points in TEXT data
- Natural Language Processing(NLP) in SPARK
- Feature extraction technique - TF-IDF, Features hashing
- NLP Processing
  - Description, extraction and analysis of data
  - Pre-processing
  - Building a TF-IDF model
  - Analyzing the TF-IDF weightings
  - Using a TF-IDF model
- Comparing raw features with processed TF-IDF features
- Word2vec 모델과의 결과 비교

KEY POINTS IN TEXT DATA

- Text data can be complex to work with for two main reasons
  - Text and language have an implicit structure that is difficult to grasp.
  - the effective dimensionality of text data is extremely large and potentially limitless. (all English word and special words, characters, slang etc.)



< Common Crawl data set – More than 840 billion words >


2019 신조어 평가 문제

1. 아바라
2. 만반잘부
3. 오놀아놈
4. 믿거페
5. 혼틈
6. 자만추
7. 스라밸
8. 꾸안꾸
9. 졌잘싸
10. JMTGR
11. 보배
12. 팬아저
13. 아이엠그루트
14. 일코노미
15. 뽀시래기
16. 롬곡옾눞
17. 별다줄
18. 롬곡
19. 엄근진
20. 애빼시

< New word >

NATURAL LANGUAGE PROCESSING(NLP) IN SPARK

- we will focus on two feature extraction techniques available within Spark MLlib and Spark ML
- **the term frequency-inverse document frequency (tf-idf)** term weighting scheme and **feature hashing**

FEATURE EXTRACTION TECHNIQUE - TF-IDF, FEATURES HASHING

- TF-IDF
  - $tf - idf$ weights each term in a **piece of text** (referred to as a **document**) based on its frequency in the document (the term frequency).
  - called the inverse document frequency, is then applied based on the frequency of this term among all documents

    (the set of documents in a dataset is commonly referred to as a corpus)

$$tf - idf(t, d) = tf(t, d) * idf(t)$$

  - Here, $tf(t, d)$ is the frequency (number of occurrences) of term t in document d and $idf(t)$ is the inverse document frequency of term t in the corpus

$$idf(t) = \log(\frac{N}{d})$$

  - Here, N is the total number of documents, and d is the number of documents in which the term t occurs.

FEATURE EXTRACTION TECHNIQUE - TF-IDF, FEATURES HASHING

- TF-IDF
  - the $IDF\ normalization(tf - idf)$ has the effect of reducing the weight of terms that are very common across all documents
  - The end result is that **truly rare or important terms** should be assigned **higher weighting**
  - while more common terms (which are assumed to have less importance) should have less impact in terms of weighting.

$$tf - idf(t,d) = tf(t,d) * idf(t)$$

$$idf(t) = \log(\frac{N}{d})$$

Soccer

Computer

BigData

	Frequency	Weight
We - Soccer, Computer, BigData	3	0
kick – Soccer	1	$log3$
spark – BigData	1	$log3$

FEATURE EXTRACTION TECHNIQUE - TF-IDF, FEATURES HASHING

- Feature hashing
  - Input : Data
  - Output : Feature (Frequency)
  - Setup value (hyperparameter) : number of Feature, 2^n
- First, the input data is converted into a hash value
- Second, the hash value is divided by the number of features.
- Finally, the remaining values become feature values.

Color	Hash Function	Divide by	Reminder
Red	36614357519	8	3
Blue	54663777951	8	7
Green	75535549907	8	7

Feature Hashing

Reminder -->	0	1	2	3	4	5	6	7
	Feature 1	Feature 2	Feature 3	Feature 4	Feature 5	Feature 6	Feature 7	Feature 8
Red	0	0	0	**1**	0	0	0	0
Blue	0	0	0	0	0	0	0	**1**
Green	0	0	0	0	0	0	0	**1**

FEATURE EXTRACTION TECHNIQUE - TF-IDF, FEATURES HASHING

- Feature hashing
- Pro
  - Feature hashing has the advantage that **we do not need to build a mapping and keep it in memory**
  - It is also **easy to implement**, **very fast**, and **can be done online and in real time**, **thus not requiring a pass through our dataset first**
  - Finally, because we selected a feature vector dimension that is significantly smaller than the raw dimensionality of our dataset, we bound the memory usage of our model both in training and production; hence, **memory usage does not scale with the size and dimensionality of our data.**
- Con
  - As we don't create a mapping of features to index values, **we also cannot do the reverse mapping of feature index to value**
  - As we are restricting the size of our feature vectors, **we might experience hash collisions**

NLP PROCESSING

- Description, extraction and analysis of data
- Pre-processing
  - Tokenization work - Applying basic tokenization, Improving our tokenization
  - delete Stop word, words based on frequency
- Building a TF-IDF model
- Analyzing the TF-IDF weightings
- Using a TF-IDF model
  - document similarity
  - Training a text classifier

DESCRIPTION, EXTRACTION AND ANALYSIS OF DATA

- we will use a well-known text dataset called **20 Newsgroups** (this dataset is commonly used for text-classification tasks)
- This is a collection of newsgroup messages posted across **20 different topics**
- **training** and **test sets** that comprise **60%** and **40%** of the original data, respectively

- alt.atheism
- comp.graphics
- comp.os.ms-windows.misc
- comp.sys.ibm.pc.hardware
- comp.sys.mac.hardware
- comp.windows.x
- misc.forsale
- rec.autos
- rec.motorcycles
- rec.sport.baseball
- rec.sport.hockey
- sci.crypt
- sci.electronics
- sci.med
- sci.space
- soc.religion.christian
- talk.politics.guns
- talk.politics.mideast
- talk.politics.misc
- talk.religion.misc

Main Category	subclass
alt(?)	atheism
computer	graphics
	ms-windows
	ibm-hardware
	mac-hardware
	window-x
misc	forsale
rec	autos
	motorcycles
	baseball
	hockey
science	crypt
	electronics
	med
	space
religion	christian
	misc
politics	guns
	mideast
	misc

< 20 Newsgroups category >

Question & Answer

```
From: kudla@acm.rpi.edu (Robert Kudla)
Subject: Re: Can I Change "Licensed To" Data in Windows 3.1?
Nntp-Posting-Host: hermes.acm.rpi.edu
Lines: 65

In <0096B130.473B17C0@vms.csd.mu.edu> 2a42dubinski@vms.csd.mu.edu writes:
>       ahh, yes, this is a fun topic.  No, once the name is incribed on the
>disk, that is it, it is encoded.  Not even a HEX editor will find it.  You can

But a disk compare utility (old versus new) will.  And Windows 3.1 is
also flexible enough at install time that you can copy all the files
onto your hard disk, which greatly speeds things up and makes them
less annoying, if you can spare the 7 or so compressed megs.

>write over the "Licensed to:", but you can't change the name underneth it.  I
>think if you wish to change this you would have to be a pirate, and we're not
>going to promote that here.

No, we're not.  But we're also not going to promote pandering to
corporate paranoia when the real issue is convenience.  I don't *like*
dealing with floppies.  Personally, I have no use for changing the
registration info, but I see it as a valid need, and one that ought to
be solved using a quick little utility rather than a half-hour
reinstall that's just about guaranteed to mess up your settings in one
way or another.

So, while I'm not going to put much time into it myself, here's the
procedure for getting on your way to finding the encoded information:

1.  Copy all your Windows disks into the directory from which you want
to install it.  I've been using c:\WINSTALL myself.

2.  From there, copy that directory to something like c:\WINORIG.

3.  Install from c:\winstall.

As I noted before, if you can afford the space on the hard disk, and
don't do much in the way of customization, reinstalling from one
directory to another may be less arduous.  Doing some of the stuff
I've mentioned here may well void your license with Microsoft, as if
they'd ever find out.  If you aren't careful with the disk editor, you
could also mung something important... duh.  I guess that's a
disclaimer.

Have at it....

Rob
--
Rob kudla@acm.rpi.edu Keywords - Oldfield Jane's Leather Yes Win3.1 Phish
light blue right Bondage r.e.m. DTP Steely Dan DS9 FNM OWL Genesis In the
spaceship, the silver spaceship, the lion takes control.....
```

Keywords

< comp.os.ms-windows.misc → 9551 (content) >

DESCRIPTION, EXTRACTION AND ANALYSIS OF DATA

- we will use a well-known text dataset called 20 Newsgroups (this dataset is commonly used for text-classification tasks)
- This is a collection of newsgroup messages posted across 20 different topics
- **training** and **test sets** that comprise **60%** and **40%** of the original data, respectively

```
val sc = new SparkContext( master = "local[2]",  appName = "First Spark App")

val path = "C:\\Users\\KJH\\Desktop\\spark\\20news-bydate-train\\*\\**"
val rdd = sc.wholeTextFiles(path)
val text = rdd.map { case (file, text) => text}
```

```
INFO FileInputFormat: Total input paths to process : 11314
```

< total recodes : 11314 >

```
val newsgroups = rdd.map { case (file, text) =>file.split( regex = "/").takeRight(2).head }
println(newsgroups.first())
val countByGroup = newsgroups.map(n => (n, 1)).reduceByKey(_ +_).collect.sortBy(-_._2).mkString("n")
println(countByGroup)
```

Folder name	File number
rec.sport.hockey	600
soc.religion.christian	599
rec.motorcycles	598
rec.sport.baseball	597
sci.crypt	595
rec.autos	594
sci.med	594
comp.windows.x	593
sci.space	593
sci.electronics	591
comp.os.ms-windows.misc	591
comp.sys.ibm.pc.hardware	590
misc.forsale	585
comp.graphics	584
comp.sys.mac.hardware	578
talk.politics.mideast	564
talk.politics.guns	546
alt.atheism	480
talk.politics.misc	465
talk.religion.misc	377
total	11314

< total recodes : 11314 >

APPLYING BASIC TOKENIZATION

- The first step in our text processing pipeline is to **split up the raw text content in each document into a collection of terms** (also referred to as **tokens**)

```
In <0096B130.473B17C0@vms.csd.mu.edu> 2a42dubinski@vms.csd.mu.edu writes:
```

```
val text = rdd.map { case (file, text) => text }
val whiteSpaceSplit = text.flatMap(t => t.split( regex = " ").map(_.toLowerCase))
println(whiteSpaceSplit.distinct.count)
```

```
402978
```

< total token : 402978 >

APPLYING BASIC TOKENIZATION

- The first step in our text processing pipeline is to **split up the raw text content in each document into a collection of terms** (also referred to as **tokens**)

```
From: kudla@acm.rpi.edu (Robert Kudla)
Subject: Re: Can I Change "Licensed To" Data in Windows 3.1?
Nntp-Posting-Host: hermes.acm.rpi.edu
Lines: 65

In <0096B130.473B17C0@vms.csd.mu.edu> 2a42dubinski@vms.csd.mu.edu writes:
>       ahh, yes, this is a fun topic.  No, once the name is incribed on the
>disk, that is it, it is encoded.  Not even a HEX editor will find it.  You can

But a disk compare utility (old versus new) will.  And Windows 3.1 is
also flexible enough at install time that you can copy all the files
onto your hard disk, which greatly speeds things up and makes them
less annoying, if you can spare the 7 or so compressed megs.

>write over the "Licensed to:", but you can't change the name underneth it.  I
>think if you wish to change this you would have to be a pirate, and we're not
>going to promote that here.

No, we're not.  But we're also not going to promote pandering to
corporate paranoia when the real issue is convenience.  I don't *like*
dealing with floppies.  Personally, I have no use for changing the
registration info, but I see it as a valid need, and one that ought to
be solved using a quick little utility rather than a half-hour
reinstall that's just about guaranteed to mess up your settings in one
way or another.

So, while I'm not going to put much time into it myself, here's the
procedure for getting on your way to finding the encoded information:

1.  Copy all your Windows disks into the directory from which you want
to install it.  I've been using c:\WINSTALL myself.

2.  From there, copy that directory to something like c:\WINORIG.

3.  Install from c:\winstall.

4.  comp the two directories to determine changes.
    i.e., comp *.* \winorig\*.* >\report.txt

5.  Look in the report file for the file(s) that change.  Assuming
they didn't cover themselves covering their own tracks, at least one
file should have a difference noted at a particular offset.  Locate
```

< Document 9551 >

```
you
could,mung,it,phish
light,than,one.

6.,one
directory,have,re:,c:\winorig.

3.,been,underneth,look,wish,who,"real,offset,out.,over,thing,ever,\winorig\*.*,the
procedure,pandering,makes,flexible,65

in,from:,one
way,aren't,information:

1.,less,control.....,finding,file,if
they'd,hex
editor,,little,change.,the,is
also,important...,the
registration,paranoia,not,annoying,,hard,we're,file(s),determine,if,when,be,mess,install;,not.,all,want
to,steely,as,dan,what's,(old,license,going,hex,it's,dtp,fun,it....

rob
--
rob,half-hour
reinstall,ms,need,,but,changes.,utility,joe,versus,space,here.

no,,megs.

>write,kudla@acm.rpi.edu,is,about,spare,to
be,least,incribed,same,cryptography.,even,that's,arduous.,getting,jane's,rather,disk,while,would,on,xor,at,blue,
>going,copy,3.1?
nntp-posting-host:,topic.,disassemble
the,guaranteed,damned,cases,c:\winstall,writes:
>       ahh,,use,may,owl,don't,find,can

but,offset.,pirates",probably,directory,ds9,to",or,don't
really,hermes.acm.rpi.edu
lines:,files
onto,<0096b130.473b17c0@vms.csd.mu.edu>,there,,win3.1,that).

as,guess,no,,code,disk,,more,see,of,void,the
spaceship,,c:\winstall.
```

IMPROVING OUR TOKENIZATION

- The preceding simple approach results in a lot of tokens and does **not filter out many nonword characters** (such as punctuation)
- Most tokenization schemes will **remove these characters(nonword)**

```
val nonWordSplit = text.flatMap(t => t.split( regex = """\W+""").map(_.toLowerCase))
println(nonWordSplit.distinct.count)
```

```
130126
```

```
402978
```

```
130126
```

IMPROVING OUR TOKENIZATION

- The preceding simple approach results in a lot of tokens and does **not filter out many nonword characters** (such as punctuation)
- Most tokenization schemes will **remove these characters**(nonword)

```
you
could,mung,it,phish
light,than,one.

6.,one
directory,have,re:,c:\winorig.

3.,been,underneth,look,wish,who,"real,offset,out.,over,thing,ever,\winorig\*.*,the
procedure,pandering,makes,flexible,65

in,from:,one
way,aren't,information:

1.,less,control.....,finding,file,if
they'd,hex
editor,,little,change.,the,is
also,important...,the
registration,paranoia,not,annoying,,hard,we're,file(s),determine,if,when,be,mess,install;,not.,all,want
to,steely,as,dan,what's,(old,license,going,hex,it's,dtp,fun,it....

rob
--
rob,half-hour
reinstall,ms,need,,but,changes.,utility,joe,versus,space,here.

no,,megs.

>write,kudla@acm.rpi.edu,is,about,spare,to
be,least,incribed,same,cryptography.,even,that's,arduous.,getting,jane's,rather,disk,while,would,on,xor,at,blue,
>going,copy,3.1?
nntp-posting-host:,topic.,disassemble
the,guaranteed,damned,cases,c:\winstall,writes:
>       ahh,,use,may,owl,don't,find,can

but,offset.,pirates",probably,directory,ds9,to",or,don't
really,hermes.acm.rpi.edu
lines:,files
onto,<0096b130.473b17c0@vms.csd.mu.edu>,there,,win3.1,that).

as,guess,no,,code,disk,,more,see,of,void,the
spaceship,,c:\winstall.
```

```
than,old,old,phish,phish,look,wish,who,over,over,thing,thing,
ever,onto,pandering,pandering,makes,makes,procedure,duh,out,out,
out,less,finding,file,file,little,little,personally,the,the,
paranoia,not,host,hard,hard,hard,determine,if,if,when,when,
2a42dubinski,all,all,vms,steely,dan,going,light,license,license,
reinstall,reinstall,reinstall,reinstall,hex,kudla,fun,fun,fun,
ms,them,them,but,versus,rob,rob,jane,space,space,dealing,is,about,
least,least,least,incribed,same,same,tracks,tracks,even,getting,
rather,rather,disk,disk,would,would,before,xor,at,topic,blue,using,
didn,didn,speeds,new,guaranteed,damned,damned,subject,cases,m,m,may,
use,use,owl,owl,owl,s,pirates,pirates,probably,files,0096b130,nntp,
nntp,nntp,need,need,need,personalizing,personalizing,aren,code,code,
locate,locate,more,see,void,void,void,product,product,noted,valid,
technique,registration,this,this,particular,particular,original,
reinstalling,stuff,themselves,right,right,right,i,i,here,some,which,
which,which,once,once,also,also,also,also,doing,doing,disassemble,
disassemble,install,lion,winorig,much,posting,his,his,up,up,issue,
issue,put,lines,enough,enough,can,can,can,half,obnoxious,solved,do,
do,anyway,anyway,leather,leather,no,no,fnm,fnm,fnm,said,said,real,
real,extreme,goes,goes,annoying,annoying,something,something,like,
far,an,an,an,an,own,keywords,keywords,keywords,keywords,keywords,r,
r,r,compressed,compressed,473b17c0,473b17c0,4,takes,microsoft,7,hour,
mentioned,quick,quick,quick,one,one,one,one,one,with,with,with,schmoe,
2,2,data,data,writes,in,3,3,silver,oldfield,ought,ought,ought,genesis,
promote,megs,megs,megs,csd,csd,csd,cryptography,able,able,able,able,well,
well,way,way,way,way,change,two,key,key,key,you,yes,name,another,another,
another,a,don,t,that,work,win3,win3,will,will,their,bondage,office,office,
office,office,floppies,floppies,changes,changes,and,covering,covering
```

< total word in 9551 - 288 > 288

IMPROVING OUR TOKENIZATION

- the next step in our pipeline will be **to filter out numbers and tokens** that are words **mixed with numbers**

```
val regex = """[^0-9]*""".r
val filterNumbers = nonWordSplit.filter(token => regex.pattern.matcher(token).matches)
println(filterNumbers.distinct.count)
```

↓

```
84912
```

402978 → 130126 → 84912

IMPROVING OUR TOKENIZATION

- the next step in our pipeline will be **to filter out numbers and tokens** that are words mixed with numbers

```
than,old,old,phish,phish,look,wish,who,over,over,thing,thing,
ever,onto,pandering,pandering,makes,makes,procedure,duh,out,out,
out,less,finding,file,file,little,little,personally,the,the,
paranoia,not,host,hard,hard,hard,determine,if,if,when,when,
2a42dubinski,all,all,vms,steely,dan,going,light,license,license,
reinstall,reinstall,reinstall,reinstall,hex,kudla,fun,fun,fun,
ms,them,them,but,versus,rob,rob,jane,space,space,dealing,is,about,
least,least,least,incribed,same,same,tracks,tracks,even,getting,
rather,rather,disk,disk,would,would,before,xor,at,topic,blue,using,
didn,didn,speeds,new,guaranteed,damned,damned,subject,cases,m,m,may,
use,use,owl,owl,owl,s,pirates,pirates,probably,files,0096b130,nntp,
nntp,nntp,need,need,need,personalizing,personalizing,aren,code,code,
locate,locate,more,see,void,void,void,product,product,noted,valid,
technique,registration,this,this,particular,particular,original,
reinstalling,stuff,themselves,right,right,right,i,i,here,some,which,
which,which,once,once,also,also,also,also,doing,doing,disassemble,
disassemble,install,lion,winorig,much,posting,his,his,up,up,issue,
issue,put,lines,enough,enough,can,can,can,half,obnoxious,solved,do,
do,anyway,anyway,leather,leather,no,no,fnm,fnm,fnm,said,said,real,
real,extreme,goes,goes,annoying,annoying,something,something,like,
far,an,an,an,an,own,keywords,keywords,keywords,keywords,keywords,r,
r,r,compressed,compressed,473b17c0,473b17c0,4,takes,microsoft,7,hour,
mentioned,quick,quick,quick,one,one,one,one,one,with,with,with,schmoe,
2,2,data,data,writes,in,3,3,silver,oldfield,ought,ought,ought,genesis,
promote,megs,megs,megs,csd,csd,csd,cryptography,able,able,able,able,well,
well,way,way,way,way,change,two,key,key,key,you,yes,name,another,another,
another,a,don,t,that,work,win3,win3,will,will,their,bondage,office,office,
office,office,floppies,floppies,changes,changes,and,covering,covering
```

→

```
mung,than,than,have,old,old,old,old,disclaimer,
disclaimer,disclaimer,could,we,we,we,we,been,been,
who,offset,ever,onto,customization,any,makes,makes,
procedure,duh,duh,finding,file,little,personally,
paranoia,not,not,hard,determine,determine,when,mess,
mess,mu,all,all,dan,dan,reinstall,reinstall,reinstall,
reinstall,hex,kudla,fun,joe,joe,but,myself,myself,myself,
utility,is,is,about,spare,spare,least,incribed,same,even,
on,on,getting,disk,disk,disk,while,before,topic,topic,blue,
blue,using,didn,copy,copy,they,new,guaranteed,damned,cases,
cases,m,use,s,find,pirates,directory,files,files,nntp,
need,or,guess,code,locate,locate,locate,more,see,see,
product,noted,noted,valid,valid,valid,re,write,write,
write,this,this,this,particular,original,original,original,
rpi,rpi,spaceship,reinstalling,right,think,think,i,i,here,
here,corporate,corporate,some,cover,cover,which,once,also,
info,should,what,what,what,just,just,just,install,control,
windows,windows,winorig,your,your,your,posting,posting,
posting,posting,compare,his,up,up,issue,put,put,lines,
settings,enough,real,real,half,half,half,solved,solved,
solved,do,do,do,do,no,no,no,arduous,into,said,said,said,
editor,there,there,annoying,annoying,like,like,stuck,far,
far,an,an,things,keywords,winstall,winstall,want,compressed,
compressed,hour,mentioned,quick,quick,with,data,data,ought,
silver,hermes,hermes,oldfield,genesis,genesis,cryptography,
able,from,well,tries,change,change,key,key,you,comp,comp,
name,acm,a,a,don,don,don,don,don,don,don,t,that,their,will,
will,report,report,to,to,information,bondage,bondage,bondage,
pirate,pirate,office,so,so,so,changes,changing,covering
```

< total word in 9551 - 275 >

275

PREPROCESSING - DELETE STOP WORD

- **Stop words refer to common words** that occur many times across almost all documents in a corpus (and across most corpuses).
- Examples of typical English stop words include **and, but, the, of**, and so on
- It is a **standard practice in text feature extraction** to exclude stop words from the extracted tokens
- it can still be beneficial to exclude stop words during feature extraction, as **it reduces the dimensionality of the final feature vectors as well as the size of the training data.**

```
val tokenCounts = filterNumbers.map(t => (t, 1)).reduceByKey(_ + _)
val oreringDesc = Ordering.by[(String, Int), Int](_._2)
println(tokenCounts.top( num = 20)(oreringDesc).mkString("n"))
```

→

Word	number
the	146532
to	75064
of	69034
a	64195
ax	62406
and	57957
i	53036
in	49402
is	43480
that	39264
it	33638
for	28600
you	26682
from	22670
s	22337
edu	21321
on	20493
this	20121
be	19285
t	18728

PREPROCESSING - DELETE STOP WORD

- **Stop words refer to common words** that occur many times across almost all documents in a corpus (and across most corpuses).
- Examples of typical English stop words include **and, but, the, of**, and so on
- It is a **standard practice in text feature extraction** to exclude stop words from the extracted tokens
- it can still be beneficial to exclude stop words during feature extraction, as **it reduces the dimensionality of the final feature vectors as well as the size of the training data.**

Word	number
the	146532
to	75064
of	69034
a	64195
ax	62406
and	57957
i	53036
in	49402
is	43480
that	39264
it	33638
for	28600
you	26682
from	22670
s	22337
edu	21321
on	20493
this	20121
be	19285
t	18728

```
val stopwords = Set(
  "the","a","an","of","or","in","for","by","on","but", "is",
  "not", "with", "as", "was", "if",
  "they", "are", "this", "and", "it", "have", "from", "at", "my",
  "be", "that", "to")
val tokenCountsFilteredStopwords = tokenCounts.filter { case (k, v) => !stopwords.contains(k) }
println(tokenCountsFilteredStopwords.top( num = 20)(oreringDesc).mkString("\n"))
```

Word	number
ax	62406
i	53036
you	26682
s	22337
edu	21321
t	18728
m	12756
subject	12264
com	12133
lines	11835
can	11355
organizati	11233
re	10534
what	9861
there	9689
x	9332
all	9310
will	9279
we	9227
one	9008

PREPROCESSING - DELETE STOP WORD

- we will use is **removing** any tokens that are **only one character in length**
- **single-character tokens** are **unlikely to be informative in our text** model
- and **can further reduce the feature dimension and model size**

Word	number
ax	62406
i	53036
you	26682
s	22337
edu	21321
t	18728
m	12756
subject	12264
com	12133
lines	11835
can	11355
organizati	11233
re	10534
what	9861
there	9689
x	9332
all	9310
will	9279
we	9227
one	9008

```
val tokenCountsFilteredSize =
  tokenCountsFilteredStopwords.filter { case (k, v) => k.size >= 2 }
println(tokenCountsFilteredSize.top( num = 20)(oreringDesc).mkString("\n"))
```

Word	number
ax	62406
you	26682
edu	21321
subject	12264
com	12133
lines	11835
can	11355
organizati	11233
re	10534
what	9861
there	9689
all	9310
will	9279
we	9227
one	9008
would	8905
do	8674
he	8441
about	8336
writes	7844

PREPROCESSING - DELETE WORDS BASED ON FREQUENCY

- It is also **a common practice to exclude** terms during tokenization when their **overall occurrence in the corpus is very low**

Word	number
altina	1
bluffing	1
preload	1
lennips	1
actu	1
vno	1
wbp	1
donnalyn	1
ydag	1
mirosoft	1
jjjjrw	1
harger	1
conts	1
bankruptc	1
uncompre	1
d_nibby	1
bunuel	1
odf	1
swith	1
pacified	1

< Frequency-based bottom 20 (before filter) >

```
val rareTokens = tokenCounts.filter{ case (k, v) => v < 2 }.map {
  case (k, v) => k }.collect.toSet
val tokenCountsFilteredAll = tokenCountsFilteredSize.filter {
  case (k, v) => !rareTokens.contains(k) }
```

Word	number
upo	2
loyalists	2
jejones	2
akl	2
glorifying	2
bxl	2
petr_klima	2
sively	2
isgal	2
eoeun	2
leymarie	2
podsiadlik	2
seetex	2
kielbasa	2
singen	2
za_	2
gottschalk	2
pmu	2
eer	2
artur	2

< Frequency-based bottom 20 (after filter) >

PREPROCESSING - DELETE WORDS BASED ON FREQUENCY

- It is also **a common practice to exclude** terms during tokenization when their **overall occurrence in the corpus is very low**

```
println(tokenCountsFilteredAll.count)
```

→

```
51801
```

PREPROCESSING SUMMARY

Total input paths to process : 11314

total record : 11314

tokenization → 402978

① remove nonword → 130126

② Token filtering by number → 84912

③ Delete stop word and low-frequency words → 51801

```
def tokenize(line: String): Seq[String] = {
①line.split( regex = """\W+""")
    .map(_.toLowerCase)
 ②.filter(token => regex.pattern.matcher(token).matches)
 ③.filterNot(token => stopwords.contains(token))
    .filterNot(token => rareTokens.contains(token))
    .filter(token => token.size >= 2)
    .toSeq
}
```

BUILDING A TF-IDF MODEL

- Feature Hashing
  - Feature hashing **converts a String or a word into a fixed length vector** which makes it easy to process text.

```
val dim = math.pow(2, 18).toInt
val hashingTF = new HashingTF(dim)
val tf = hashingTF.transform(tokens)
tf.cache
```

hashing dimension : 2^{18}

token → hashing

BUILDING A TF-IDF MODEL

- Feature Hashing
  - Feature hashing **converts a String or a word into a fixed length vector** which makes it easy to process text.

```
val v = tf.first.asInstanceOf[SV]
println(v.size)
println(v.values.size)
println(v.values.take(100).toSeq)
println(v.indices.take(100).toSeq)
```

```
262144
706
WrappedArray(1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 3.0, 4.0,
WrappedArray(15, 1469, 2276, 2329, 2366, 2410, 2548, 2710, 2992
```

hashing dimension : 2^{18}
number of entry : 706
Frequency of words
word to entry

BUILDING A TF-IDF MODEL

- TF-IDF

```
val idf = new IDF().fit(tf)
val tfidf = idf.transform(tf)
val v2 = tfidf.first.asInstanceOf[SV]
println(v2.values.size)
println(v2.values.take(1000).toSeq)
println(v2.indices.take(1000).toSeq)
```

calculate idf
calculate tf-idf

```
262144
706
WrappedArray(1.0, 1.0, 1.0, 1.0, 1.0, 1.0, 2.0, 1.0, 3.0, 4.0,
WrappedArray(15, 1469, 2276, 2329, 2366, 2410, 2548, 2710, 2992
```

hashing dimension : 2^{18}
number of entry : 706
Frequency of words
word to entry

```
706
WrappedArray(3.291251724385256, 6.3894455789011975, 6.694827228452379, 5.421861552639491, 4.964436705600616,
WrappedArray(15, 1469, 2276, 2329, 2366, 2410, 2548, 2710, 2992, 3834, 4200, 5381, 5519, 5595, 5795, 6183, 6
```

Frequency of words to TF-IDF

ANALYZING THE TF-IDF WEIGHTINGS

- TF-IDF weight analysis for **words**

```
val common = sc.parallelize(Seq(Seq("ax", "you", "edu","upo","loyalists","jejones")))
val tfCommon = hashingTF.transform(common)
val tfidfCommon = idf.transform(tfCommon)
val commonVector = tfidfCommon.first.asInstanceOf[SV]
println(commonVector.values.toSeq)
```

Word	number
ax	62406
you	26682
edu	21321
•	
•	
•	
upo	2
loyalists	2
jejones	2

→

Word	TF-IDF
ax	8.235272
you	8.235272
edu	8.235272
upo	5.932687
loyalists	0.42546
jejones	0.545444

→

Word	number	TF-IDF	IDF
ax	62406	8.235272	0.000132
you	26682	8.235272	0.000309
edu	21321	8.235272	0.000386
upo	2	5.932687	2.966344
loyalists	2	0.42546	0.21273
jejones	2	0.545444	0.272722

USING A TF-IDF MODEL

- TF-IDF weight analysis for **document similarity**
- Using **the cosine similarity method**
  - Simply measure similarity based on **word frequency** between documents.
  - Does not affect Scala.

Frequency	banana	apple	i	like
document 1	0	1	1	1
document 2	1	0	1	1
document 3	2	0	2	2

Cosine Similarity	document 1	document 2	document 3
document 1	**1**	0.66	0.66
document 2	0.66	**1**	1
document 3	0.66	1	**1**

USING A TF-IDF MODEL

- TF-IDF weight analysis for **document similarity**

```
val computerText = rdd.filter { case (file, text) =>
  file.contains("comp.os.ms-windows.misc") }
val computerTF = computerText.mapValues(doc =>
  hashingTF.transform(tokenize(doc)))
val computerTfIdf = idf.transform(computerTF.map(_._2))
```

access the comp.os.ms-windows.misc folder
Preprocessing and tf-idf calculation

```
import breeze.linalg._
val computer1 = computerTfIdf.sample(
  withReplacement = true, fraction = 0.1, seed = 45).first.asInstanceOf[SV]
val breeze1 = new SparseVector(computer1.indices,
  computer1.values, computer1.size)
val computer2 = computerTfIdf.sample( withReplacement = true, fraction = 0.1,
  seed = 47).first.asInstanceOf[SV]
val breeze2 = new SparseVector(computer2.indices,
  computer2.values, computer2.size)
val cosineSim = breeze1.dot(breeze2) /
  (norm(breeze1) * norm(breeze2))

println(cosineSim)
```

Same topic but **different documentation**
seed = 45, 47
Cosine similarity = 1.0

```
1.0000000000000002
```

```
import breeze.linalg._
val computer1 = computerTfIdf.sample(
  withReplacement = true, fraction = 0.1, seed = 45).first.asInstanceOf[SV]
val breeze1 = new SparseVector(computer1.indices,
  computer1.values, computer1.size)
val computer2 = computerTfIdf.sample( withReplacement = true, fraction = 0.1,
  seed = 48).first.asInstanceOf[SV]
val breeze2 = new SparseVector(computer2.indices,
  computer2.values, computer2.size)
val cosineSim = breeze1.dot(breeze2) /
  (norm(breeze1) * norm(breeze2))

println(cosineSim)
```

Same topic but **different documentation**
seed = 45, 48
Cosine similarity = 0.0297

```
0.029650986773823537
```

Measure cosine similarity based on any two documents (**Same topic but different documentation**)

USING A TF-IDF MODEL

- TF-IDF weight analysis for **document similarity**

```
val computerText = rdd.filter { case (file, text) =>
  file.contains("comp.os.ms-windows.misc") }
val computerTF = computerText.mapValues(doc =>
  hashingTF.transform(tokenize(doc)))
val computerTfIdf = idf.transform(computerTF.map(_._2))
```

access the comp.os.ms-windows.misc folder
Preprocessing and tf-idf calculation

```
val graphicsText = rdd.filter { case (file, text) =>
  file.contains("comp.graphics") }
val graphicsTF = graphicsText.mapValues(doc =>
  hashingTF.transform(tokenize(doc)))
val graphicsTfIdf = idf.transform(graphicsTF.map(_._2))
val graphics = graphicsTfIdf.sample( withReplacement = true,  fraction = 0.1,
  seed = 42).first.asInstanceOf[SV]
val breezeGraphics = new SparseVector(graphics.indices,
  graphics.values, graphics.size)
val cosineSim2 = breeze1.dot(breezeGraphics) / (norm(breeze1) *
  norm(breezeGraphics))

println(cosineSim2)
```

different topic : ms-windows.misc and **graphics**
Cosine similarity = 0.00597

```
0.005971365556642889
```

```
val christianText = rdd.filter { case (file, text) =>
  file.contains("baseball") }
val christianTF = christianText.mapValues(doc =>
  hashingTF.transform(tokenize(doc)))
val christianTfIdf = idf.transform(christianTF.map(_._2))
val christian = christianTfIdf.sample( withReplacement = true,  fraction = 0.1,
  seed = 42).first.asInstanceOf[SV]
val breezechristian = new SparseVector(christian.indices,
  christian.values, christian.size)
val cosineSim3 = breeze1.dot(breezechristian) / (norm(breeze1) *
  norm(breezechristian))

println(cosineSim3)
```

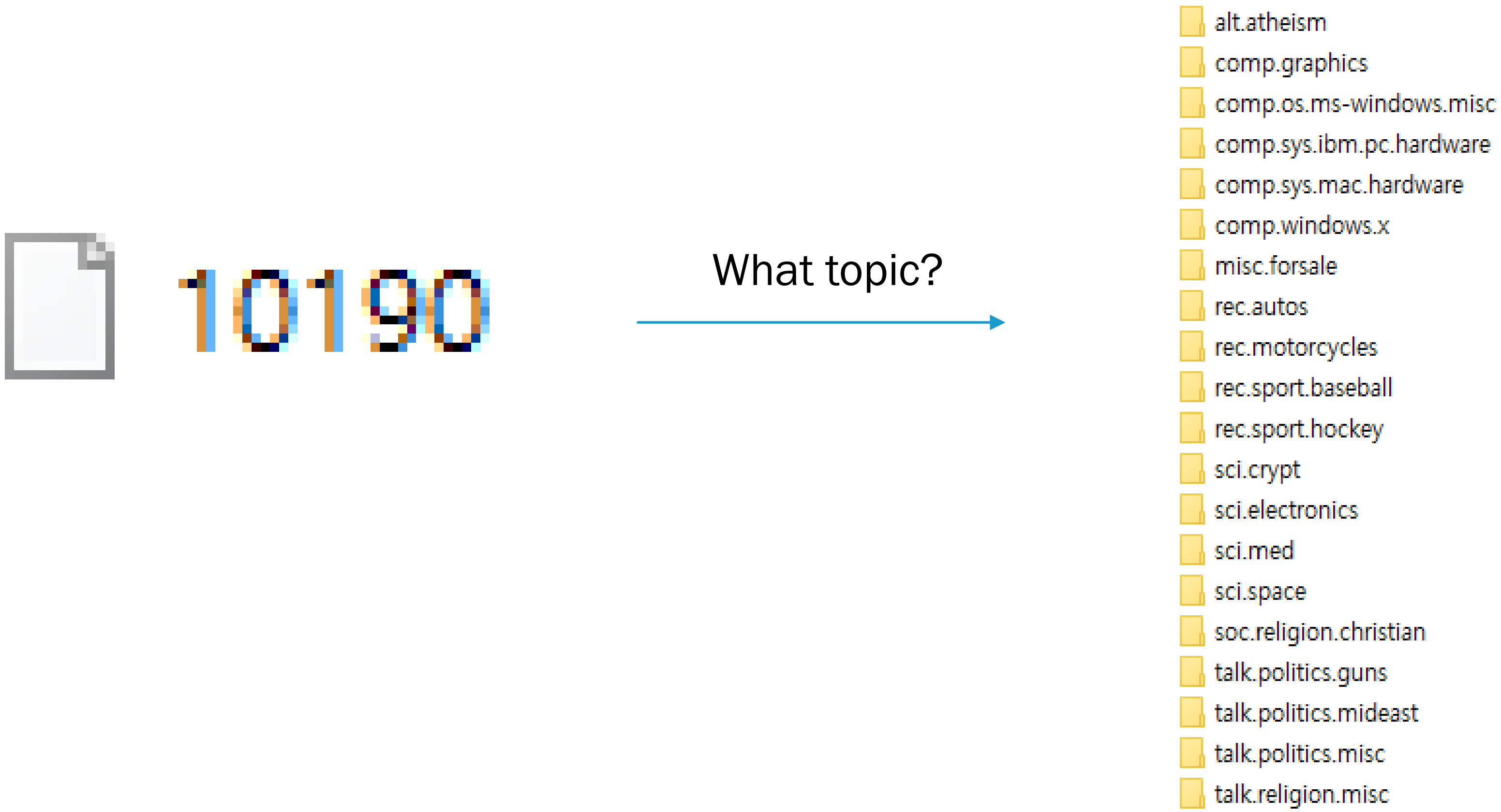
Measure cosine similarity based on any two documents (**different topic** and **documentation**)

different topic : ms-windows.misc and **christian**
Cosine similarity = 0.00637

```
0.006376513513613797
```

USING A TF-IDF MODEL

- Training a text classifier
  - The classification in this chapter is to **find out what topics the document belongs to.**



USING A TF-IDF MODEL

- Training a text classifier
  - The classification in this chapter is to **find out what topics the document belongs to.**

```scala
val newsgroupsMap =
  newsgroups.distinct.collect().zipWithIndex.toMap
val zipped = newsgroups.zip(tfidf)
val train = zipped.map { case (topic, vector) =>
  LabeledPoint(newsgroupsMap(topic), vector) }
train.cache

val model = NaiveBayes.train(train, lambda = 0.1)

val testPath = "C:\\Users\\KJH\\Desktop\\spark\\20news-bydate-test\\*\\**"
val testRDD = sc.wholeTextFiles(testPath)
val testLabels = testRDD.map { case (file, text) =>
  val topic = file.split( regex = "/").takeRight(2).head
  newsgroupsMap(topic)}

val testTf = testRDD.map { case (file, text) =>
  hashingTF.transform(tokenize(text)) }
val testTfIdf = idf.transform(testTf)
val zippedTest = testLabels.zip(testTfIdf)
val test = zippedTest.map { case (topic, vector) =>
  LabeledPoint(topic, vector) }

val predictionAndLabel = test.map(p =>
  (model.predict(p.features), p.label))
val accuracy = 1.0 * predictionAndLabel.filter(x => x._1 == x._2).count() / test.count()
val metrics = new MulticlassMetrics(predictionAndLabel)

println(accuracy)
println(metrics.weightedFMeasure)
```

Accuracy
F-measure

* **Filtered data**, Naive Bayes Classification

Accuracy : 0.7928 0.7928836962294211
F-measure : 0.7822 0.782264437643170

$$\text{F-measure} = 2 * \frac{precision * recall}{precision + recall}$$

* how reliable it is for accuracy

USING A TF-IDF MODEL

- Training a text classifier
  - The classification in this chapter is to **find out what topics the document belongs to.**

```
val rawTokens = rdd.map { case (file, text) => text.split( regex = " ") }
val rawTF = rawTokens.map(doc => hashingTF.transform(doc))
val rawTrain = newsgroups.zip(rawTF).map { case (topic, vector)
=> LabeledPoint(newsgroupsMap(topic), vector) }
val rawModel = NaiveBayes.train(rawTrain, lambda = 0.1)
val rawTestTF = testRDD.map { case (file, text) =>
  hashingTF.transform(text.split( regex = " ")) }
val rawZippedTest = testLabels.zip(rawTestTF)
val rawTest = rawZippedTest.map { case (topic, vector) =>
  LabeledPoint(topic, vector) }
val rawPredictionAndLabel = rawTest.map(p =>
  (rawModel.predict(p.features), p.label))
val rawAccuracy = 1.0 * rawPredictionAndLabel.filter(x => x._1
  == x._2).count() / rawTest.count()
println(rawAccuracy)
val rawMetrics = new MulticlassMetrics(rawPredictionAndLabel)
println(rawMetrics.weightedFMeasure)
```

Accuracy

F-measure

* **Unfiltered data,** Naive Bayes Classification

Accuracy : 0.7661 0.7661975570897503

F-measure : 0.7653 0.7653320418573546

$$\text{F-measure} = 2 * \frac{precision*recall}{precision+recall}$$

* how reliable it is for accuracy

COMPARING RAW FEATURES WITH PROCESSED TF-IDF FEATURES

- Filtered data, Naive Bayes Classification
  - Accuracy : 0.7928
  - F-measure : 0.7822
- Unfiltered data, Naive Bayes Classification
  - Accuracy : 0.7661
  - F-measure : 0.7653
- the raw model does quite well, although both accuracy and F-measure are a few percentage points lower than those of the TF-IDF model.
- This is also partly a reflection of the fact that the naive Bayes model is well suited to data in the form of raw frequency counts

WORD2VEC WITH SPARK ML

- Word2Vec weights **the words most similar to the Specific words**.

```
import org.apache.spark.mllib.feature.Word2Vec
val word2vec = new Word2Vec()
val word2vecModel = word2vec.fit(tokens)

word2vecModel.findSynonyms( word = "space",  num = 20).foreach(println)

word2vecModel.findSynonyms( word = "graphics",  num = 20).foreach(println)

word2vecModel.findSynonyms( word = "christian",  num = 20).foreach(println)
sc.stop()
```

Words similar to the 'space'	
word	weight
program	0.588155389
launch	0.577293336
mission	0.566586077
manned	0.543366373
redesign	0.523284137
shuttle	0.522432506
japanese	0.52094686
national	0.520386159
funding	0.516603291
planned	0.511429608
colony	0.506110787
orbital	0.503944159
aiaa	0.502817929
probes	0.501937211
spacecraft	0.501250148
vehicle	0.498602539
dc	0.489739865
race	0.485708505
added	0.485268354
moon	0.484716713

WORD2VEC WITH SPARK ML

- Word2Vec weights **the words most similar to the Specific words**.

```
import org.apache.spark.mllib.feature.Word2Vec
val word2vec = new Word2Vec()
val word2vecModel = word2vec.fit(tokens)

word2vecModel.findSynonyms( word = "space",  num = 20).foreach(println)

word2vecModel.findSynonyms( word = "graphics",  num = 20).foreach(println)

word2vecModel.findSynonyms( word = "christian",  num = 20).foreach(println)
sc.stop()
```

Words similar to the 'graphics'	
word	weight
tektronix	0.605110109
silicon	0.596348822
shoreline	0.595423639
incorporated	0.593367755
cascade	0.581241608
interactive	0.570908785
kubota	0.557497501
consoles	0.555823922
graphing	0.55426544
wgt	0.551806271
consulting	0.546853483
concurrent	0.546537757
oxnard	0.545503259
ati	0.534702539
microsystems	0.532416582
wilsonville	0.530458272
minivas	0.526643515
productivity	0.522192538
typewriter	0.521038473
polk	0.516390085

WORD2VEC WITH SPARK ML

- Word2Vec weights **the words most similar to the Specific words**.

```
import org.apache.spark.mllib.feature.Word2Vec
val word2vec = new Word2Vec()
val word2vecModel = word2vec.fit(tokens)

word2vecModel.findSynonyms( word = "space",  num = 20).foreach(println)

word2vecModel.findSynonyms( word = "graphics",  num = 20).foreach(println)

word2vecModel.findSynonyms( word = "christian",  num = 20).foreach(println)
sc.stop()
```

Words similar to the 'christian'	
word	weight
religion	0.745340228
christians	0.725812495
doctrine	0.723084092
jews	0.71306175
christianity	0.708423793
commited	0.690073848
worship	0.689117491
pauline	0.686294615
zionist	0.685362279
church	0.680460453
orthodox	0.678712189
oneness	0.677396953
conception	0.676100552
hebrews	0.674821794
religious	0.674247384
clh	0.670958221
greek	0.670535922
non	0.665308356
arabs	0.664627492
ephesians	0.661558509