NLP Course

Daniel Dan

11/12/2024

Executive Academy, WU, Vienna

Natural Language Processing (NLP)

Outline

- Text Preparation
- Clustering
- Classification
- Sentiment Analysis
- Named Entity Recognition
- Topic Modeling
- Word Embeddings
- Transformers
- Chatbots
- Chatbots implmentation

Text and computers

Computers work with numbers, they are extremely efficient at this task, and this is the reason they were initially built for: mathematical computations.

Slowly, some of the real-world mechanics has been adapted and shifted to digital use. Although once there were plenty of typewriters, now we have switched to computers that allow us to lay text on a white canvas of pixels mimicked as a sheet of paper.

Most people write text, but just **a tiny fraction analyze** it and the former one represent our focus here. In Text Analytics, text has to be made ready for computer interpretation and we have the duty to ensure this process. We act at a higher level and do not see the transformation, but we have to follow specific strategies and procedures to convert text documents into vectors of numbers. This mapping between a sequence of symbols and digits is called **feature representation, feature extraction or feature encoding**.

There are many basic features that can be extracted from text, such as: the **number of words**, the **number of characters**, the **number of special characters**, the **number of numerics**, the **average word length**, and so on.

These features can be extracted through a simple method called bag-of-words. The name “bag” hints that words are thrown in a container without a specific order, and this is true.

The bag of words

Whenever we apply this model, we deal exclusively with the presence of the words in a text and their frequency but not with the structure of the text. This simple model allows us to roughly grasp the meaning of the text only from the analysis of some of its (unordered) words.

We will see how by destroying the text's structure and leave only some of the words, we can gain some insight about it without reading hundreds of pages. First, we start with a reasoned text **preprocessing**, transform the text into a **matrix** so that our computer can deal with it, and then we analyze the **outcome**.

What we are trying to do is a **dimensionality reduction**. From huge amounts of documents, we will try to understand their latent meaning and as such we have to cut and remove parts of the text leaving only the essential (and meaningful) elements.

Since text must be structured and made ready for computer elaborations it is essential to normalize or preprocess it.

After **editing the text** with the help of the **regular expressions**, the main preliminary operations are the following: **corpus preparation, white space contraction, lowercase transformation, stopwords removal, implicit words removal, long words removal, punctuation removal, numbers removal** or **numbers to text translation and vice versa** and the **reduction of the inflectional forms**. We will analyze each of them in the sequence they have to be applied.

Corpus preparation

The package we will use is called `tm` which is the abbreviation for Text Mining. The **functions** in this section are referred to it unless differently specified. The structure the `tm` package works with is the Corpus, plural Corpora. It is an abstract notion consisting of a collection of texts that have to be analyzed. `tm` implements by default the Volatile Corpus or VCorpus type which is a structure held in memory in an R object that vanishes if destroyed. The source can be a directory, a vector which considers every document as a single item or a data frame. Here we will use the VectorSource. To have a sneak preview of how the first document in the corpus looks like we use the function `inspect()`.

```r
# avoid strings factorization
options(stringsAsFactors = FALSE)

library(tm)

# read the text from your local data folder (or from the internet)
prodreviews <- read.csv("Product Reviews.csv")

# read the content field from the dataset and make sure it is character casted
content <- as.character(prodreviews$Content)
```

Corpus preparation

```r
# select the content as a Vector Source,transform it in a Volatile Corpus object
docs <- VCorpus(VectorSource(content))

# have a preview of a document inside the VCorpus
inspect(docs[[1]])
```

```
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 1384
##
## I love my fire and highly recommend it to anyone who wants to watch videos (netflix,
hulu, amazon), read ebooks (purchased or from the local library), surf the net and play
games. I work in the tech field and I LOVE apple entertainment products (I own many apple
products and at work I work with several). I am very thrilled with my fire (I LOVE IT
TOO!) because it works great as an entertainment product (and more affordable than my
apple products). I also think the fire is a great product because of Amazons clout and
support. Any tech product I buy needs to be useable for years and updateable during those
years. The amazon market (still growing) is and will be getting better all the time. Why?
Sheer number of devices sold by Amazon equals a large potential population of people who
will buy software equals software programmers wanting to be a part of that market. Also
in IT the biggest frustration is having an older device and not being able to update it
or not being able to use new software or technology because the manufacturer does not
update their devices (software) or provide backwards support. I am not saying that Amazon
will continue to provide support (that remains to be seen), but the size, longevity,
profitability and attitude of pro consumer of a company are determining factors and
compared to other companies I would bet that Amazon will not disappoint.
```

Lowercase transformation

For humans, the distinction between the word hotel and Hotel is minimal and anyway, they both talk about a structure that provides lodging. For computers this is different.

There is no meaning to them, just a sequence of letters. If one of the letters is uppercase, then the two words are completely distinct. That's why we need to lowercase all the characters and thus to make all the sequences uniform so that they can be counted. The function that can accomplish this is `tolower()` and is part of the R base program. One thing has to be taken care of, the function `tm_map()` returns a character vector not a PlainTextDocument. This can be overcome with the help of the `content_transformer()` wrapper function applied to `tolower()`.

```r
# transform all the characters to lowercase
docs <- tm_map(docs, content_transformer(tolower))

# have a preview of the transformed document
inspect(docs[[1]])
```

```
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 1384
##
## i love my fire and highly recommend it to anyone who wants to watch videos (netflix,
hulu, amazon), read ebooks (purchased or from the local library), surf the net and play
games. i work in the tech field and i love apple entertainment products (i own many apple
products and at work i work with several). i am very thrilled with my fire (i love it
too!) because it works great as an entertainment product (and more affordable than my
apple products). i also think the fire is a great product because of amazons clout and
support. any tech product i buy needs to be useable for years and updateable during those
years. the amazon market (still growing) is and will be getting better all the time. why?
```

Stop words removal

If you tried to find the most frequent word in a text, you would realize that the most common one is the article the. It does not add value to our research and will not distinguish the analyzed text from any other, if we compared them, because they would all list it as being prevalent. Such words that appear very frequently and do not give insight into the unicity of a text are called stop words. As you can see there are some words written in contracted form such as isn't. And that's why this step must be done before the punctuation removal. The function that defines/contains the stop words is `stopwords()` and the function that removes them is `removeWords()`.

```r
# remove the stopwords and specify the language
docs <- tm_map(docs, content_transformer(removeWords), stopwords("english"))

# have a preview of the transformed document
inspect(docs[[1]])
```

```
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 1071
##
## love fire highly recommend anyone wants watch videos (netflix, hulu, amazon), read
ebooks (purchased local library), surf net play games.  work tech field love apple
entertainment products ( many apple products work work several).  thrilled fire ( love !)
works great entertainment product ( affordable apple products).  also think fire great
product amazons clout support.  tech product buy needs useable years updateable years.
amazon market (still growing) will getting better time. ? sheer number devices sold
amazon equals large potential population people will buy software equals software
programmers wanting part market. also biggest frustration older device able update able
use new software technology manufacturer update devices (software) provide backwards
support.  saying amazon will continue provide support ( remains seen), size, longevity,
```

Own dictionary words removal

Alongside the stop words, we might need to remove other additional words. This operation comes with some trial and error attempts as well as with an educated guess. In a data set where we talk extensively about a city name, maybe it becomes so often in the corpus that we would like to remove it because it will not yield any insight. The function to do that is again `removeWords()`. Let us say that we want to remove the words **amazon** and **love**.

```r
# remove some specific words
docs <- tm_map(docs, content_transformer(removeWords), c("amazon", "love"))

# have a preview of the transformed document
inspect(docs[[1]])
```

```
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 1029
##
## fire highly recommend anyone wants watch videos (netflix, hulu, ), read ebooks
(purchased local library), surf net play games.  work tech field apple entertainment
products ( many apple products work work several).  thrilled fire ( !)  works great
entertainment product ( affordable apple products).  also think fire great product
amazons clout support.  tech product buy needs useable years updateable years.  market
(still growing) will getting better time. ? sheer number devices sold equals large
potential population people will buy software equals software programmers wanting part
market. also biggest frustration older device able update able use new software
technology manufacturer update devices (software) provide backwards support.  saying will
continue provide support ( remains seen), size, longevity, profitability attitude pro
consumer company determining factors compared companies bet will disappoint.
```

Punctuation removal

The punctuation has to be removed because it is useless for the analysis purpose in a bag of words approach. In this case, semantics is not important and thus punctuation does not add value: function `removePunctuation()`.

```
# remove the punctuation
docs <- tm_map(docs, removePunctuation)

# have a preview of the transformed document
inspect(docs[[1]])
```

```
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 995
##
## fire highly recommend anyone wants watch videos netflix hulu read ebooks purchased
local library surf net play games work tech field apple entertainment products many apple
products work work several thrilled fire works great entertainment product affordable
apple products also think fire great product amazons clout support tech product buy needs
useable years updateable years market still growing will getting better time sheer number
devices sold equals large potential population people will buy software equals software
programmers wanting part market also biggest frustration older device able update able
use new software technology manufacturer update devices software provide backwards
support saying will continue provide support remains seen size longevity profitability
attitude pro consumer company determining factors compared companies bet will disappoint
```

Numbers removal or transformation

Numbers may or may not be relevant to our analyses but most of the time they are not. Nevertheless, through experience you will realize if in your need to transform digits into written numbers (e.g. from 3 to three) or to remove them completely. A function can be easily created to map between numbers and text. The function `removeNumbers()` will remove the numbers from the text.

```
# remove the numbers
docs <- tm_map(docs, removeNumbers)

# have a preview of the transformed document
inspect(docs[[1]])
```

```
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 995
##
## fire highly recommend anyone wants watch videos netflix hulu read ebooks purchased
local library surf net play games work tech field apple entertainment products many apple
products work work several thrilled fire works great entertainment product affordable
apple products also think fire great product amazons clout support tech product buy needs
useable years updateable years market still growing will getting better time sheer number
devices sold equals large potential population people will buy software equals software
programmers wanting part market also biggest frustration older device able update able
use new software technology manufacturer update devices software provide backwards
support saying will continue provide support remains seen size longevity profitability
attitude pro consumer company determining factors compared companies bet will disappoint
```

Reduction of inflectional forms

To reduce the inflectional forms, stemming and lemmatization are regularly employed. A stemming algorithm refers to a heuristic computational process that tries to uniform the words to a common root or a common stem depending if we removed the prefix or the end of the words.

For example, a stemmer would consider the word **market** to capture the words **marketing**, **marketed** or **marketization**. The hope is to achieve this goal correctly most of the times. The `tm` package uses the `SnowballC` package for stemming which in turn implements Martin Porter's version of the stemming algorithm. This version became standard for the English language stemming. It does not reduce to a known stem word but to something that is created, for example, the words **replacement, replace, replacing** are reduced to replac.

Lemmatization is a process of the removal of the inflectional ending through the use of the vocabulary and the morphological analysis to form another word, known as lemma. For example, the words **do, did, done** are reduced to the lemma do.

Reduction of inflectional forms: stemming

```r
# load the SnoballC library for stemming
library(SnowballC)

# stemming
docs <- tm_map(docs, content_transformer(stemDocument))

# have a preview of the transformed document
inspect(docs[[1]])
```

```
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 758
##
## fire high recommend anyon want watch video netflix hulu read ebook purchas local
librari surf net play game work tech field appl entertain product mani appl product work
work sever thrill fire work great entertain product afford appl product also think fire
great product amazon clout support tech product buy need useabl year updat year market
still grow will get better time sheer number devic sold equal larg potenti popul peopl
will buy softwar equal softwar programm want part market also biggest frustrat older
devic abl updat abl use new softwar technolog manufactur updat devic softwar provid
backward support say will continu provid support remain seen size longev profit attitud
pro consum compani determin factor compar compani bet will disappoint
```

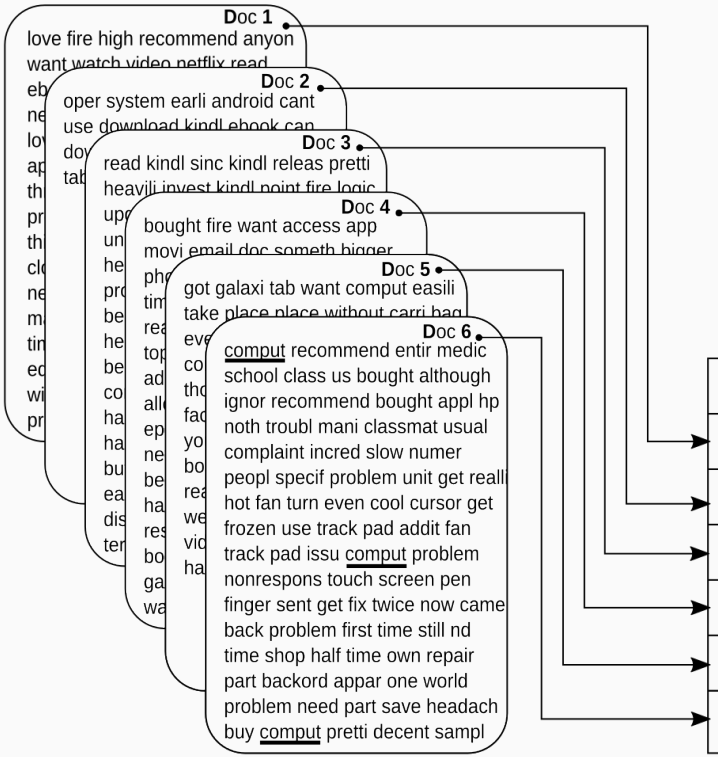
White space removal

In case there are multiple blank spaces also called white spaces, the objective is to reduce them into a single blank one. It also removes space between paragraphs, etc. It is useful for better readability.

```r
# extra space removal
docs <- tm_map(docs, stripWhitespace)

# have a preview of the transformed document
inspect(docs[[1]])
```

```
## <<PlainTextDocument>>
## Metadata: 7
## Content: chars: 758
##
## fire high recommend anyon want watch video netflix hulu read ebook purchas local
librari surf net play game work tech field appl entertain product mani appl product work
work sever thrill fire work great entertain product afford appl product also think fire
great product amazon clout support tech product buy need useabl year updat year market
still grow will get better time sheer number devic sold equal larg potenti popul peopl
will buy softwar equal softwar programm want part market also biggest frustrat older
devic abl updat abl use new softwar technolog manufactur updat devic softwar provid
backward support say will continu provid support remain seen size longev profit attitud
pro consum compani determin factor compar compani bet will disappoint
```

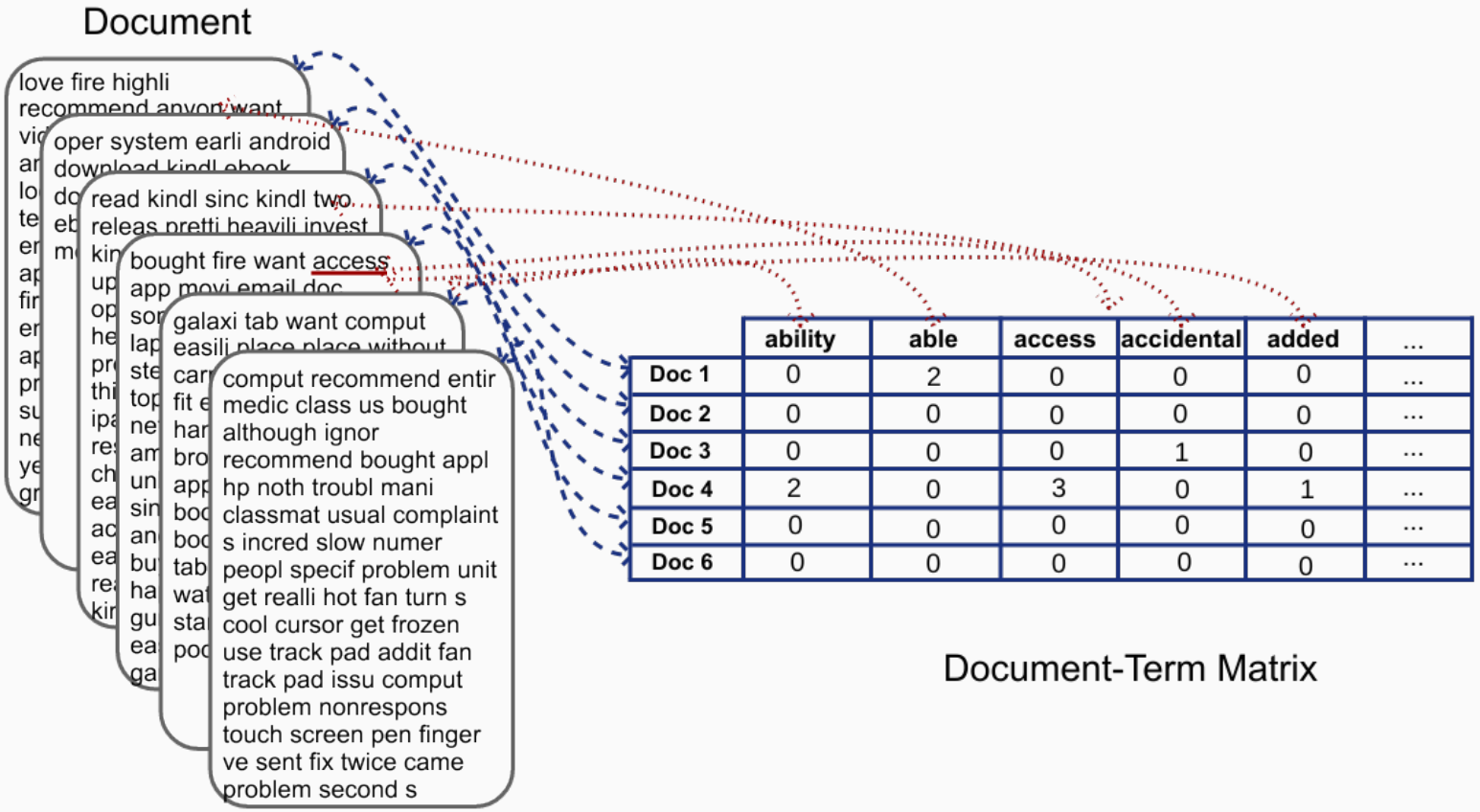
Document-term matrix: one hot encoding



Document-Term Matrix

	ability	able	access	accidental	added	…	comput	…
Doc 1	0	1	0	0	0	…	0	…
Doc 2	0	0	0	0	0	…	0	…
Doc 3	0	0	0	1	0	…	0	…
Doc 4	1	0	1	0	1	…	0	…
Doc 5	0	0	0	0	0	…	0	…
Doc 6	0	0	0	0	0	…	1	…

Document-term matrix



	ability	able	access	accidental	added	...
Doc 1	0	2	0	0	0	...
Doc 2	0	0	0	0	0	...
Doc 3	0	0	0	1	0	...
Doc 4	2	0	3	0	1	...
Doc 5	0	0	0	0	0	...
Doc 6	0	0	0	0	0	...

TF-IDF: term frequency - inverse document frequency

Since only a few number of terms appear inside each document, the result is a very sparse matrix. To reduce furthermore its dimensionality, several schemes can be employed to determine the value each entry should take. Translated into plain English, we will apply another dimensionality reduction technique. This operation is similar to the removal of specific words, it will remove the words that occur too frequently and as such give little information. A scheme example can be the term frequency-inverse document frequency (**tf-idf**) which is widely employed in information retrieval.

Each matrix value represents the frequency of the specific term "j" which appears in the document "i", we call this part **tf**. So, **tf** measures the importance of a term in a specific document: **the higher the frequency, the more important the term**. The inverse document frequency measures, on the corpus, **the number of documents which contain the term "j"** and is a measure of how significant a certain term is in the whole set of documents. We call it **idf**. The **higher the idf, the rarer the term** while the **lower the idf, the more common the term** is.

The tf-idf score is the product of the two metrics: tf $\times$ idf. The **tf-idf** score increases when term "j" appears frequently in document "i" while it decreases if the term appears in other documents. The IDF part is computed on a logarithmic scale, i.e idf (of a term) $= \log_e \frac{\text{number of documents}}{\text{number of documents containing the term}}$. Basically, **what TF-IDF accomplishes** is to **spot the words that are important to a document** in a corpus. While **TF accomplished the first part of a BoW approach, the TF-IDF metric will make us understand what words characterize what documents**.

TF-IDF example

Two documents:

D1: The best marketing strategies are the simplest.

D2: Here are the best marketers.

	are	best	here	marketers	marketing	simplest	strategies	the
TF(Doc1)	$\frac{1}{7}$	$\frac{1}{7}$	0	0	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{1}{7}$	$\frac{2}{7}$
TF(Doc2)	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	$\frac{1}{5}$	0	0	0	$\frac{1}{5}$
IDF(Term)	$log_e \frac{2}{2} = 0$	$log_e \frac{2}{2} = 0$	$log_e \frac{2}{1} \approx 0.69$	$log_e \frac{2}{1} \approx 0.69$	$log_e \frac{2}{1} \approx 0.69$	$log_e \frac{2}{1} \approx 0.69$	$log_e \frac{2}{1} \approx 0.69$	$log_e \frac{2}{2} = 0$
TF(D1) x IDF(Term)	$\frac{1}{7} \times 0 = 0$	$\frac{1}{7} \times 0 = 0$	$0 \times log_e \frac{2}{1} = 0$	$0 \times log_e \frac{2}{1} = 0$	$\frac{1}{7} \times log_e \frac{2}{1} = 0.043$	$\frac{1}{7} \times log_e \frac{2}{1} = 0.043$	$\frac{1}{7} \times log_e \frac{2}{1} = 0.043$	$\frac{2}{7} \times 0 = 0$
TF(D2) x IDF(Term)	$\frac{1}{5} \times 0 = 0$	$\frac{1}{5} \times 0 = 0$	$\frac{1}{5} \times log_e \frac{2}{1} = 0.06$	$\frac{1}{5} \times log_e \frac{2}{1} = 0.06$	$0 \times log_e \frac{2}{1} = 0$	$0 \times log_e \frac{2}{1} = 0$	$0 \times log_e \frac{2}{1} = 0$	$\frac{1}{5} \times 0 = 0$

Sparse terms removal

We are often not interested in infrequent terms in our documents. Such "sparse" terms should be removed from the corpus to be analyzed but sometimes there are unique words that describe a situation or a product enriching the text. This might give us some insight and we suggest not to dismiss the hypothesis of getting knowledge from them. These words are technically called **hapax legomena** or in a shortly **hapax**. The longer the text the higher the probability to see an increase in **hapax**.

As a last operation, the sparse terms are removed from the dtm. These terms are words that appear very infrequently in a document and are represented by a dtm that contains a huge number of entries set to zero. The sparse option from the `SparseTerms()` function was used. The sparsity of a term refers to a limit of a relative document frequency above which the term will be discarded. **Sparsity decreases as it approaches 1.0 and can take values in the open interval (0, 1.0)**.

The sparse parameter measures the percentage of zeros contained in each term. It is like a threshold.

For example, sparse $= 0.1$ will keep the terms that have 10% (or fewer) zeros among all documents. If the cut-off threshold is at 10% most of the words will be comprised. At the other side of the interval if we take, sparse $= 0.95$ include the terms that have 95% zeros. Most corpora use values between 0.95 and 0.999999. If an element has 95% of zeros, it is very rare. Is it so?

Word frequency

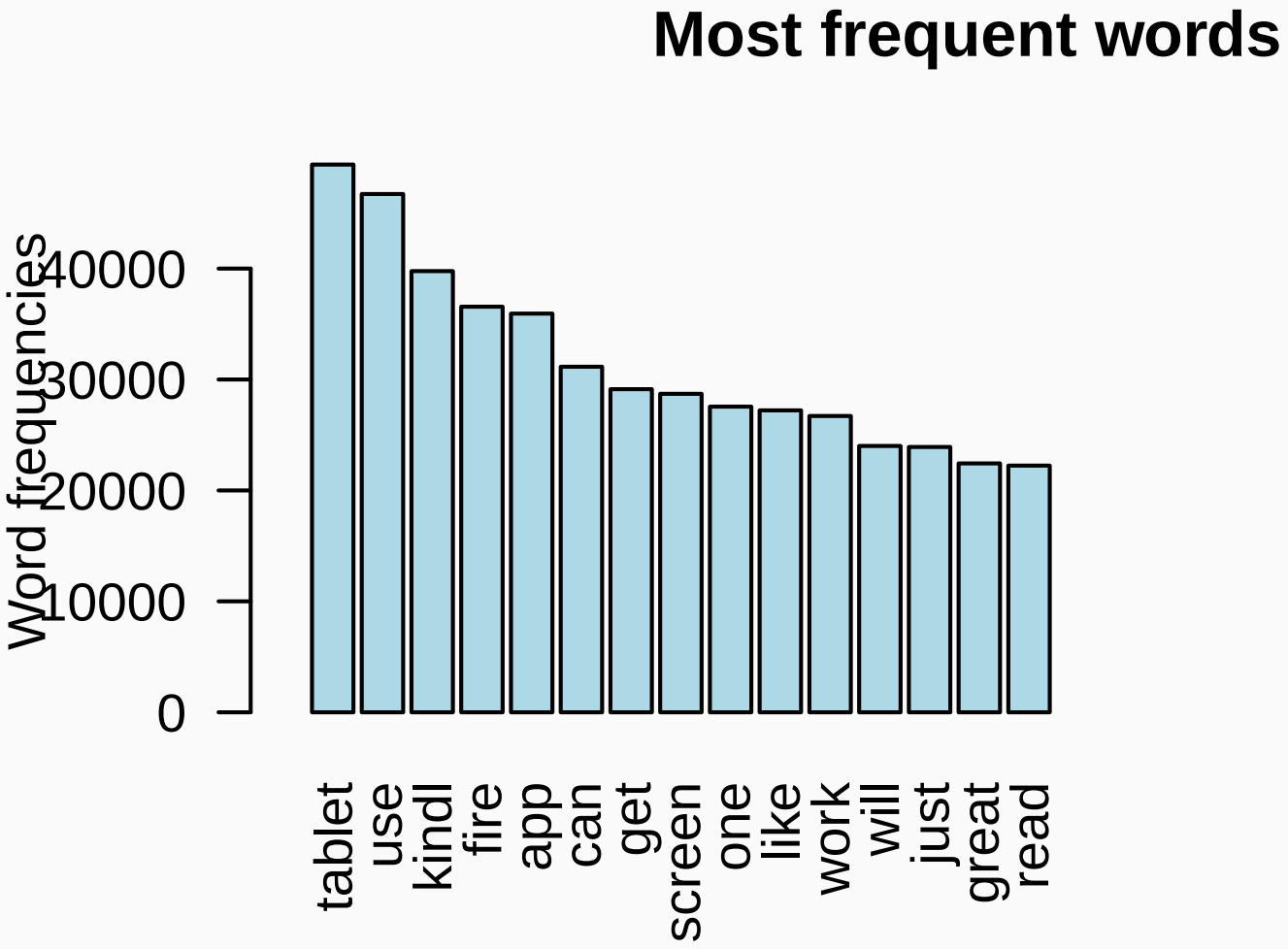
After the text preprocessing, a simple analysis can be developed in order to search for words that best describe the content. One objective is to find the frequency for each term. We calculate it by adding up the term occurrences column wise. Then we sort them in decreasing order. As a preview of the data, we display the 10 most frequent terms together with their corresponding frequency:

This simple form of text analysis can be visualized under the form of a list, like in the image above or in a more graphic way with plots or word clouds. Frequently, words can be associated because they occur together, and they can be represented on a graph. This type of analysis gives a hint on the structure that characterize the text.

```
library(slam)
tdm <- TermDocumentMatrix(docs)
m <- removeSparseTerms(tdm, 0.7)
v <- sort(row_sums(m), decreasing=TRUE)
d <- data.frame(word = names(v), freq=v)
head(d, 10)
```

```
##          word  freq
## tablet tablet 49375
## use       use 46724
## kindl   kindl 39774
## fire     fire 36568
## app       app 35944
## can       can 31150
## get       get 29128
## screen screen 28710
## one       one 27557
## like     like 27221
```

```r
barplot(d[1:30,]$freq, las = 2, names.arg = d[1:30,]$word, col ="lightblue",
        main ="Most frequent words", ylab = "Word frequencies", xlim=c(0,32))
```



Word frequency

```r
library(wordcloud)
set.seed(1234)
wordcloud(words = d$word, freq = d$freq, min.freq = 1, max.words=200,
          random.order=FALSE, rot.per=0.25, colors=brewer.pal(8, "Dark2"))
```

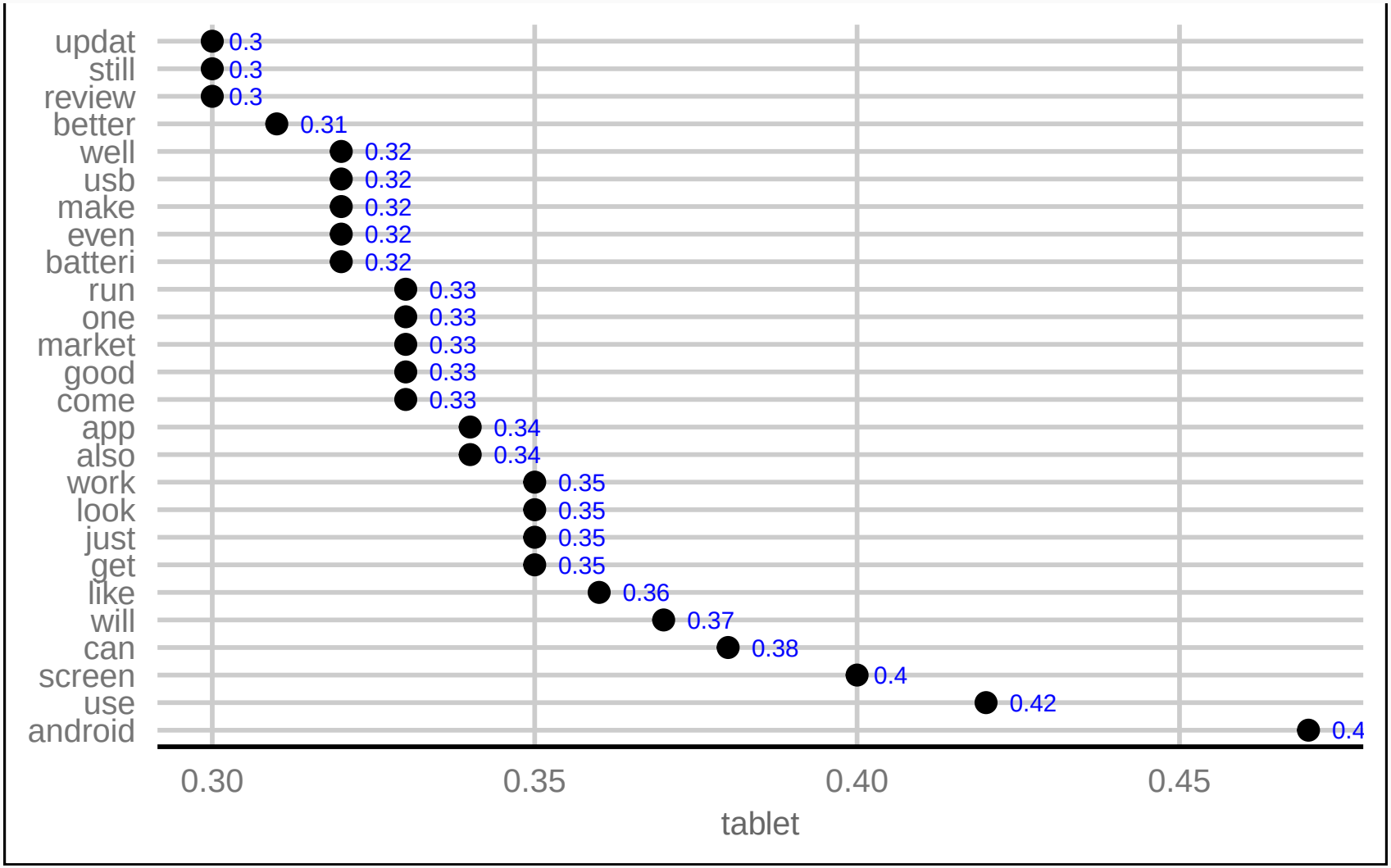
Word correlation

Words in a text are not unrelated. They are linked to other words to create meaning. To get a better grasp of our data, we can check for correlations between some of the most frequent terms and the other words. By correlation, in this context, we mean a measure of the words that co-occur in a document set.

A correlation of the most frequently used words can be assessed through the `findAssocs()` function of the tm package. We just have to specify the document-term matrix and the correlation limit which is a number between 0 and 1. The limit represents a lower bound for the correlation strength between a chosen term and the co-occurred ones. As an example, we try to find the term co-occurrence at a correlation threshold of at least 0.10.

```
associations <- findAssocs(tdm,terms="tablet", corlimit=0.78)
associations <- as.data.frame(associations)
associations$terms <- row.names(associations)
associations$terms <- factor(associations$terms, levels=associations$terms)
library(ggplot2)
library(ggthemes)
ggplot(associations, aes(y=terms))+
geom_point(aes(x=tablet), data=associations, size=5)+
theme_gdocs()+ geom_text(aes(x=tablet, label=tablet),
color="blue", hjust=-.5, size=5)+
theme(text=element_text(size=10), axis.title.y=element_blank())
```

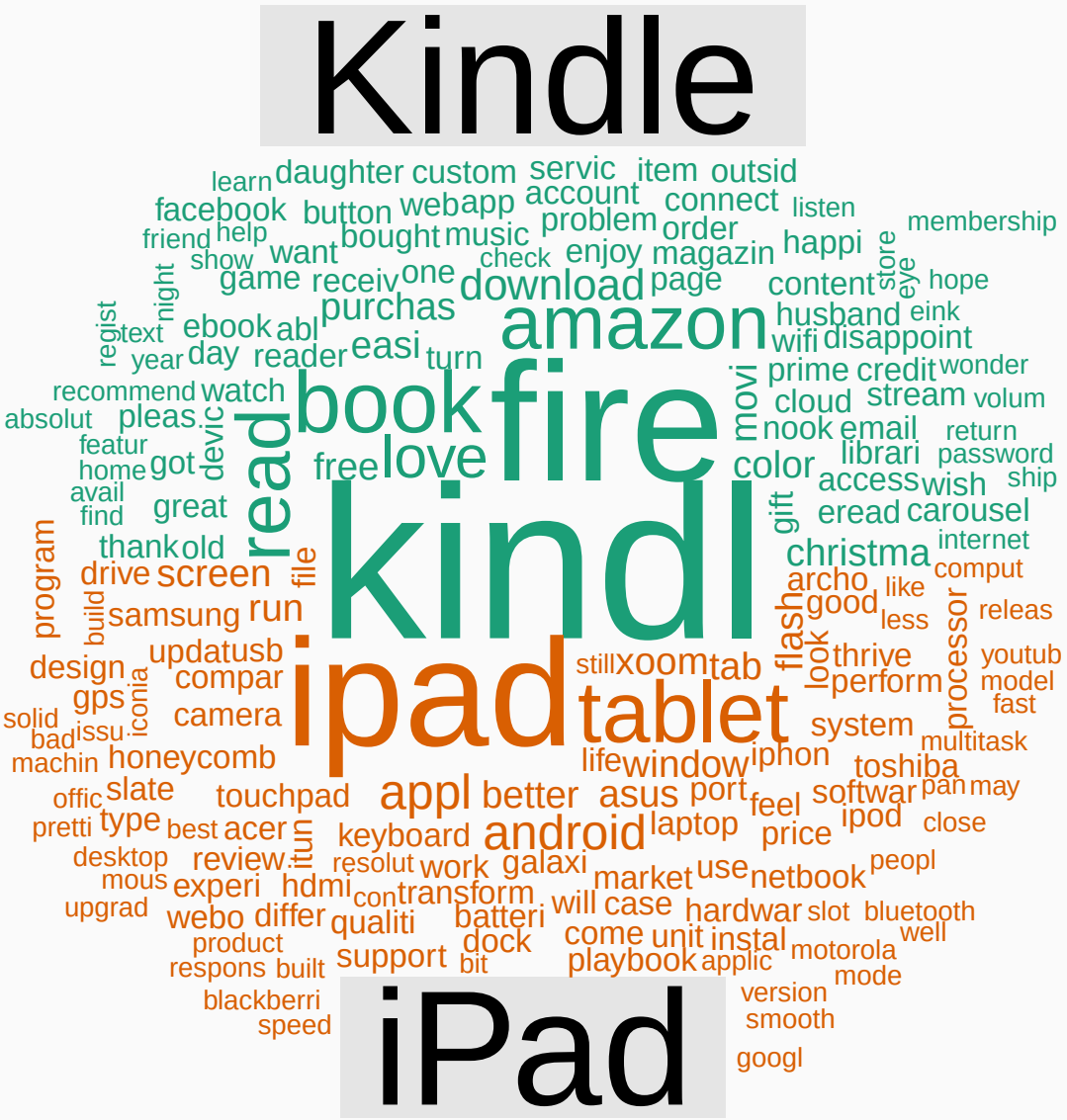
Word correlation

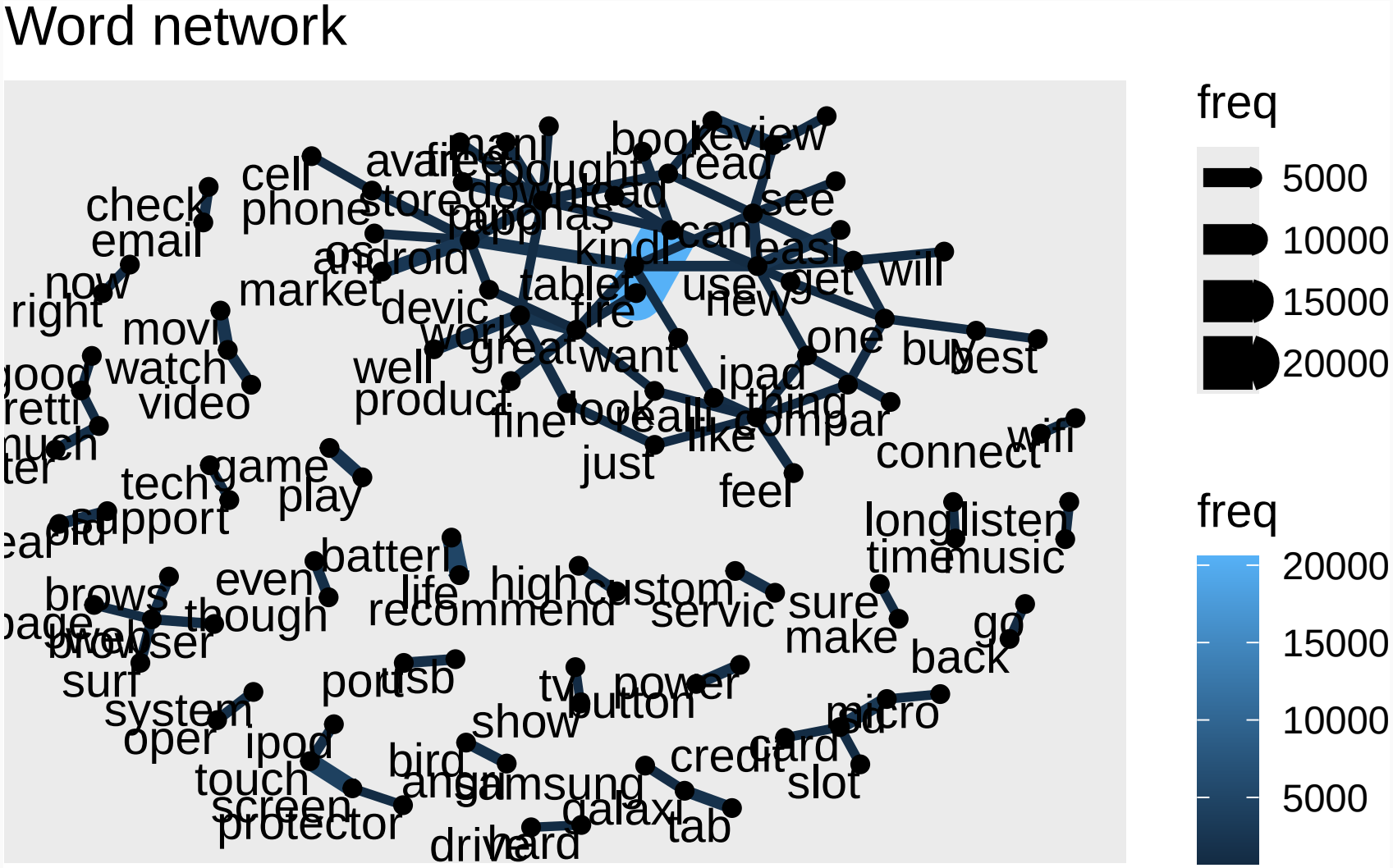


Bigrams wordcloud

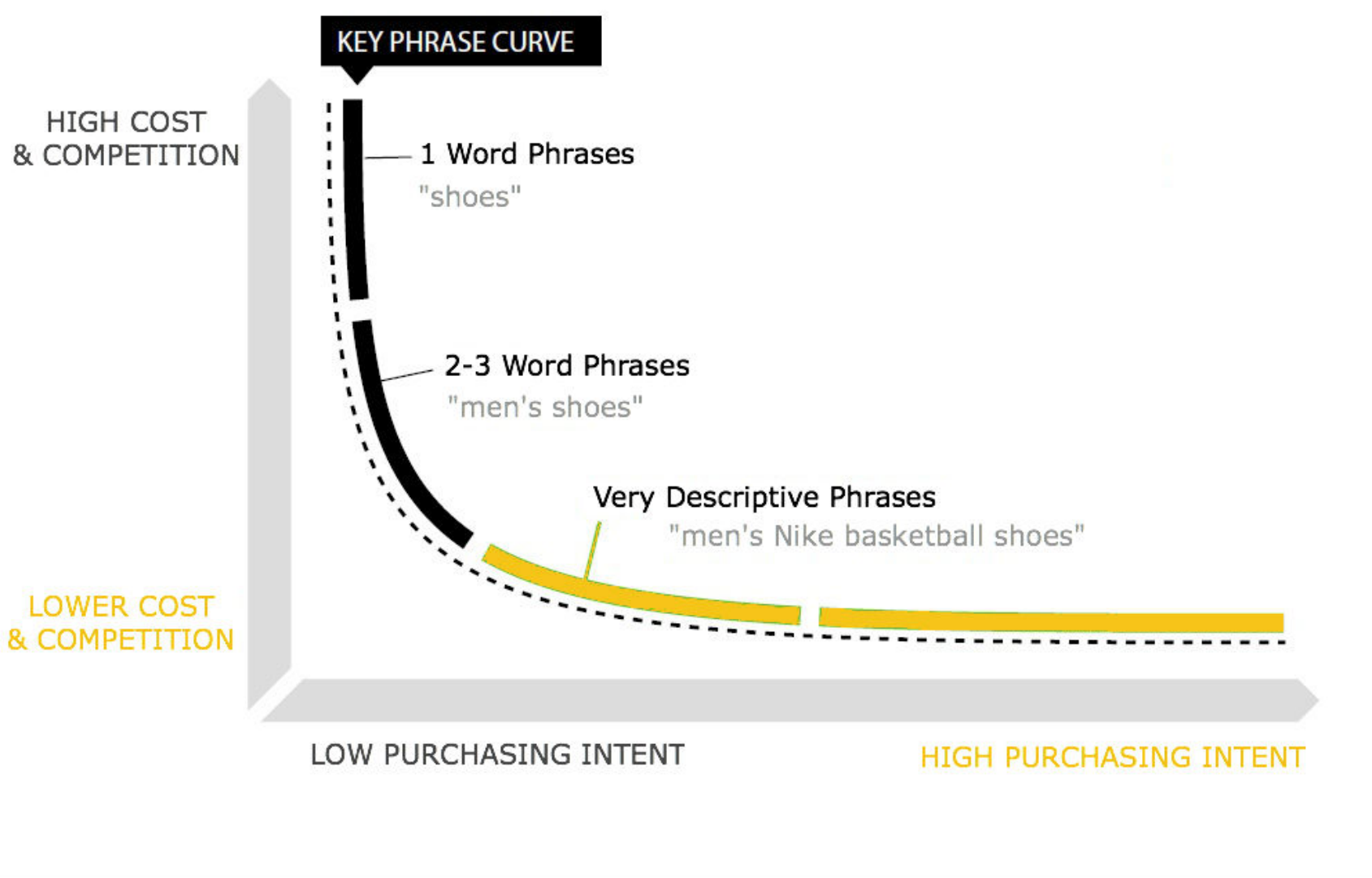
Commonality cloud

Comparison cloud




Word network
freq
5000
10000
15000
20000
freq
20000
15000
10000
5000
book
review
read
see
can
easi
will
get
use
new
one
buy
best
cell
phone
check
email
now
right
movi
watch
video
good
retti
much
ter
market
devic
work
great
want
well
product
ipad
thing
compar
like
just
feel
connect
wifi
game
play
tech
support
long
listen
time
music
batteri
even
though
brows
page
browser
surf
system
oper
ipod
touch
screen
protector
usb
tv
power
button
micro
show
bird
credit
card
slot
samsung
galaxi
tab
hard
drive
life
high
custom
recommend
servic
sure
make
go
back

Long Tail + Google Trends: SEO Keywords



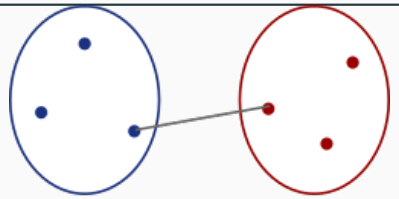
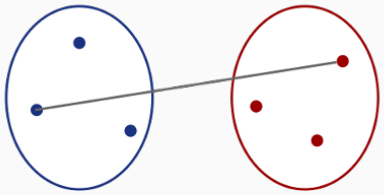
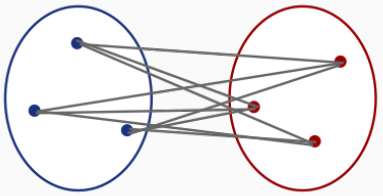
Clustering

Metric distance, hierarchial clustering

Hierarchical Agglomerative Clustering = HAC

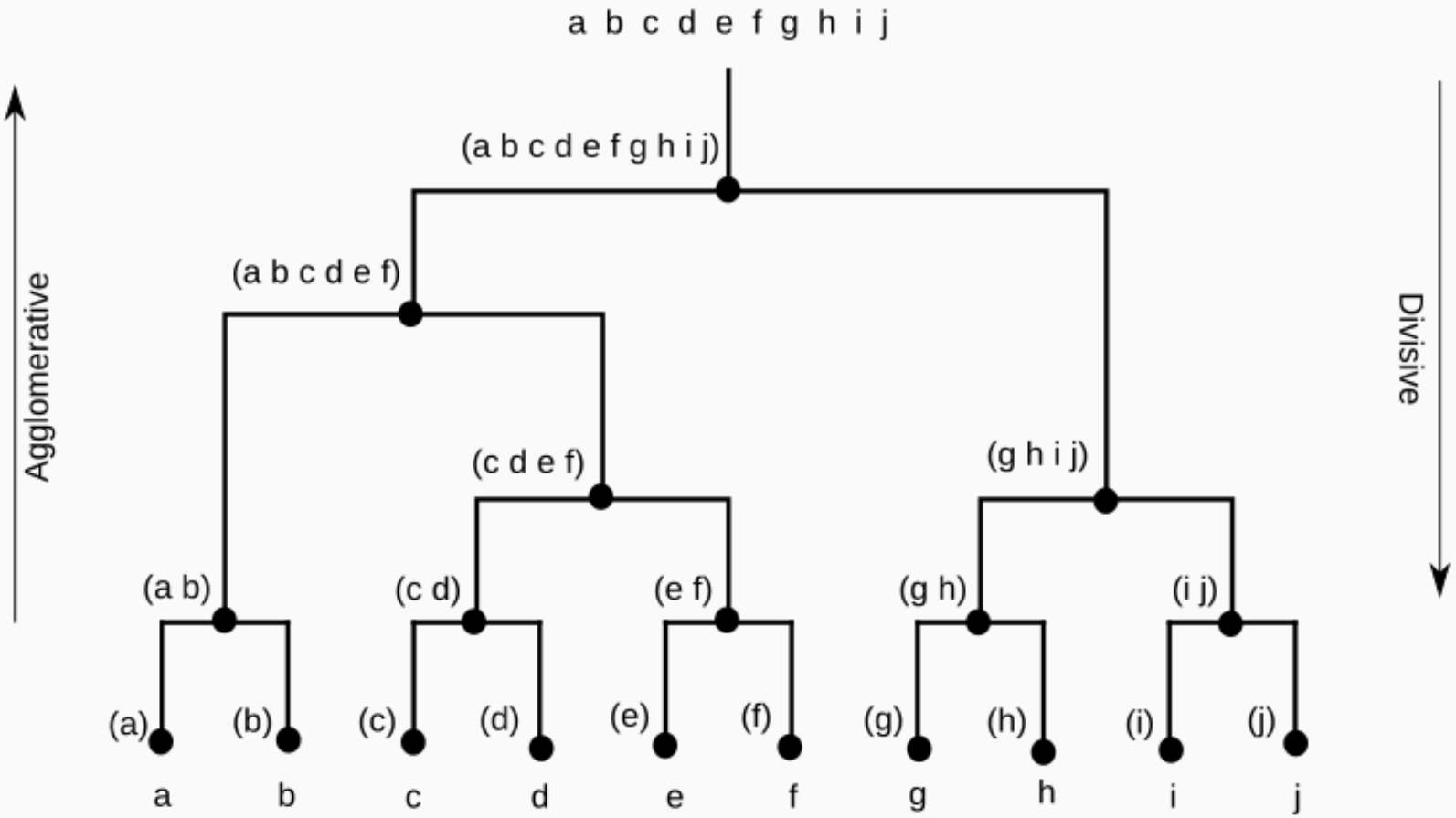
- Single-nearest distance, or single linkage;
- Complete-farthest distance or complete linkate;
- Average-average distance or average linkage.

The choice of a linking criterion determines how the distance between two groups of observations is calculated during the agglomeration procedure.

Type	Description	Image
Single linkage	This is the distance between the closest members of the two clusters	 Single linkage: minimum distance
Complete linkage	This is the distance between the members that a farthest apart	 Complete linkage: maximum distance
Average linkage	This method involves looking at the distances between all pairs and averages them. This is also called Unweighted Pair Group Mean Averaging	 Average linkage: average distance

Some linkage types

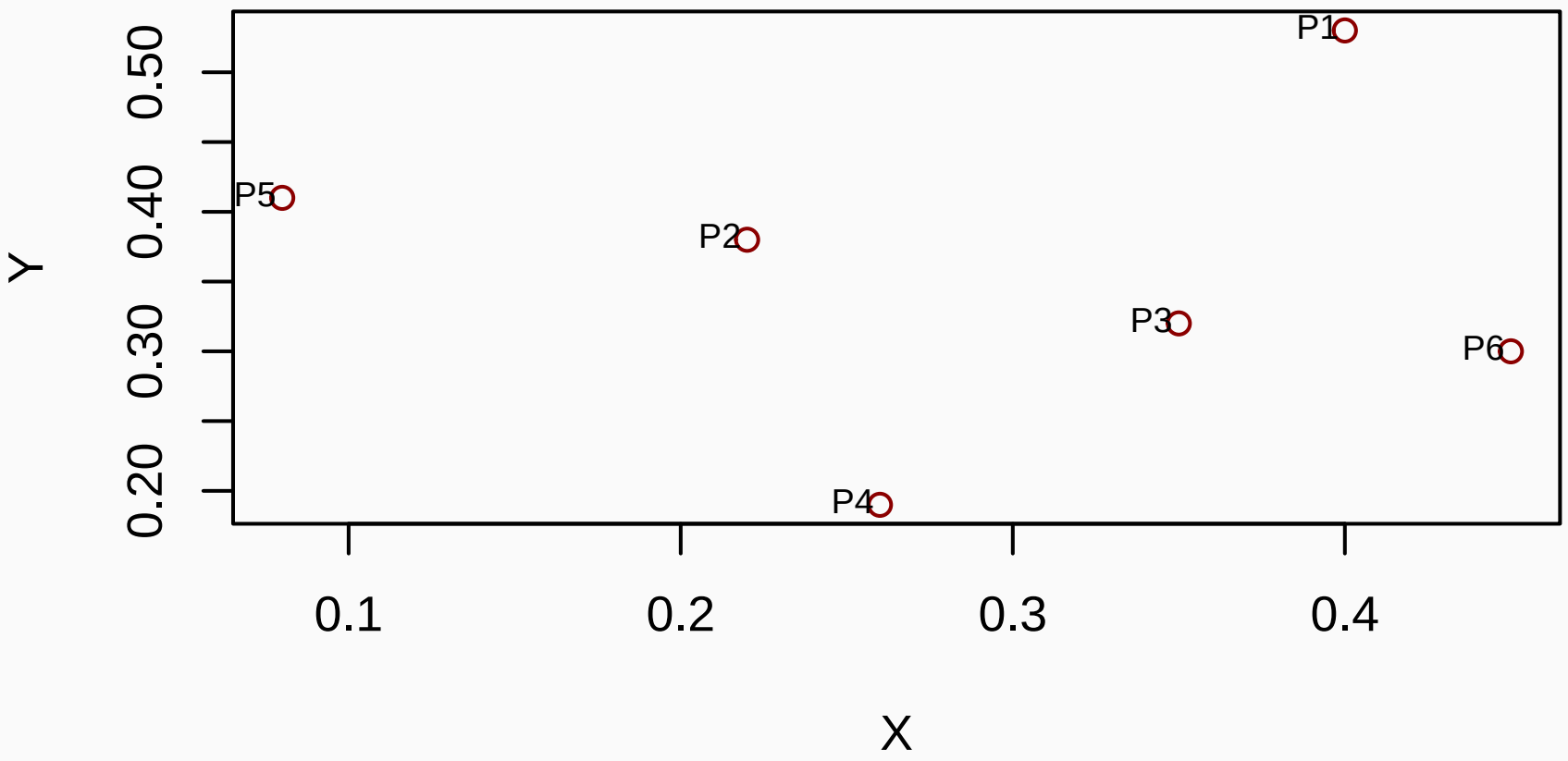
Clustering - HAC



Clustering - Example

	X	Y
P1	0.40	0.53
P2	0.22	0.38
P3	0.35	0.32
P4	0.26	0.19
P5	0.08	0.41
P6	0.45	0.30

Clustering - Example



Clustering - The distance matrix

Distance $D(x_1, y_1), (x_2, y_2) = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

Distance $D(P_1), (P_2) = \sqrt{(0.40 - 0.22)^2 + (0.53 - 0.38)^2} = \sqrt{(0.18)^2 + (0.15)^2} = \sqrt{0.0324 + 0.0225} = \sqrt{0.0549} = 0.23$

In an analogous way we calculate the distance between P_1 and P_3, $(P_1, P_4), (P_1, P5)$ ...

	P_1	P_2	P_3	P_4	P_5	P_6
P_1	0					
P_2	0.23	0				
P_3	0.22	0.15	0			
P_4	0.37	0.20	0.15	0		
P_5	0.34	0.14	0.28	0.29	0	
P_6	0.23	0.25	0.11	0.22	0.39	0

Clustering: New distance matrix

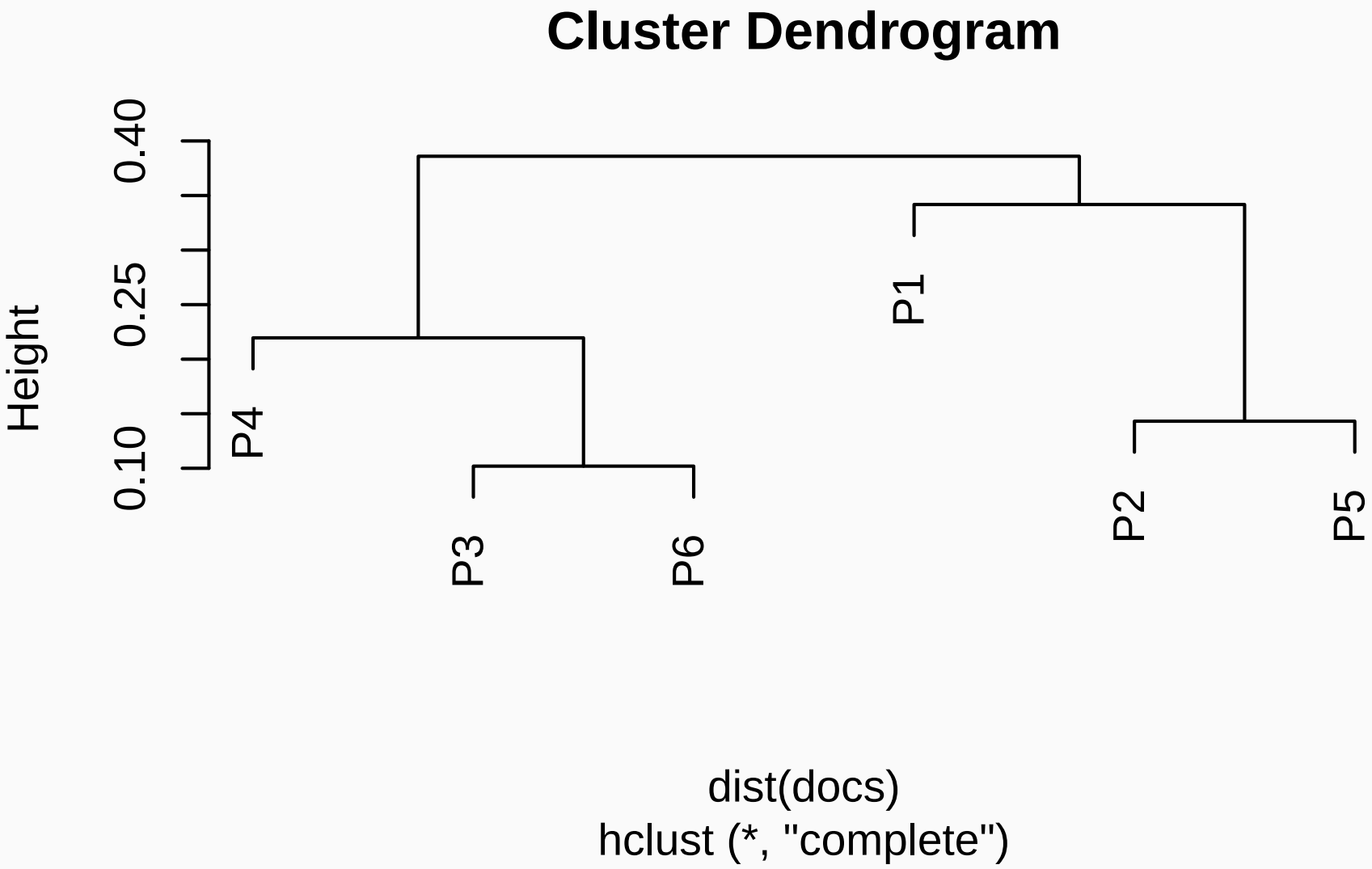
	P_1	P_2	(P_3, P_6)	P_4	P_5
P_1	0				
P_2	0.23	0			
(P_3, P_6)	$(0.22, 0.23)$	$(0.15, 0.25)$	0		
P_4	0.37	0.20	$(0.15, 0.22)$	0	
P_5	0.34	0.14	0.28	0.29	0

Clustering: New distance matrix ctd.

	P_1	(P_2, P_5)	(P_3, P_6)	P_4
P_1	0			
(P_2, P_5)	$(0.23, 0.34)$	0		
(P_3, P_6)	$(0.22, 0.23)$	$(0.15, 0.25), (0.28, 0.39)$	0	
P_4	0.37	0.20	$(0.15, 0.22)$	0

Clustering: New distance matrix ctd.

	P_1	$(P_3, P_6)(P_2, P_5)$	P_4
P_1	0		
$(P_3, P_6)(P_2, P_5)$	$(0.22, 0.23)(0.23, 0.34)$	0	
P_4	0.37	$(0.15, 0.22)(0.20, 0.25)$	0


Cluster Dendrogram
Height
0.10
0.25
0.40
P4
P3
P6
P1
P2
P5
dist(docs)
hclust (*, "complete")

Clustering, Topic Models, and Sentiments

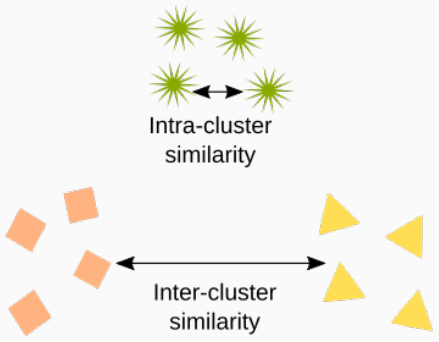
Text clustering (continued)

We, humans, have the tendency to categorize, it is a shortcut method of our minds to put together things that have common features and deal with them as groups instead of single items. Categorization is part of our human cognitive nature and allows us to act properly when a specific, already categorized situation arises. It is also an excellent path to evaluate homogeneity in the observed data so that we can make decisions on restricted number of groups with specific characteristics. It is like summarizing, we condense a lot of information in a few important points that well describe the whole. Knowing some features of these categories, we can go on and do clustering on them. There are several types of clustering techniques and there are also many fields of application. In Marketing, clustering can also be called segmentation when we try to find a segment of a population that behaves and acts in a predetermined way.

There are two big groups of clustering methods, the first one is called model-based and uses a statistical probability models, and the other is called distance-based which has a strong exploratory component. A similar concept to clustering is classification. Classification uses already known classes with already known features: new unclassified elements are tested against these features and then assigned to the appropriate class, so classification is a supervised method.

Text clustering

In any clustering algorithm we will be faced with the similitude concept of the features: very similar within a cluster (homogeneity), and less similar between the clusters (heterogeneity), So, when we apply a clustering algorithm, we are interested in how many clusters the algorithm creates, how distant they are from one another, their size, their shape and how dense they are.

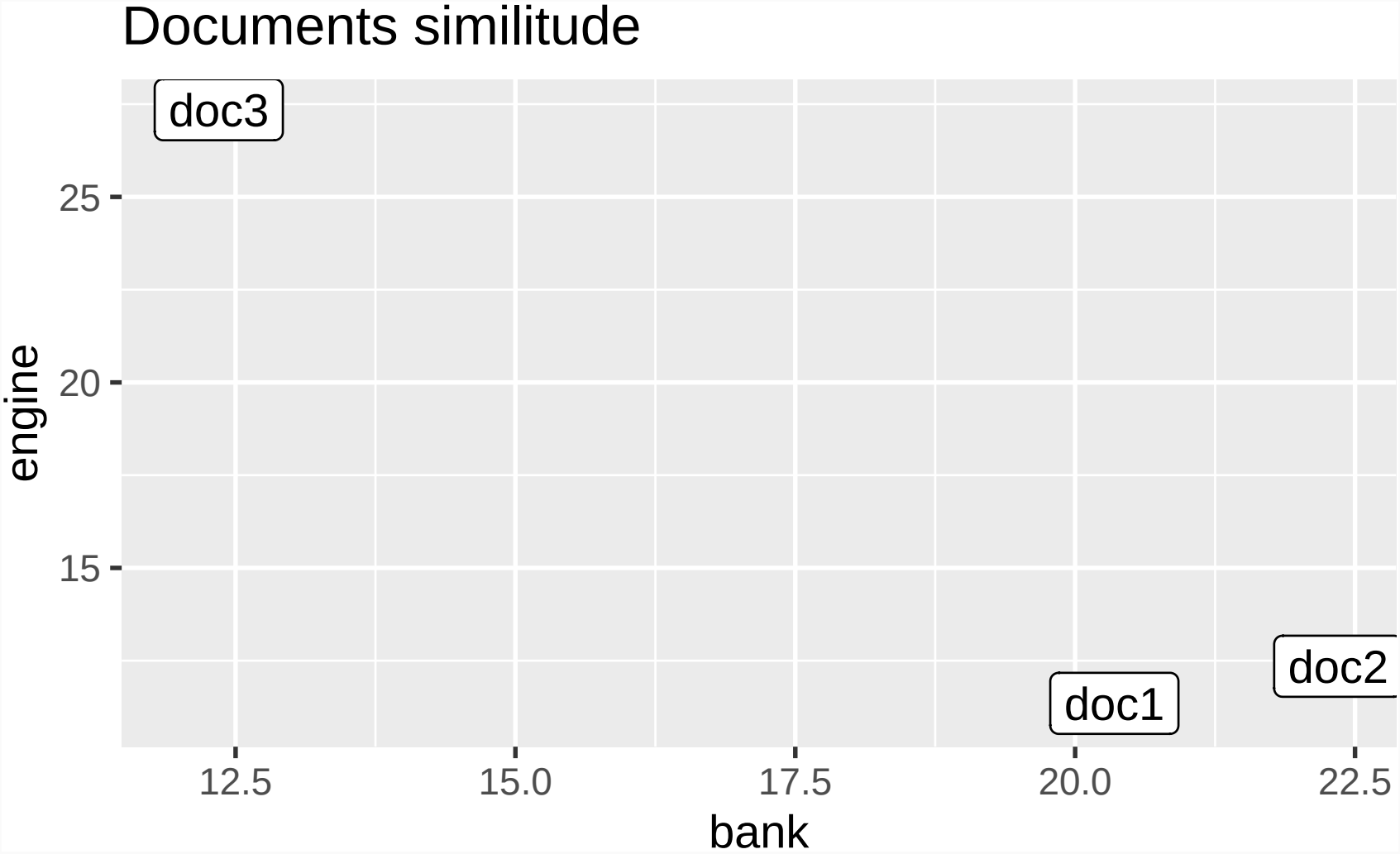


Text clustering will give us a quick overlook on the content of a corpus extracting the most representative features of it, will put together similar words in order to understand better the topics underneath. These methods can be very useful, for example, to cluster search results or to find similar text in the emails from the customers. This will allow you to segmentize better and also understand some common pain points in a one-to-one marketing approach. **So far, we have seen hyerarchical clustering**

Text clustering

Upgrade in code to have better-looking plots. Let us see some similitude of documents. The magrittr pipe. %>%

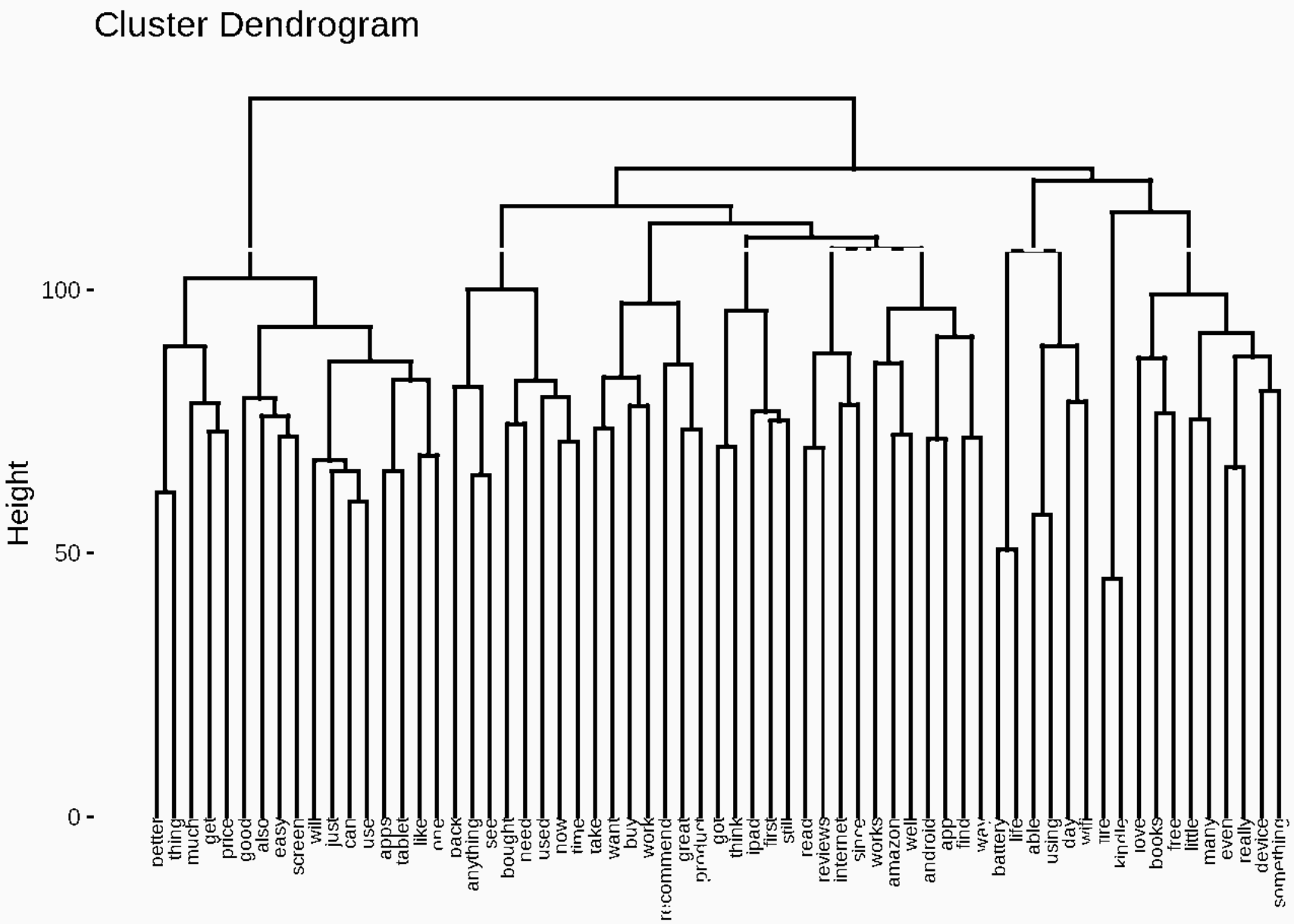
```
library(tidyverse)
library(magrittr)

# Example, the word bank (appears 20 times in the first document),
# engine (appears 11 times in the same document)

doc1 <- c(20, 11)
doc2 <- c(22, 12)
doc3 <- c(12, 27)

docs <- rbind(doc1, doc2, doc3) %>%
  as.data.frame %>%

  rename(bank=1, engine=2)

ggplot(docs, aes(x=bank, y=engine)) +
  geom_point() +
  labs(title="Documents similitude", x="bank", y="engine") +
  geom_label(label=rownames(docs), nudge_x = 0.35, nudge_y = 0.35)
```


Documents similitude
doc3
doc1
doc2
25
20
15
engine
12.5
15.0
17.5
20.0
22.5
bank

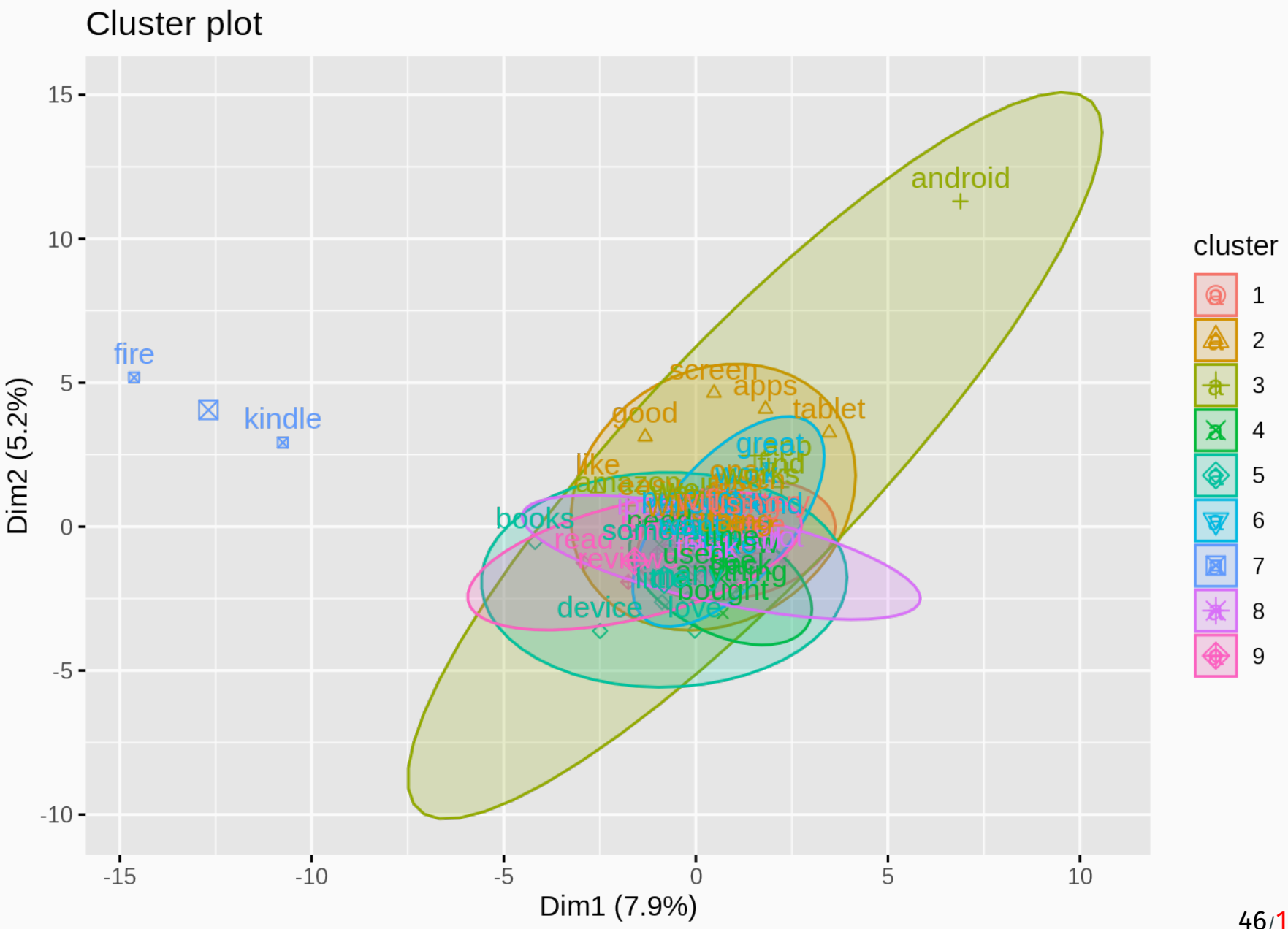
Text clustering

```r
# we transform the term-document matrix into weighted term-document matrix
library(factoextra)
library(NbClust)
library(proxy)

# calculate the optimal number of clusters and save it into the variable nchac
nchac <- NbClust(as.matrix(tdm), distance = "canberra", min.nc = 1,
                 max.nc = 20, method = "ward.D2", index = "ch")

# extract the optimal number of clusters from the nc structure and print it
#(the parenthesis are used for that)
(nbc <- as.vector(nchac$Best.nc[1]))

# check how many words were assigned to each cluster
table(nchac$Best.partition)

# calculate the hierarchical clustering and save it into the dend variable
dend <- tdm %>%
  as.matrix %>%
  dist(method = "canberra") %>%
  hcut(k = nbc, hc_method = "ward.D2")

# visualize the dendrogram
fviz_dend(dend, cex = 0.5, rect = TRUE, k_colors ="black", rect_border = "lightgray",
          rect_lty = 2)
```

Text clustering: HAC Dendrogram



Text clustering: HAC



Text clustering: K-Means

K-Means (or **k**-means) is a clustering algorithm. Unlike HAC, it is not hierarchical. It starts off with a k number of centroids randomly assigned. Then it determines the centroid using the Euclidean distance and groups the objects based on the minimum distance.

Text clustering: K-Means, example

	X	Y
P1	0.40	0.53
P2	0.22	0.38
P3	0.35	0.32
P4	0.26	0.19
P5	0.08	0.41
P6	0.45	0.30

Example, k=2. Consider the centroids to be randomly assigned, we will take the first two points as the initial centroids.

Point/Cluster	X	Y
P1/K1	0.40	0.53
P2/K2	0.22	0.38

The distance from cluster 1 made of $P1$ to cluster 1 made of $P1$ is 0. The distance from cluster 2 made of $P2$ to cluster 2 made of $P2$ is 0.

The general formula for the distance is
$[(x_1, y_1), (x_2, y_2)] = \sqrt{(x_1 - x_2)^2 + (y_1 - y_2)^2}$

The distance from $P1$ to $P2$,
Dist$[P1, P2] = \sqrt{(0.40 - 0.22)^2 + (0.53 - 0.38)^2} =$
$\sqrt{(0.18)^2 + (0.15)^2} = \sqrt{0.0324 + 0.0225} = \sqrt{0.0549} = 0.23$

Point	Centroid		
	P1	P2	Assignment
P1	0	0.23	K1
P2	0.32	0	K2

So, the distance between point $P1$ and point $P2$ is 0.23

Text clustering: K-Means

	X	Y	Centroid
P1	0.40	0.53	K1
P2	0.22	0.38	K2
P3	0.35	0.32	?
P4	0.26	0.19	?
P5	0.08	0.41	?
P6	0.45	0.30	?

Distance $[P1, P3] = \sqrt{(0.40 - 0.35)^2 + (0.53 - 0.32)^2} = 0.22$

Distance $[P2, P3] = \sqrt{(0.22 - 0.35)^2 + (0.38 - 0.32)^2} = 0.14$

Min(Distances$(0.22, 0.14)) = 0.14$. Assign to cluster 2.

Point	Dataset	Cluster 1	Cluster 2	Assignment
P3	(0.35, 0.32)	0.22	0.14	2

Text clustering: K-Means

Point	Dataset	Cluster 1	Cluster 2	Assignment
P3	(0.35, 0.32)	0.22	0.14	2

Update the centroid: ↓

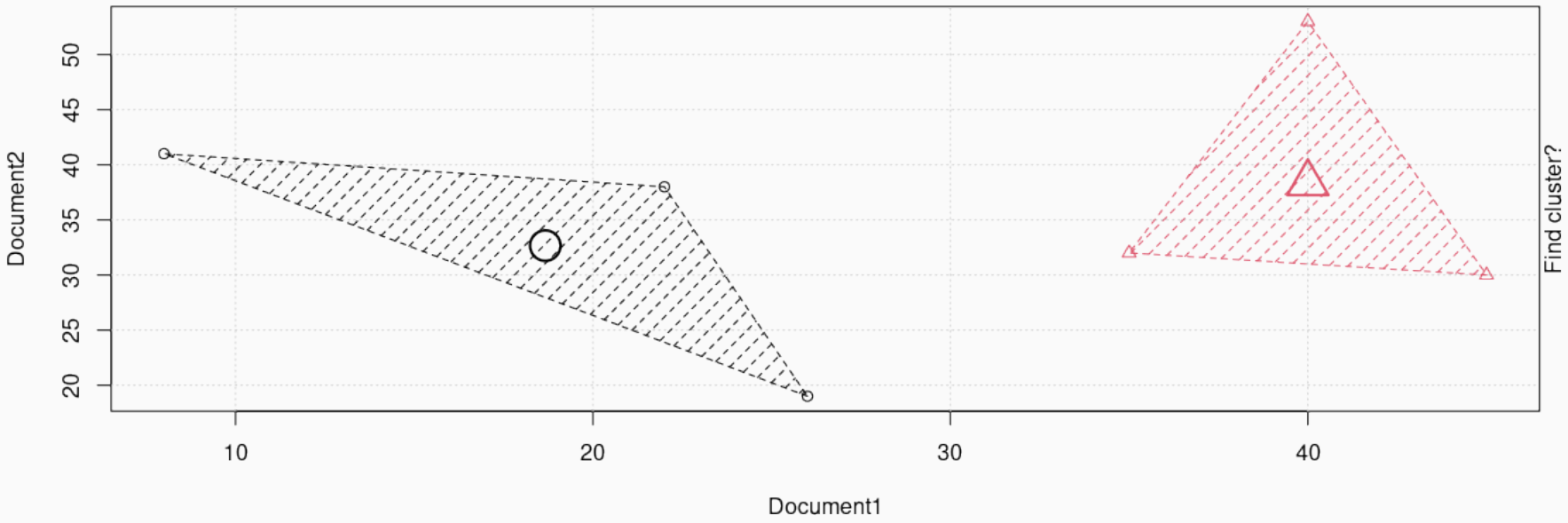
Cluster	X	Y
K1	0.40	0.53
K2	$\frac{(0.22+0.35)}{2} = 0.285$	$\frac{(0.38+0.32)}{2} = 0.35$

Text clustering: K-Means

	X	Y	Centroid
P1	0.40	0.53	K1
P2	0.22	0.38	K2
P3	0.35	0.32	K2
P4	0.26	0.19	?
P5	0.08	0.41	?
P6	0.45	0.30	?

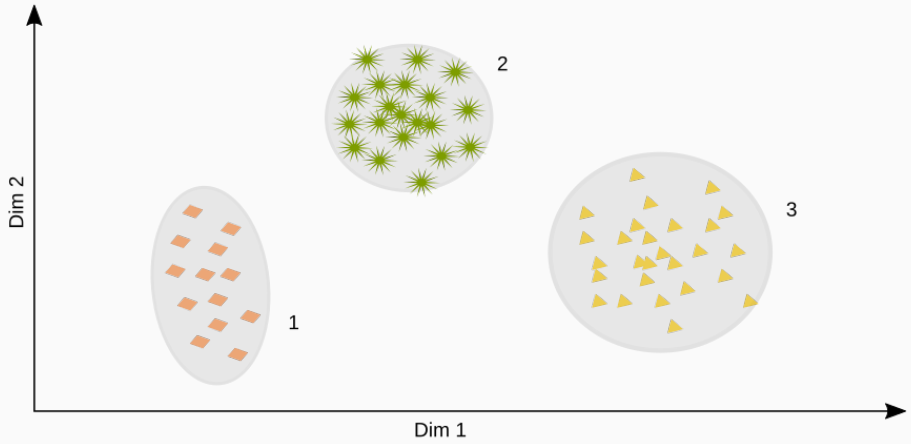
Text clustering: K-Means

(see code)



Text clustering: Spherical K-Means

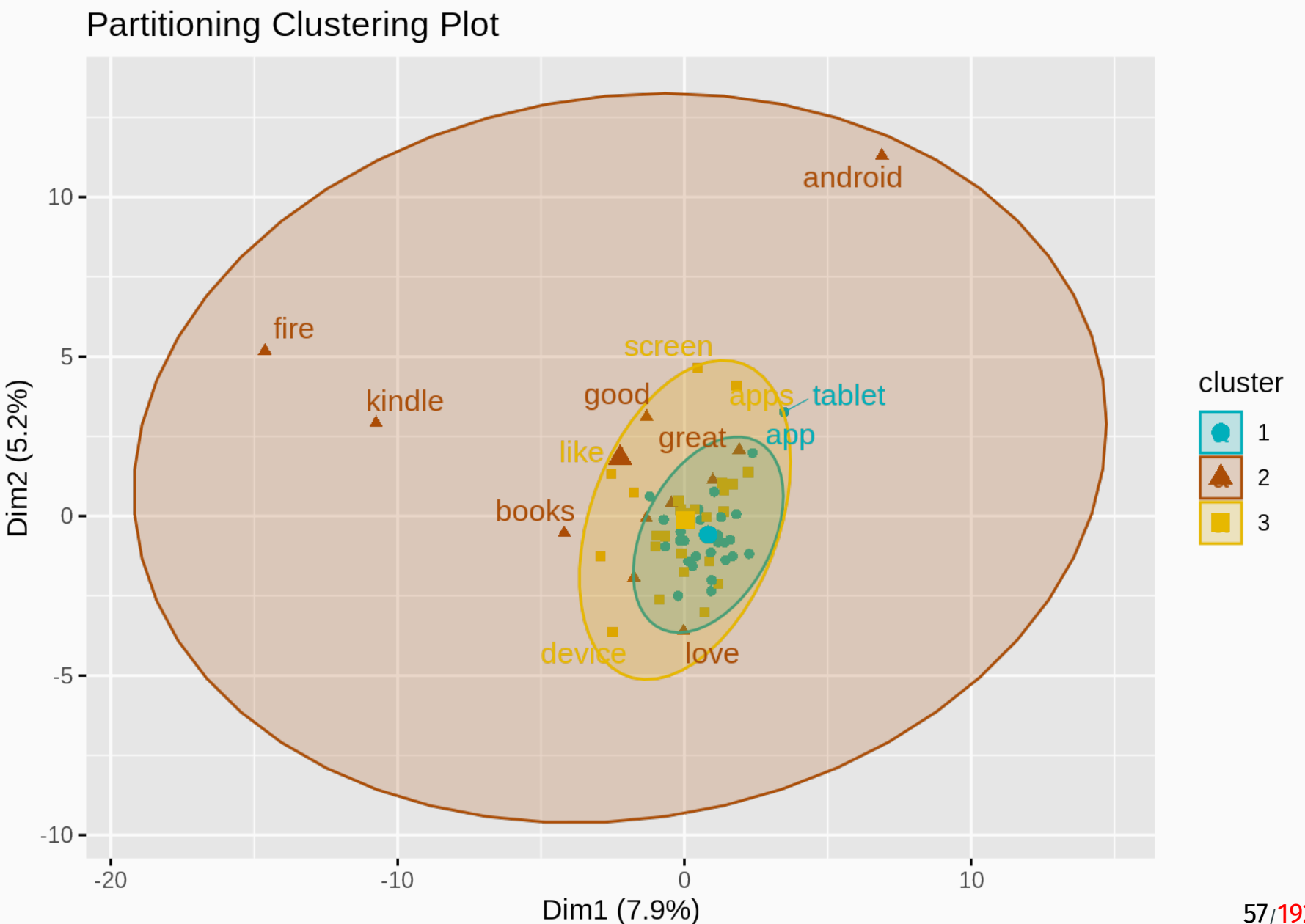
In the k-means clustering, there are some drawbacks: the algorithm does not perform well when the clusters differ in size and density, there are outliers in data, or the data are not distributed in a spherical way (or circular in a 2d plane). Circular means that the points that belong to a cluster are within a circle with the center of the cluster as center and radius equal to the distance to the furthest element. To put it in different terms, in a plane of two dimensions, we might better have a circular data spread such as clusters 2 and 3 than an oblong one as it is in cluster 1, which, of course, is not ideal.



Text clustering: K-Means

```r
set.seed(123)

# calculate the optimal number of clusters and save it into the variable nc
nckm <- NbClust(as.matrix(tdm), distance = "canberra", min.nc = 2,
                max.nc = 20, method = "kmeans", index = "ch")

# extract the optimal number of clusters from the nc structure and print it
(nbc <- as.vector(nckm$Best.nc[1]))

# calculate k-means assigning 3 clusters
km <- tdm %>%
  as.matrix %>%
  dist %>%
  kmeans(nbc, nstart = 30)

fviz_cluster(km, data = tdm,
             palette = c("#00AFBB", "#AB4E07", "#E7B800"),
             main = "Partitioning Clustering Plot",
             ellipse.type = "norm",
             repel = TRUE,
             show.clust.cent = TRUE
```

Text clustering: K-Means



Text clustering: K-Means

K-means is an easy to apply algorithm, suitable for big data sets. You just need to set the number of clusters and it does the rest even in cases when several variables are involved, due to its efficient computational cost. It adapts to a new clustering partition just by changing the k number. When the clusters are spherical (or round in a 2D plane) the algorithm is very efficient. With respect to the hierarchical clustering, the clusters are tighter, and the interpretation of the outcome is uncomplicated. It is hard to find an optimal number of clusters and each time one runs the algorithm my end with a different set of clusters and as such the results might be inconsistent. The output clusters have equal sizes no matter what the input data are, and the outcome cannot be assessed in terms of quality. A drawback can be that k-means considers only numerical values.

Gaussian Mixture Clustering

The name Gaussian Mixture Clustering (also known as Expectation Maximization clustering) hints at the fact that we deal with more than one clustering, hence the word Mixture and the data behaves in a Gaussian manner [ref Gaussian]. So, we can consider that the elements belong to several clusters that are mixed and as such we separate them in the corresponding cluster.

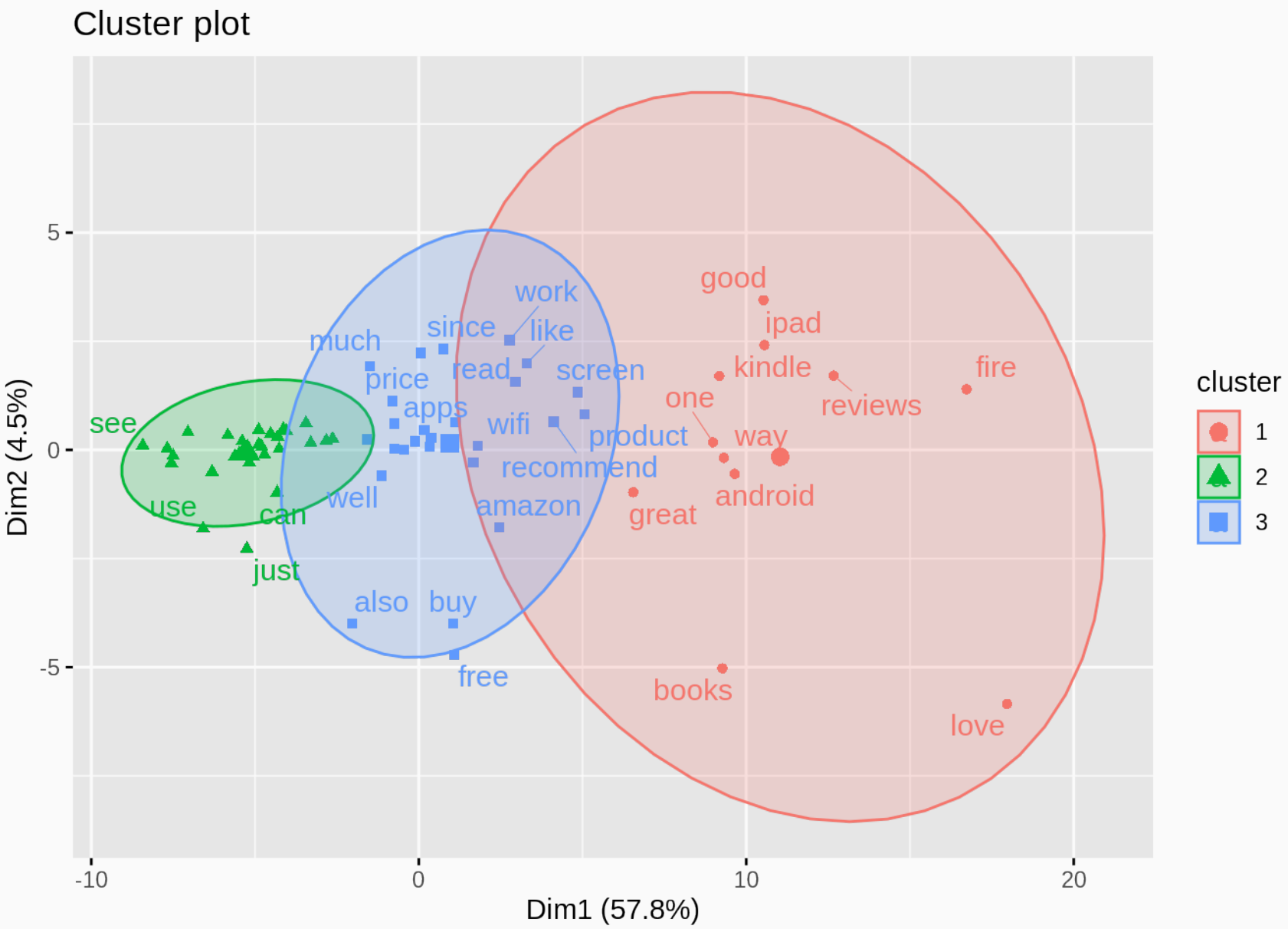
We have seen that the k-means algorithm cannot handle well situations of oblongated clusters.

Gaussian Mixtures Clustering comes into play in this situations but with a twist. This model introduces the fact that a probability is assigned to each element to belong to each cluster. As such, an element can belong to more than one cluster, and this is called soft clustering. This is in contrast with the k-means hard clustering where each element is assigned to a one and only cluster.

Gaussian Mixture Clustering

```r
library(ClusterR) # Gaussian Mixtures

# calculate the distance matrix and transform it into a matrix then into
# a data frame
dist.matrix.df <- tdm %>%
    as.matrix %>%
    dist %>%
    as.matrix %>%
    as.data.frame

# cluster creation with GMM and elements assignment
gmc <- dist.matrix.df %>%

GMM(gaussian_comps = nbc, dist_mode = "maha_dist", seed_mode = "random_subset",
    km_iter = 10, em_iter = 5, verbose = FALSE, var_floor = 1e-10, seed = 1)

predict_GMM(dist.matrix.df, CENTROIDS = centroids,
            COVARIANCE = covariance_matrices, WEIGHTS = weights)

# check how data was distributed
gmc$cluster_labels
```

Gaussian Mixture Clustering



Gaussian Mixture Clustering

With respect to the k-means, GMM clustering introduces flexibility with respect to the cluster variance. One could even consider k-means as GMM case where variables do not covariate. It deals well with the oblong or rotated clusters and allows¸ due to its probabilistic nature based on the Gaussian distribution, for multi assignment of the objects to different clusters while in k-means they must belong to exclusively one cluster. In text analytics a word may belong to several topics and not exclusively to one. GMM uses the mean and the variance to update the centroids, unlike the k-means when only the mean is considered. Like any algorithm, there is no perfect solution, GMM clustering does not perform well when the dataset is not Gaussian or when data is highly correlated, otherwise it can cluster well if you know ahead the number of clusters.

DBSCAN

Hierarchical Density-Based Spatial Clustering of Applications with Noise

What type of data are we trying to cluster?

- we assume that there are clusters in our data, and we try to discover them.
- we assume that there is noise in this data

Sometimes K-Means have low performance, but it performs nicely when we have spherical (or round) clusters, whey they have balanced sizes and densities and the last one is mostly in the centre, also noise and/or outliers destabilize the outcome. So, when there are arbitrary shapes, different sizes, noise and outliers... K-Means clusters have a hard time.

But

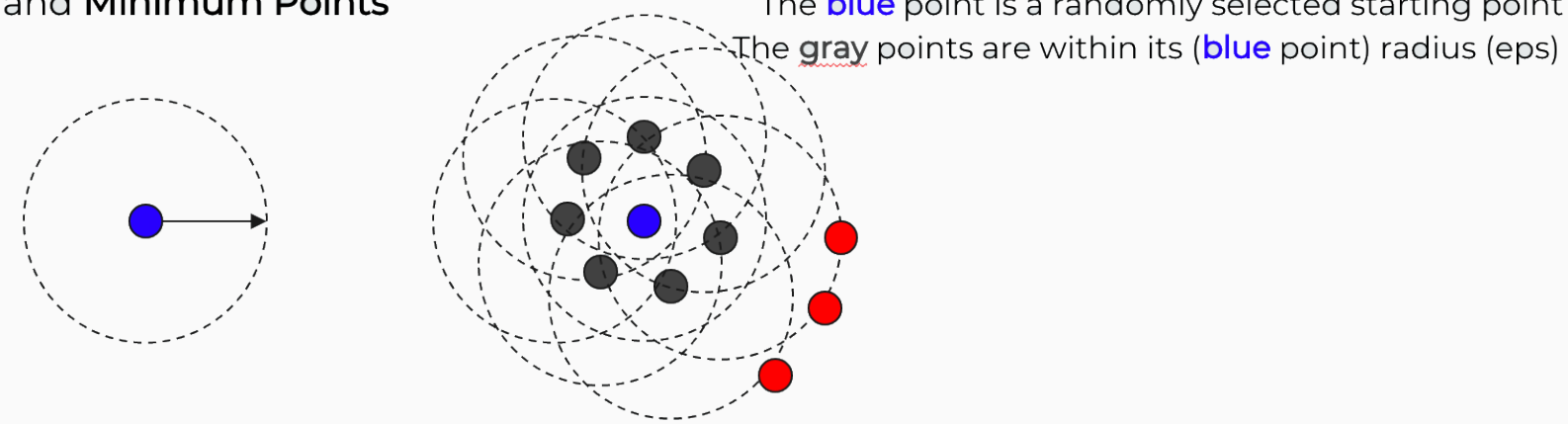
We have HDBSCAN which can work well when we have not perfectly shaped clusters, clusters can have different sizes and densities. Noise is also a natural part of the dataset and HDBSCAN considers that. It does not look for clusters with specific shape but for regions where data are denser with respect to the nearby space. Sometimes in a dataset we have very dense regions separated by very sparse regions. Density can be understood also from a probabilistic perspective where dense can be interpreted as probable. So denser means more probable.

DBSCAN uses two parameters that can be fine-tuned.

Hierarchical Density-Based Spatial Clustering of Applications with Noise

We have two important parameters:

Epsilon and **Minimum Points**

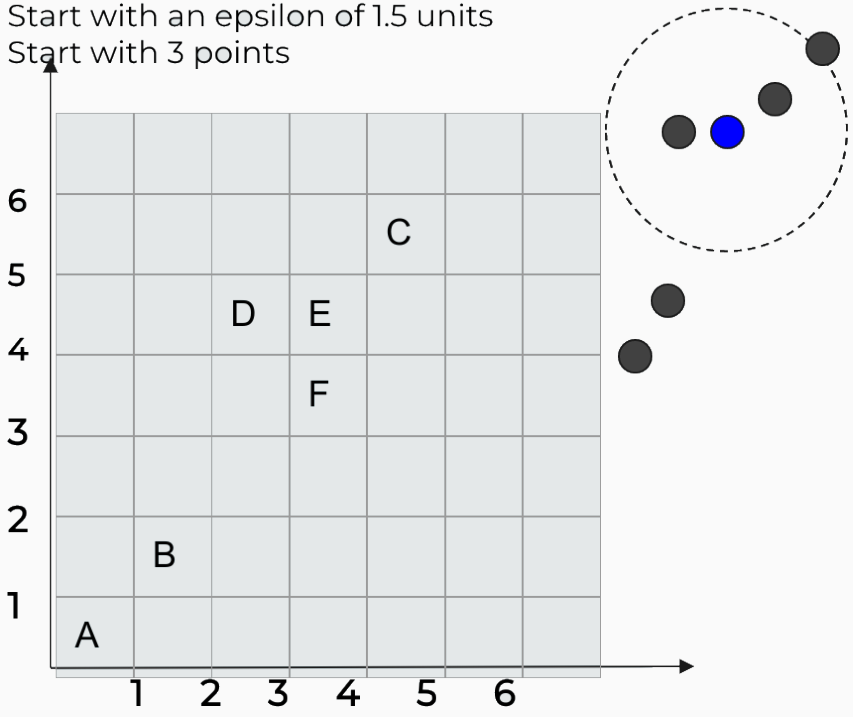


The minimum points are $7 + 1 = 8$ so are the minimum number of core points including the starting point. So minimum points must exist in order to start initiating a cluster! Once a cluster is formed, it will continue to grow because each point will broadcast their own area with radius $= \epsilon$.

The red points are called border points which cannot directly reach the starting point but can be chained to it through the epsilon of the other points. The newly acquired points will broadcast their epsilon out too until the cluster is finalized

DBSCAN

Start with an epsilon of 1.5 units
Start with 3 points

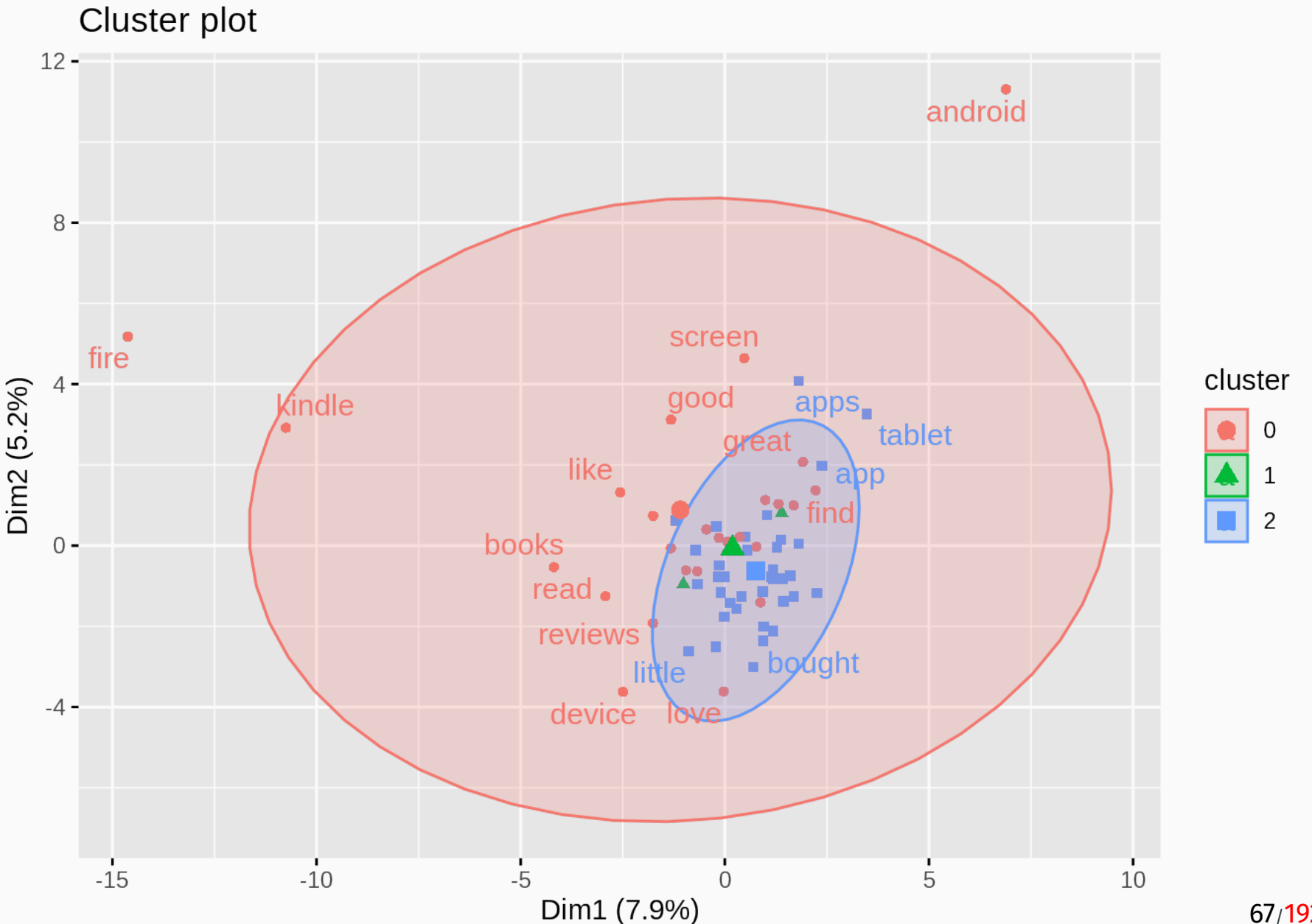


Points	A Noise	B Noise	C Border	D Core	E Core	F Core
A	0	0.7	5.7	3.6	4.2	3.2
B	0.7	0	4.9	2.9	3.5	2.5
C	5.7	4.9	0	2.2	1.4	2.5
D	3.6	2.9	2.9	0	1	5
E	4.2	3.5	1.4	1	0	1.1
F	3.2	2.5	2.5	5	1.1	0

DBSCAN

```r
library(dbscan)

# calculate the hdbscan cluster
clustering.dbscan <- hdbscan(tdm, minPts = 2)
clustering.dbscan$cluster

# visualize the clusters created with hdbscan
fviz_cluster(list(data = as.matrix(tdm), cluster = clustering.dbscan$cluster),
             ellipse.type = "norm",
             repel = TRUE)
```

DBSCAN


Cluster plot
12
8
4
0
-4
Dim2 (5.2%)
-15
-10
-5
0
5
10
Dim1 (7.9%)
android
fire
kindle
screen
good
great
like
books
read
reviews
device
little
love
apps
tablet
app
find
bought
cluster
0
1
2

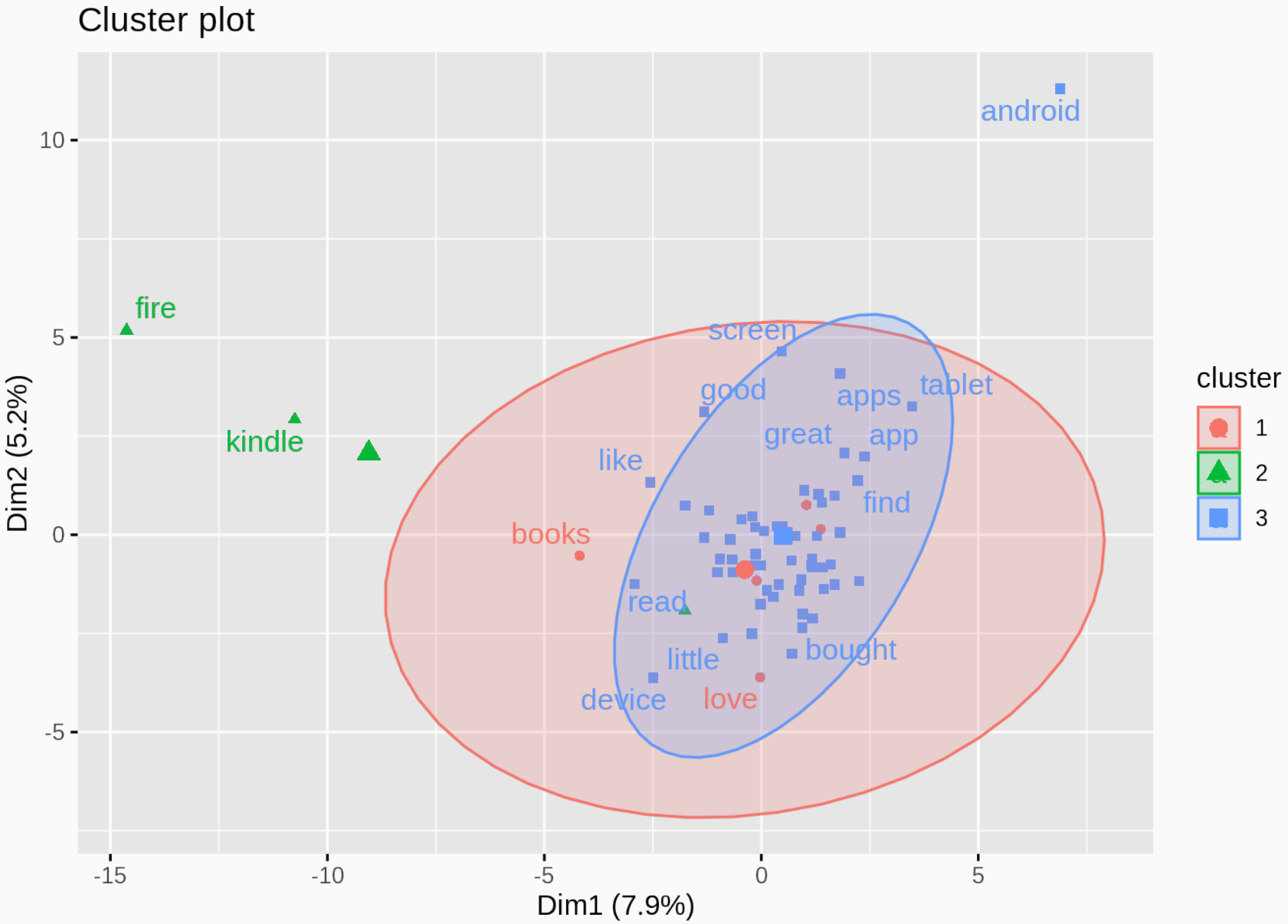
Affinity propagation clustering

Affinity propagation clustering introduces the concept of “exemplars”. These are somewhat the core of a data set around which clustering is done. The idea behind the concept is that all the elements are potential exemplars and a message system between them allows for selecting a set of them and the clusters around them. The package we used here is `apcluster`. Unlike in the previous clustering algorithms, the data can be of any shape and variable density. Another important thing is that the basic version of `apc` will calculate the optimal number of clusters iteslf. Nevertheless, we can use the function `apclusterK()` in the `apcluster` package that allows us to set the number of clusters beforehand. The first attribute, s, in the function, calculates the similarity function using the negative squared distances. If this is not suitable, you can use other similarity methods based on Spearman correlation.

Affinity propagation clustering

```r
# affinity propagation
library(apcluster)
library(reshape2)

# apply the clustering algorithm to extract 3 clusters
apck <- apclusterK(s=negDistMat(r=2), x=as.matrix(tdm), K=3)

# convert data from a cluster type into a data frame
apckmelt <- melt(apck@clusters)

# visualize the data
fviz_cluster(list(data = as.matrix(tdm)[apckmelt$value,],
            cluster = apckmelt$L1), ellipse.type = "norm",
            repel = TRUE)

# check the words that belong to each cluster
apck
```

Affinity propagation clustering



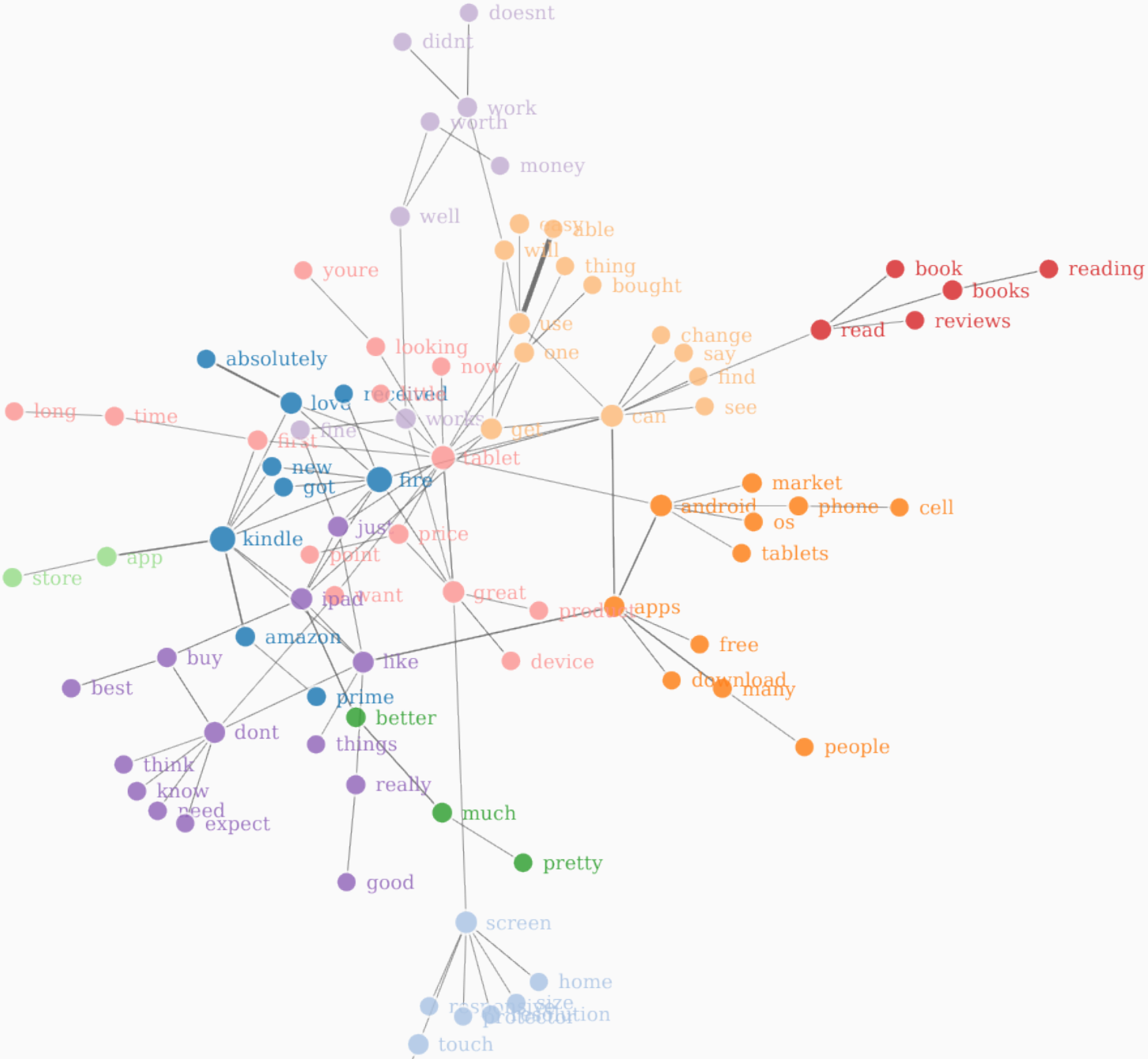
Affinity propagation clustering

Each cluster has the “exemplar” cluster, a prominent word that best represents the cluster. It is a word that should define the other words that come in the same cluster with it. It can be considered the “topic” of the cluster. In our case the exemplars are **free**, **kindle** and **see**. For the practical part of this lecture, check the words in each cluster and see if you can relate them to the exemplar.

Considerations

We might ask ourselves what algorithm performs best and/or gives the best number or clusters? While there are methods and packages to compare that such as **cValid**, the experience of the researcher will make a difference because it is important to know what is your dataset, how is data distributed, etc. Making exercises to hone your skills is the best approach in the case of unsupervised clustering methods.

Networking


doesnt
didnt
work
worth
money
well
easy
able
will
thing
bought
youre
use
one
change
say
find
see
can
looking
now
absolutely
love
received
long
time
first
fine
works
get
tablet
new
got
fire
kindle
just
price
point
app
store
ipad
want
great
product
apps
amazon
buy
like
device
best
prime
better
dont
things
think
know
need
expect
really
much
pretty
good
screen
home
size
protection
touch
book
reading
books
reviews
read
market
android
phone
cell
os
tablets
free
download
many
people

Topics

What is a document?

Is a collection of words, a bag of words.

What is a model? What about a Document Model? Word, Document, Corpus.

What is a Topic Model?

What is a topic?

A document is a latent mixture of topics

$$p(w|d) = p(w|z) \cdot (z|d)$$

LDA

Topics

```
gene      0.04
dna       0.02
genetic   0.01
...
```

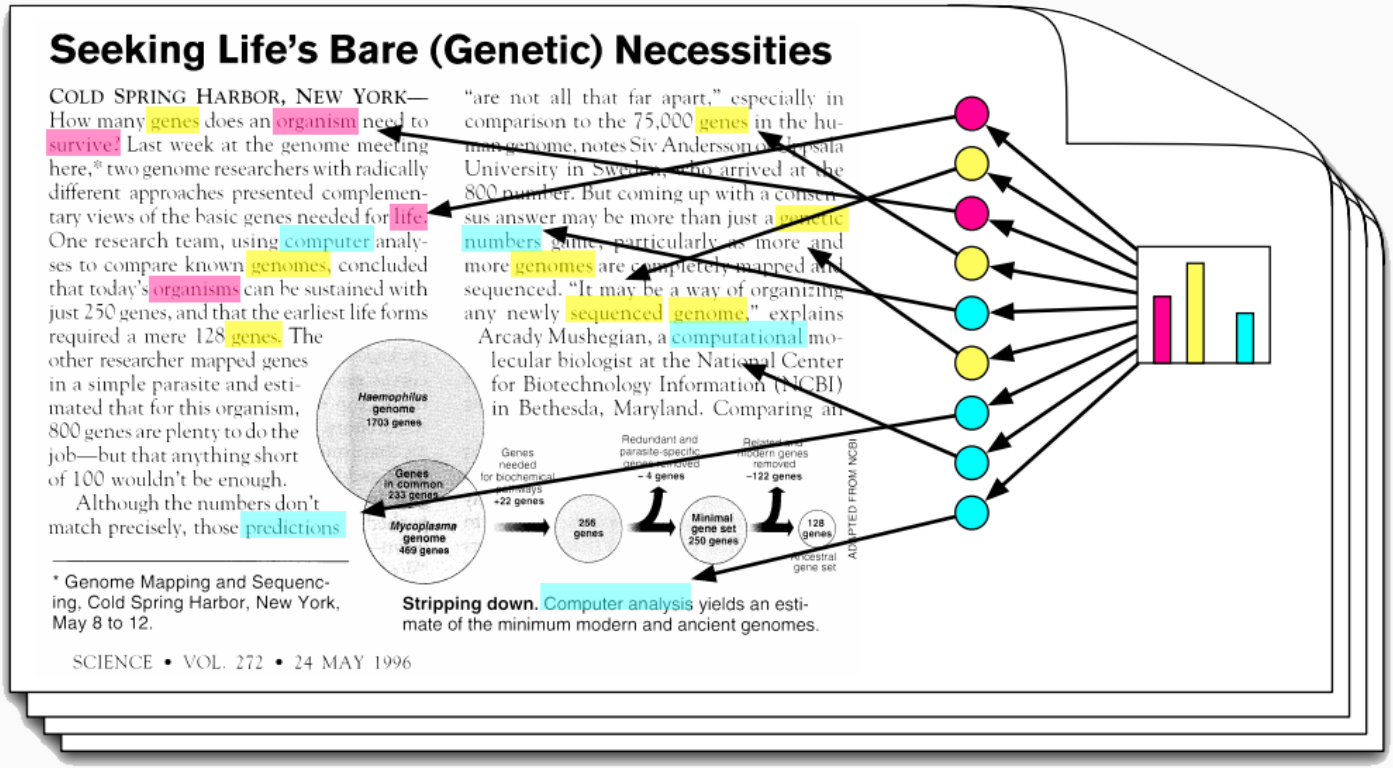
```
life      0.02
evolve    0.01
organism  0.01
...
```

```
brain     0.04
neuron    0.02
nerve     0.01
...
```

```
data      0.02
number    0.02
computer  0.01
...
```

Documents

Topic proportions & assignments



Seeking Life's Bare (Genetic) Necessities

COLD SPRING HARBOR, NEW YORK—How many genes does an organism need to survive? Last week at the genome meeting here,* two genome researchers with radically different approaches presented complementary views of the basic genes needed for life. One research team, using computer analyses to compare known genomes, concluded that today's organisms can be sustained with just 250 genes, and that the earliest life forms required a mere 128 genes. The other researcher mapped genes in a simple parasite and estimated that for this organism, 800 genes are plenty to do the job—but that anything short of 100 wouldn't be enough.

Although the numbers don't match precisely, those predictions "are not all that far apart," especially in comparison to the 75,000 genes in the human genome, notes Siv Andersson of Uppsala University in Sweden, who arrived at the 800 number. But coming up with a consensus answer may be more than just a genetic numbers game, particularly as more and more genomes are completely mapped and sequenced. "It may be a way of organizing any newly sequenced genome," explains Arcady Mushegian, a computational molecular biologist at the National Center for Biotechnology Information (NCBI) in Bethesda, Maryland. Comparing an

Stripping down. Computer analysis yields an estimate of the minimum modern and ancient genomes.

* Genome Mapping and Sequencing, Cold Spring Harbor, New York, May 8 to 12.

SCIENCE • VOL. 272 • 24 MAY 1996

Topic models

We have a set of documents.

We need to know what these documents talk about.

The topic is over the entire vocabulary set.

The topic proportion: in each document we understand what the subject is.

Topic modelling can be seen as a dimensionality reduction tool. We can also classify documents using the document proportions.

With these tools we try to get a gist of the text and we know that the words are the bricks for a beautiful construction. Topics can be determined by paying close attention to what kind of words appear into it, which of them could be good indicators of a particular topic, some are fuzzier. We might need to set up the themes in advance. Topics are very good for huge text corpora. Text Mining creates aggregates that, in the topics, creates aggregates that seem to be more meaningful than the words that constitutes them.

Topic models

Topics are studied (together) with the words that constitutes them. Words can also mislead us.

We might try to find different meanings of the text but LDA, the algorithm used for getting topics, is a tool for information retrieval. Sometimes reading topics can be potential for groundless or even misleading insights!

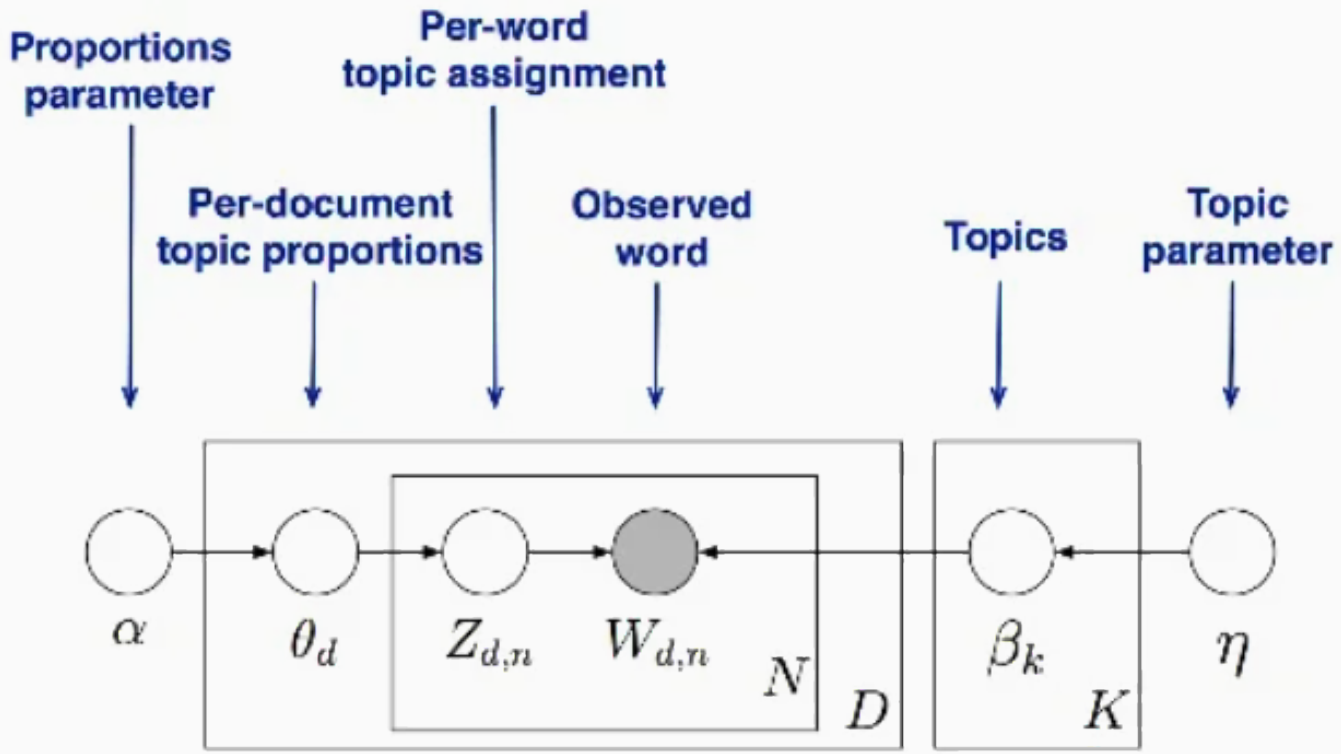
For a topic to be called so, it must be:

Coherent, meaning that the words must have a number of features (things) in common.

Stable, meaning that if a topic appears at the same rate (or a similar one) in two different types of documents, it means essentially the same thing in both.

Topics are useful because they describe things that resemble “concepts”, “discourses” or “fields”.

Topic models



$$p(\beta, \theta, z, w | \alpha, \eta) = \prod_{i=1}^{K} p(\beta_i | \eta) \prod_{d=1}^{D} p(\theta_d | \alpha) \left(\prod_{n=1}^{N} p(z_{d,n} | \theta_d) p(w_{d,n} | \beta_{1:K}, z_{d,n}) \right)$$

Source: Blei. ICML 2012 tutorial

Topic models

η and α are model parameters.

β is distributed as a Dirichlet with parameter $\eta_{1,K}$ - per corpus topic distribution θ is distributed as a Dirichlet with parameter $\alpha_{1,D}$ - per document topic proportion

For j in $(1, N_d)$

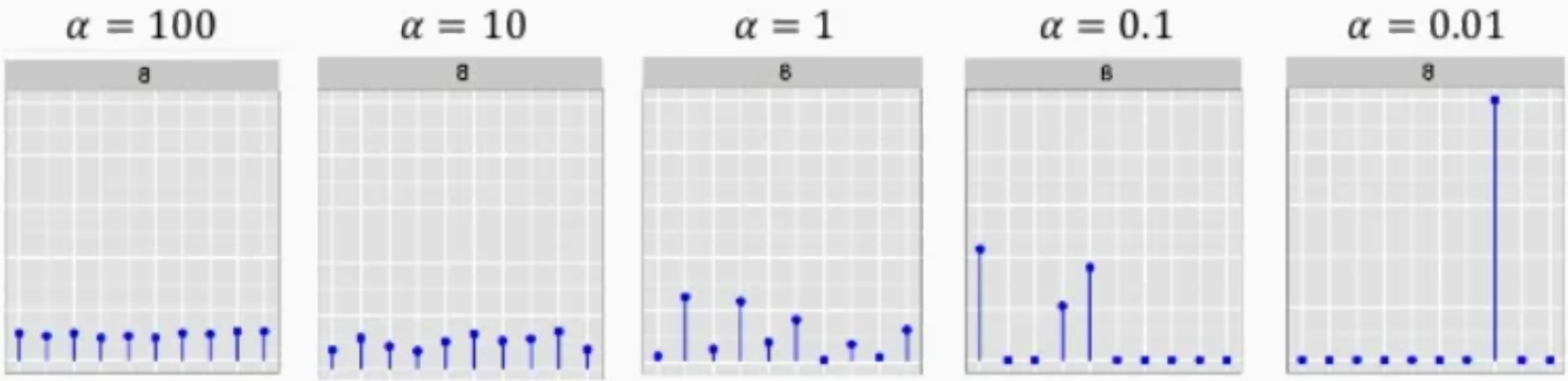
Select topic $z_{d,n}$ distributed as Multinomial(θ_d)

Select words $w_{d,n}$ distributed as Multinomial$(w_{d,n}|z_{d,n}, \beta_k)$

- Dirichlet distribution is *conjugate prior* of Multinomial

$$p(\theta|\alpha) = \frac{\Gamma(\sum_{i=1}^{k} \alpha_i)}{\prod_{i=1}^{k} \Gamma(\alpha_i)} \theta_1^{\alpha_1 - 1} \cdots \theta_k^{\alpha_k - 1}$$

- The parameter α controls the shape and sparsity of θ
  - high α = uniform θ, small α = sparse θ



Topic models

How does it work?

Assign randomly

See probabilities of words assigned to topics

And topics to documents

Update these probabilities until convergence.

(See code)

Feeling good?

What is a feeling?

What is a sentiment?

When a person speaks about their feelings, they are referring to their overall emotional state or reactions to someone or something. Feelings are a person's level of sensitivity or susceptibility to impression either good or bad. A person's feelings also relate to their tendency to become upset about things done or said. In addition, a feeling is a sensation, particularly through skin.

A 'sentiment' is a specific emotion, attitude or opinion. Quite often it refers to an emotional expression of love, sympathy, kindness or another strong positive feeling. In addition, it can be a thought, feeling or sense.

'sentiment' is closer to 'opinion'

http://www.differencebetween.net/language/words-language/difference-between-feelings-and-sentiment/

An emotion is a complex reaction of the brain to a stimulus.

A sentiment is the combination of emotion and thought. It's the subjective experience of our emotions.

Emoticons? Are they useful? (See code)

Feeling good?

Generally, can be subdivided into positive and negative

Emoticons, how to represent them in R

Sentiment with the `tidytext` package (in revision!!!!)

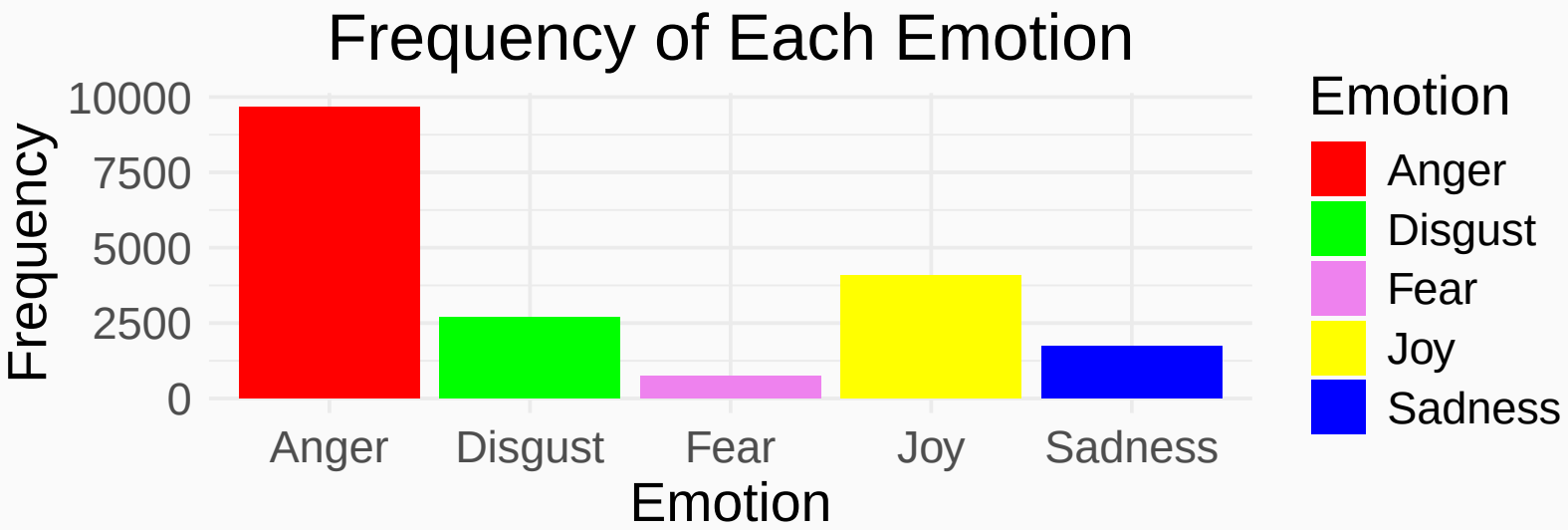
Lexicon: **AFINN**, **bing**, **nrc** (based on unigrams)

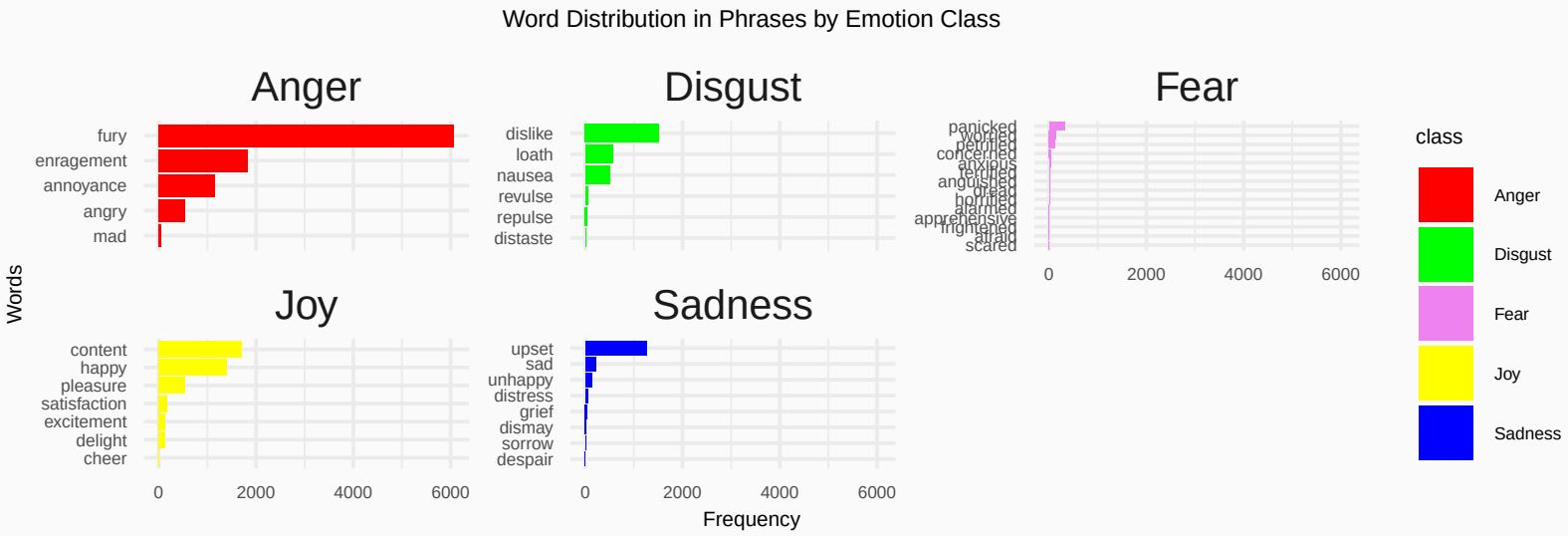
- AFINN (Finn Arup Nielsen) (scores that run between -5 and 5)
- bing (Bing Liu and collaborators) (binary fashion, positive or negative)
- nrc (Salf Mohammad and Peter Turney) (binary fashion in categories of positive, negative, anger, anticipation, disgust, fear, joy sadness, surprise and trust)

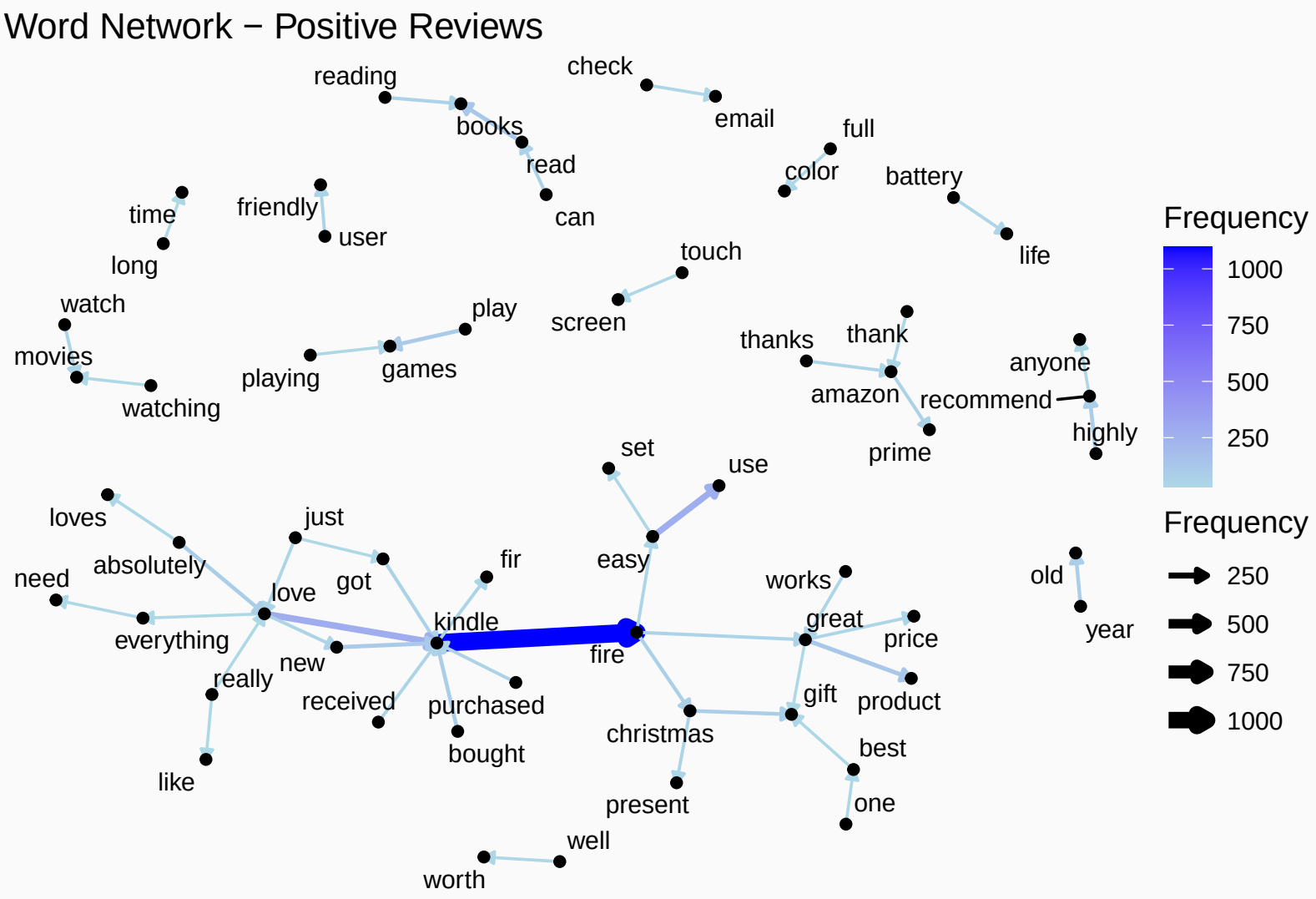
Sentiment with the **shyuzet** or **sentimentr** package.

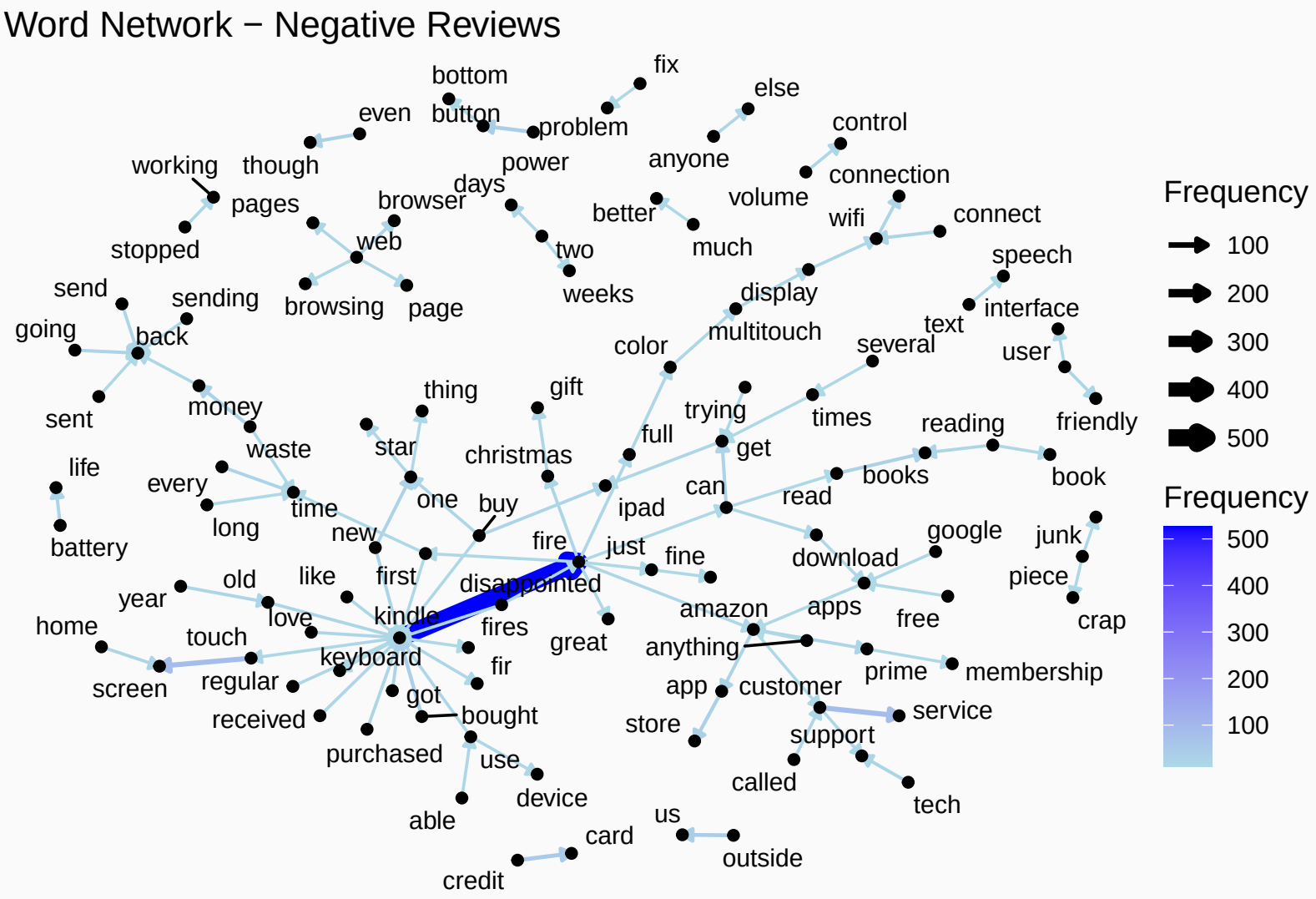
Link to article that uses Sentiment Analysis

Wheel of emotions


Frequency of Each Emotion
Frequency
10000
7500
5000
2500
0
Anger
Disgust
Fear
Joy
Sadness
Emotion
Emotion
Anger
Disgust
Fear
Joy
Sadness


Word Distribution in Phrases by Emotion Class
Anger
fury
enragement
annoyance
angry
mad
Disgust
dislike
loath
nausea
revulse
repulse
distaste
Fear
panicked
worried
petrified
concerned
anxious
terrified
anguished
dread
horrified
alarmed
apprehensive
frightened
afraid
scared
Joy
content
happy
pleasure
satisfaction
excitement
delight
cheer
Sadness
upset
sad
unhappy
distress
grief
dismay
sorrow
despair
Words
0
2000
4000
6000
Frequency
class
Anger
Disgust
Fear
Joy
Sadness


Word Network − Positive Reviews
reading
books
read
can
check
email
full
color
battery
life
time
long
friendly
user
touch
screen
play
games
playing
watch
movies
watching
thanks
thank
amazon
recommend
prime
anyone
highly
set
use
easy
loves
absolutely
need
everything
just
got
love
fir
kindle
fire
new
really
like
received
purchased
bought
christmas
present
works
great
price
product
gift
best
one
old
year
well
worth
Frequency
1000
750
500
250
Frequency
250
500
750
1000


Word Network − Negative Reviews
bottom
fix
else
even
button
problem
control
working
though
power
anyone
connection
days
volume
pages
browser
better
wifi
connect
stopped
web
two
much
speech
send
sending
browsing
page
weeks
display
interface
going
back
color
multitouch
text
several
user
thing
gift
trying
times
reading
friendly
sent
money
full
get
waste
star
christmas
books
book
life
every
can
read
time
one
buy
ipad
long
new
fire
just
google
junk
battery
fine
download
first
disappointed
piece
year
old
like
amazon
apps
free
crap
home
love
kindle
fires
great
anything
touch
keyboard
fir
prime
membership
screen
regular
app
customer
got
service
received
bought
store
support
purchased
use
called
device
tech
able
us
card
outside
credit
Frequency
100
200
300
400
500
Frequency
500
400
300
200
100

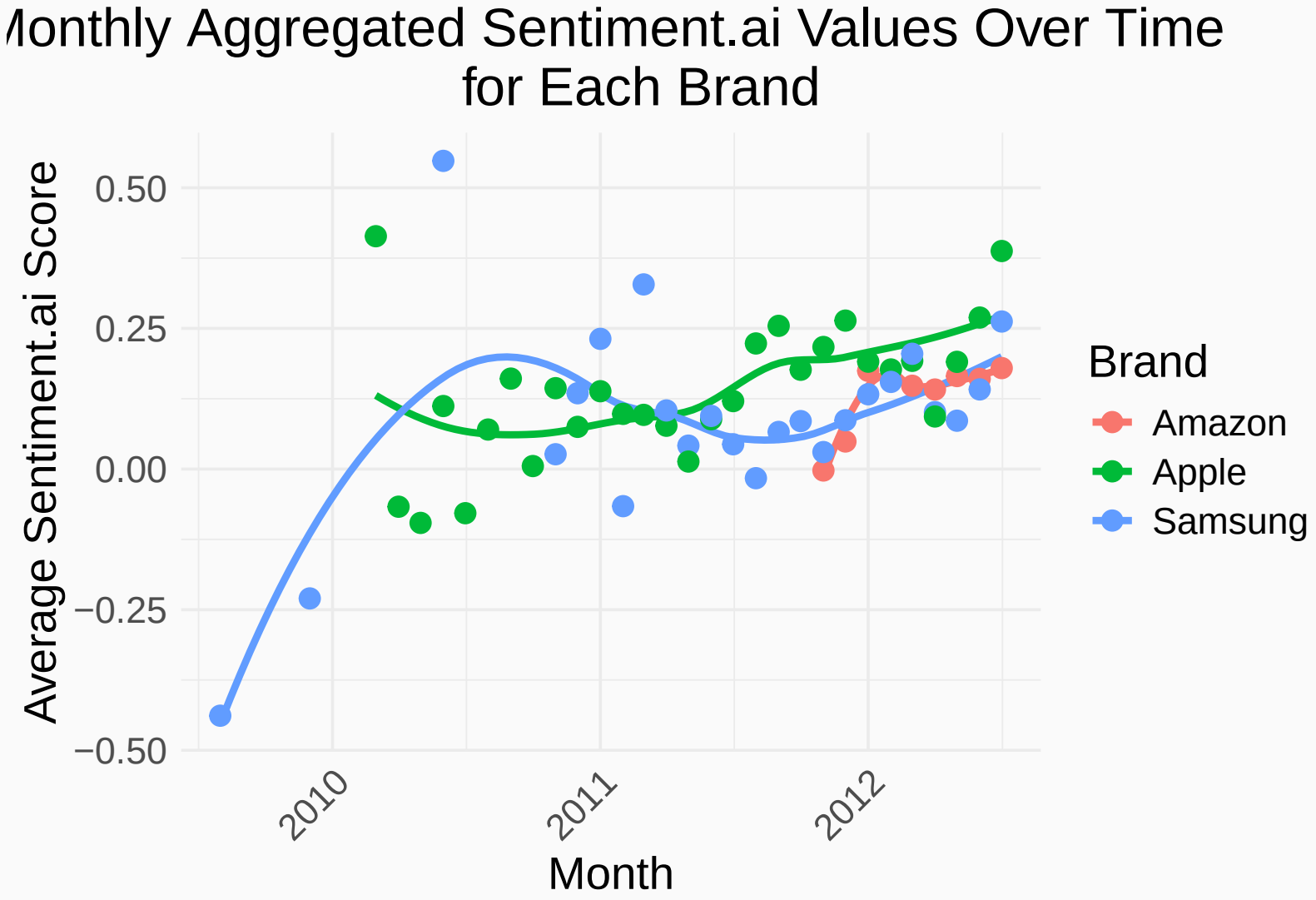
Sentiment Analysis

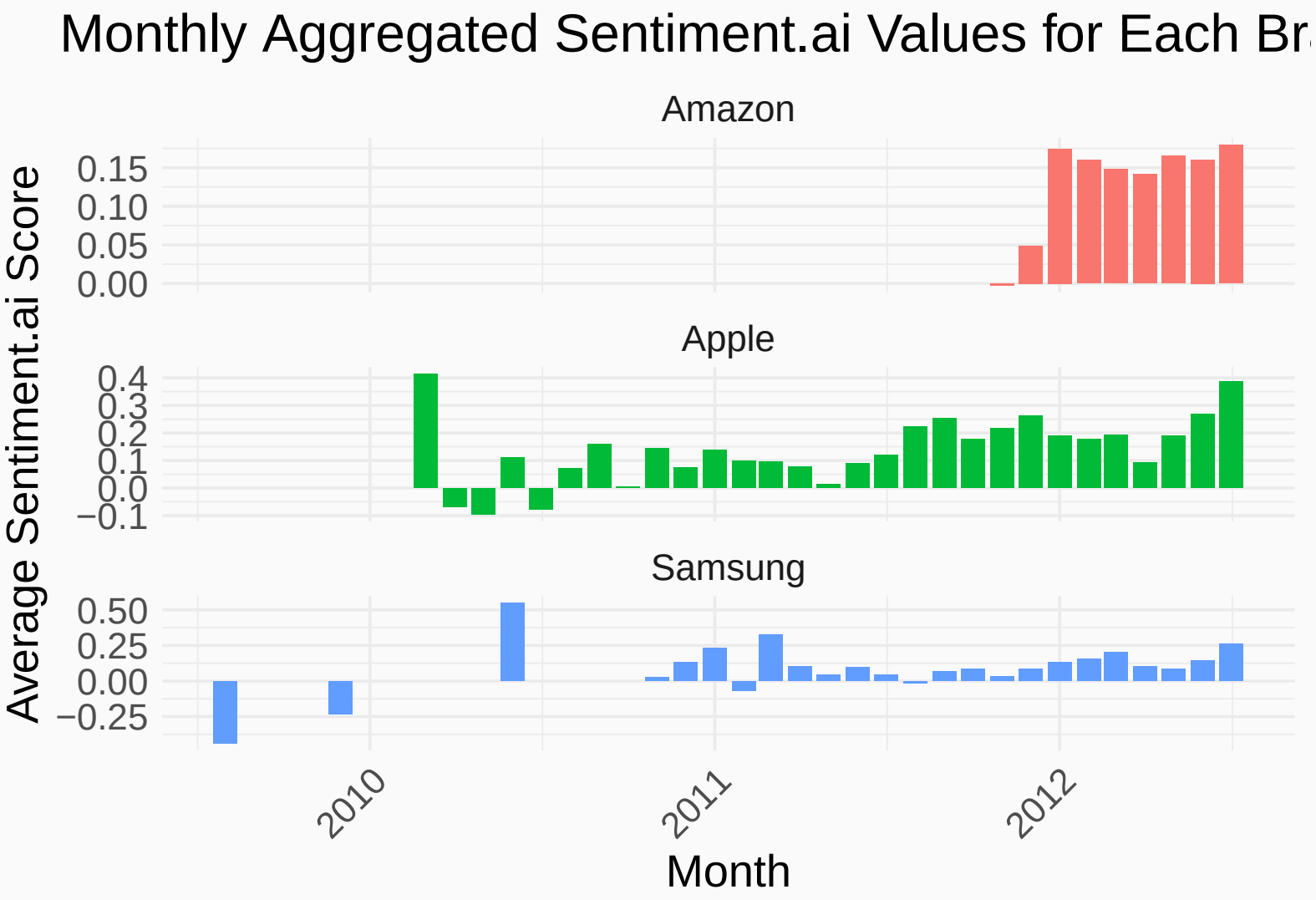
5574: ***-.009***

OK, I've had my iPad for almost a month. I continue to suffer repeated WiFi problems. It's not my router, it works fine with my iMac and my iPod. It's time for Apple to admit there is a problem and get a software fix for this thing. It becomes useless to you since you won't be able to maintain a connection with the internet. WAIT TO BUY THIS UNTIL APPLE ADMITS AND FIXES THE PROBLEM!!! Other than that, it is very nice, but it really needs to work.

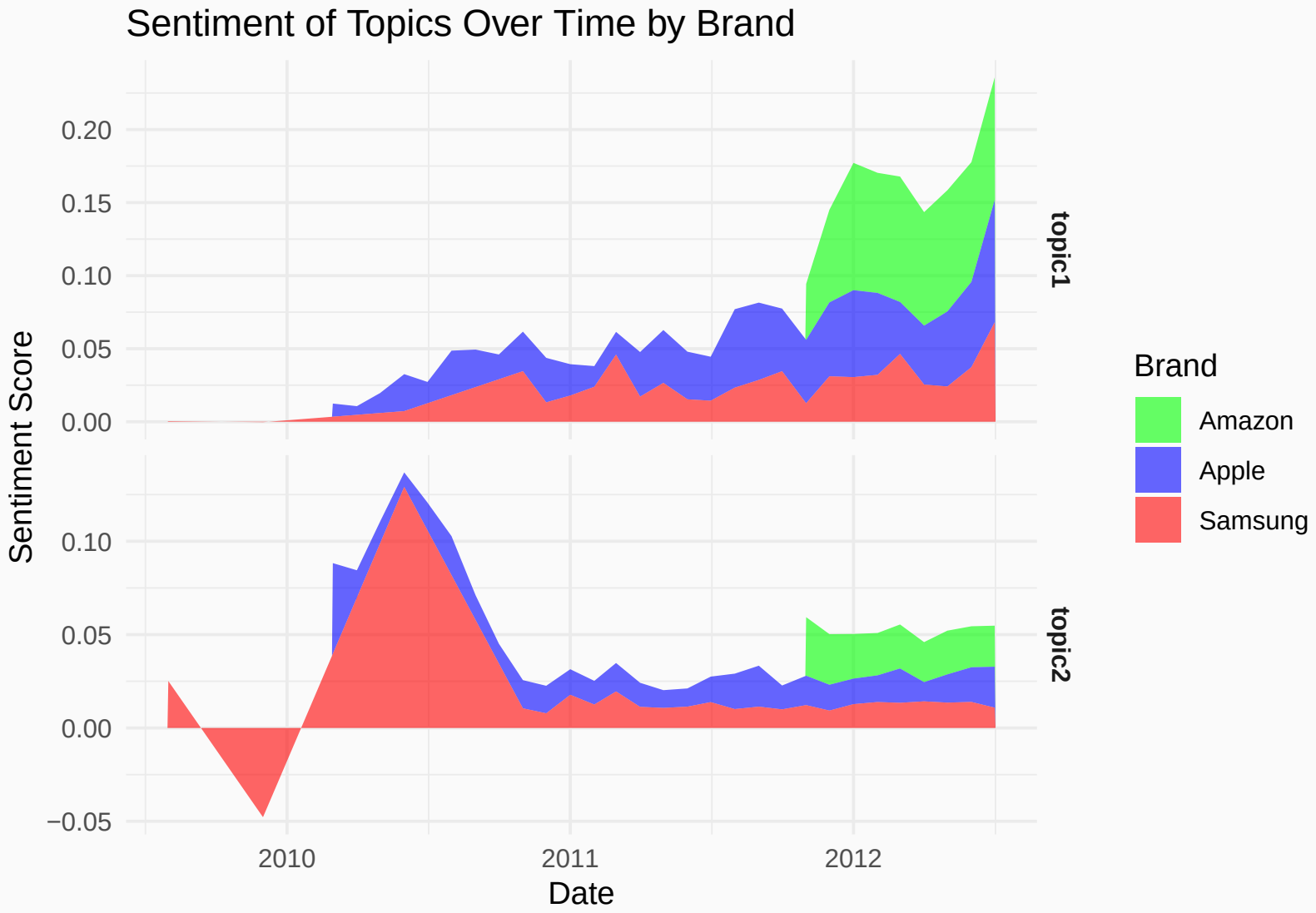
8367: ***-.003***

For the money, you'd think the screen resolution would be acceptable, but it's not. The text is fuzzy and unpleasant to read, and, as you interact visually with this device, the screen is by far its most important feature. Otherwise, the device is excellent: fast, responsive, light-weight, etc. Unfortunately, the screen ruins this product. It looks like a knock-off.


Monthly Aggregated Sentiment.ai Values Over Time
for Each Brand
Average Sentiment.ai Score
0.50
0.25
0.00
−0.25
−0.50
2010
2011
2012
Month
Brand
Amazon
Apple
Samsung


Monthly Aggregated Sentiment.ai Values for Each Br
Amazon
Apple
Samsung
Average Sentiment.ai Score
0.15
0.10
0.05
0.00
0.4
0.3
0.2
0.1
0.0
−0.1
0.50
0.25
0.00
−0.25
2010
2011
2012
Month

Sentiment Analysis



Topic Models, Classification, Named Entity Recognition

LDA (continued)

LDA, again

- For humans, topics are easily distinguishable, but not for machines
- Topic modeling solves this problem (for the machines)
- LDA is the most popular modeling technique
- Words are NOT immediately grouped into topics, a probability algorithm will compute their distribution grouping them around a topic with “a certain probability”.
- LDA is an **unsupervised** machine learning algorithm
- In a supervised mechanism one can debug the decision-making process
- In LDA you can tweak some parameters though
- Important: topics that have mixed words from different other topics do not have enough documents to work with
- There are several side-projects for LDA, such as “seeded words LDA”
- Still, sometimes the topics (the words that define them) to not make sense when they appear together

How does LDA work?

One topic modeling algorithm is LDA. It is a statistical model that finds patterns in text data by identifying the underlying (latent) topics that are present in a corpus (collection of documents).

1. LDA assumes each document is made up of a mixture of topics
2. Each topic is a collection of related words
3. In hotel reviews, **location** is a topic characterized by words such as: position, near, by, subway, across, park, far, etc.
4. LDA initiates by randomly assigning each word in each document to a topic
5. It iteratively updates these assignments based on two factors:

- the probability of each topic generating each word: what is the probability that this word belongs to topic z? This is done by counting the number of times the word appears in each topic, divided be the total number of words assigned to that topic.
- the probability of each document containing each topic: what is the probability that this topic belongs to document d? This is calculated by counting the number of words in the document that are assigned to each topic, and dividing by the total number of words in the document

6. After several iterations, LDA converges (meaning that it stops)

Example

The PROBABILITY of **across** to fit in topic location from {location, value, rooms, service, ... } when the word is present in a document, is given by the the number of times **across** is assigned to a topic {location, value, rooms, service, ... } times the number of other words in that specific document that already belong to location.

$$P(Z|W, D) = \frac{\text{count of W in topic Z} * \text{count of words in D that belong to Z}}{\text{total count of words in Z}}$$

Translation: for each document D, and for each word W in document D, calculate the probability of a specific word (such as across) to belong to each topic Z.

Example. Let us say that initially, the word **across**, in document 1, was assigned to topic rooms and the probability to be there is 0.003242115. Similarly, that word will belong to other topics in other documents. Now, in the whole corpus, the word **across** was initially assigned 5 times to topic location, 3 times to topic rooms, etc. Also initially, the number of words in document 1 that belong to topic location is 2 and the total count of words in location is 1836.

$$P(\text{location}|\text{across, document 1}) = \frac{5 * 2}{1836} = 0.005446623$$

Compare this probability, to the probability that the word **across** was assigned to one of the topics present and choose the highest value, e.g. P(location|across, document 1) > P(room|across, document 1). Whichever topic the probability was maximum for W.

$$W_Z = max(P(Z|W, D))$$

Until convergence.

How to obtain “qualitatively” superior topics

- The quality and quantity of the corpus and its pre-processing
- The variety of topics in the corpus
- The number of grams
- The number of topics chosen
- The fine-tuning parameters.

LDA with seeded words and/or bigrams

(see code)

- What changes if we put in some seed words?
- Can we bias the words towards desired topics through seeded words?
- What changes if we used bigrams? Trigrams?
- Can we have better topic representations?

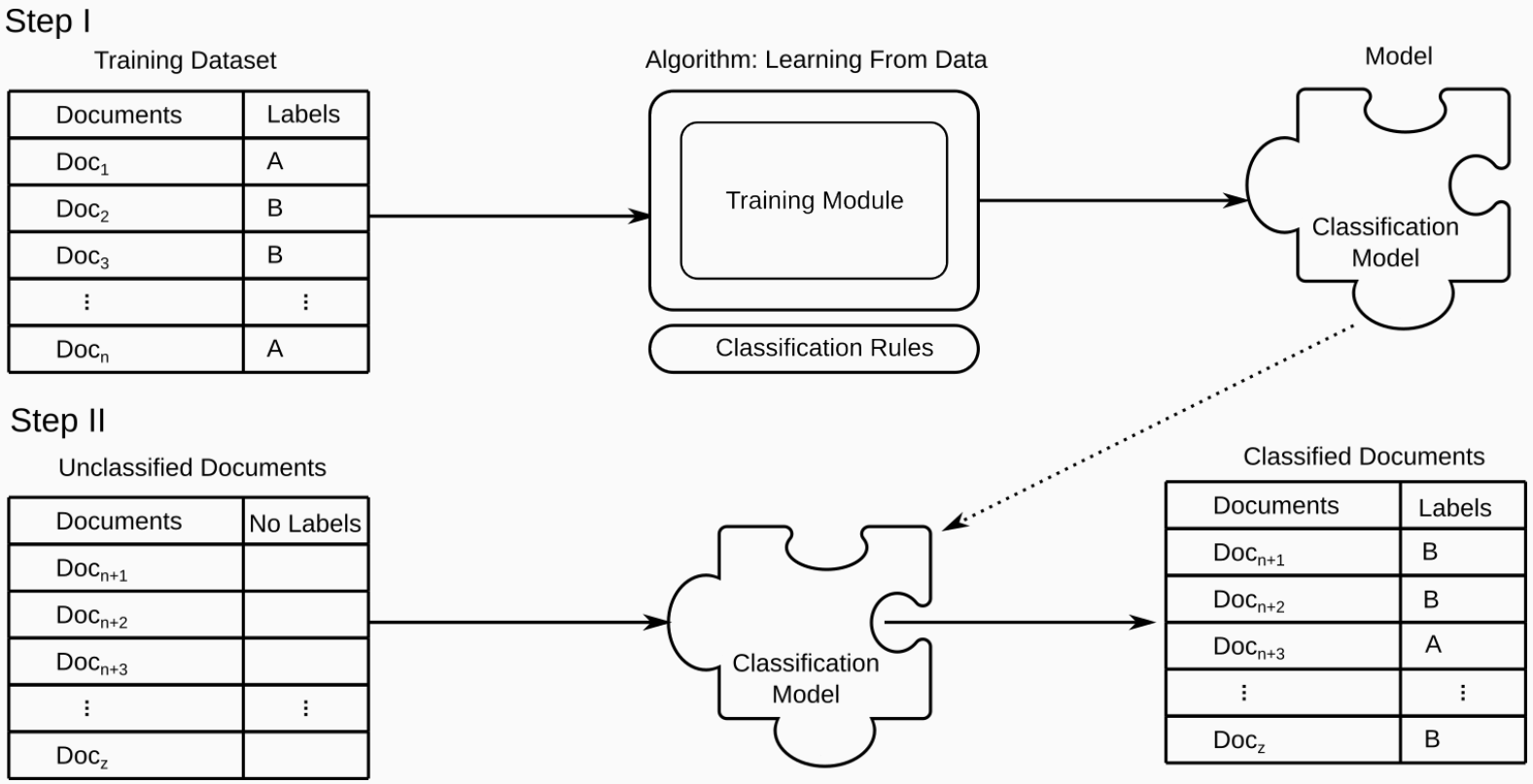
Text classification

To classify means to decide and assign some “things”, according to one or many features of those “things”, into pre-existing classes. The word class comes from Latin and, in origin, the classes were six orders the Roman people were subdivided in for taxation (although at its very origin, classes were a military term). Depending on their income, people were assigned to their corresponding classes. Any “Roman”, had to fit into one of the predetermined classes and pay taxes accordingly to their wealth status. Some two thousand years fast forward, there are several ways to accomplish the task of classification. One of them is manual which we avoid due to obvious reasons: huge data volumes cannot be processed in a reasonable amount of time by humans.

In this section we will focus on some Supervised Machine Learning algorithms to conduct the operation of classification. We have basically two elements here: the text which is the content of the review, and its association with a specific brand. We will use these two elements that we know well so that the algorithm can create a model. Based on this model, we can have in input new content that is not brand labeled and that should be correctly classified. This means that the correct brand label will be assigned to it.

Text classification models

The whole process can be summarized in the figure below:



Classification Models

This is a typical machine learning procedure; in this instance, we're using it to process an corpus of documents, but the process would be the same for datas ets of various kinds. We feed a group of classified documents in the first step. Keep in mind that a doc is essentially an review. We add a brand label to each review. The already classified documents data are called the training set because they will give the algorithm a complete scheme on how reviews are associated (or classified).

The machine learns from this association and through some classification rules, and it builds a model. The words in each review, or the data received as insight, serve as the model's foundation. We will feed a group of unlabeled(or unclassified) documents into the first step's model. Based on the model developed, we expect a classification of the unmarked documents. Overall, this is a process for prediction: the machine will determine which brand will be assigned to what reviews based on the model developed.

Will the algorithm always perfectly perform? No, at least not at this stage of technological development. But it will work well enough to save you a lot of time. If data are misclassified, then there is the need of human intervention and manual labeling. A sound procedure would involve retraining the dataset regularly, after the misclassification was corrected. This way you would renew the model with new words and new scenarios and the precision of results will increase.

There are many classification algorithms, some perform better then other, depending on your (data) needs and the situation. We can divide classification in two main types: linear and non linear. The first can be seen schematically as a line that separates data in classes. This is a simple but powerful implementation while the non linear counterpart is more cumbersome.

Procedure

How to separate into training and testing?

```r
dtm <- bind_cols(dtm, "Brand" = dt3$Brand, "Type" = dt3$Type)

# split the dtm into the "train" and "test" subsets and remove the Type column
# from the subsets as it became redundant
dtm_train <- dtm %>%
    filter(Type == "train") %>%
    select(-Type)

dtm_test <- dtm %>%
    filter(Type == "test") %>%
    select(-Type)
```

```
         test train
  Apple   372  1520
  Samsung 365  1419
[1] 3676  804
```

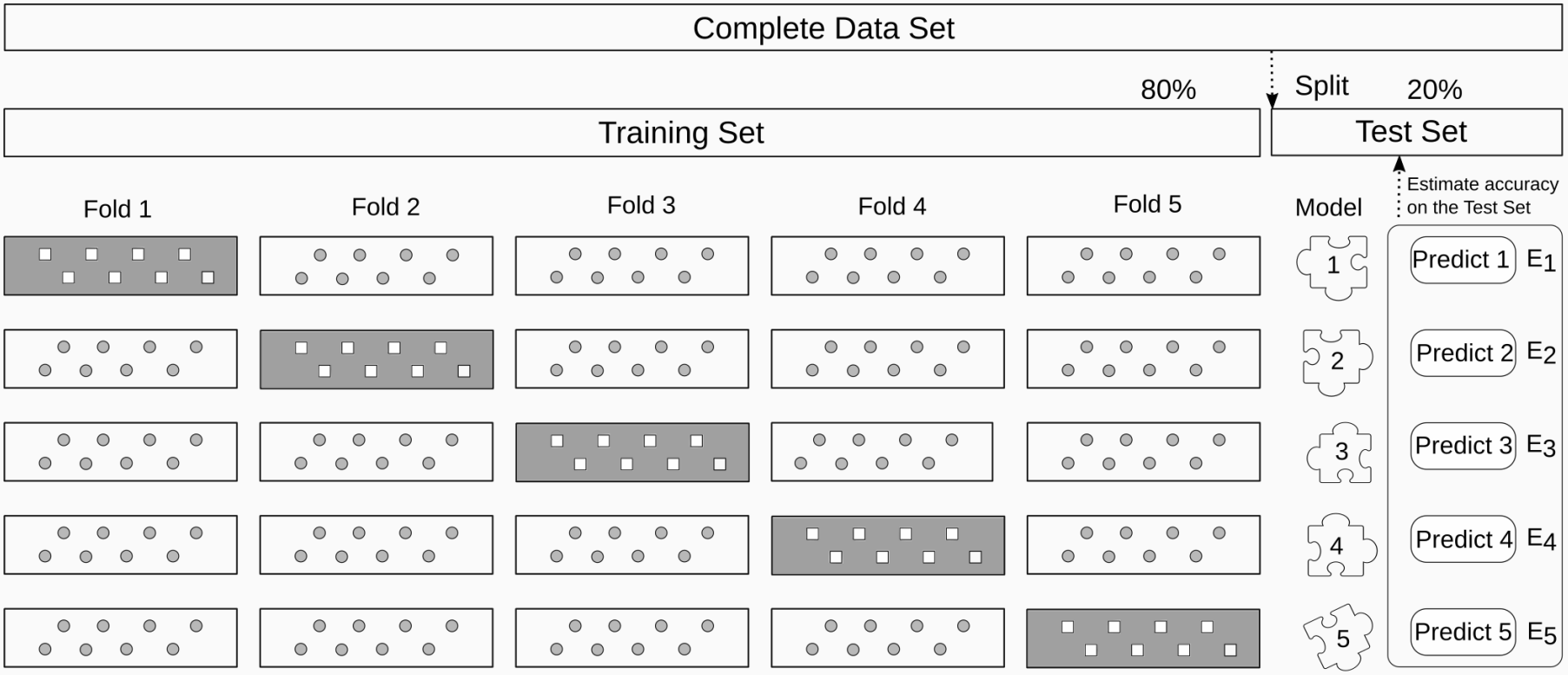
Cross validation

We need a further step to create a classification model. Something that is generic and can be applied to several classification algorithms. It is called cross validation.

Cross validation is an internal procedure that fits into the first step of the previous figure in which the training set is separated again into a training and testing (or validating) part.

The training data set is sliced into K parts, also called folds. As you have already guessed, K is a number and it usually spans between 5 and 10 but you can have as many values as the data set has. You will see, in the `trainControl()` function you can achieve that with the help of the attribute **number**. For example, if $K = 5$, 4 parts will be used for training and 1 for validating.

Cross validation



$$E = \frac{1}{5}\sum_{i=1}^{5}\frac{1}{E_i}$$

Latent Discriminant Analysis (LDA)

A widely used algorithm that classifies is logistic regression. It can classify objects into two classes which is fine if you need just that but if you want more than two classes, then we need something more powerful such as Multinomial Logistic Regression or Linear Discriminant Analysis (LDA). The algorithm uses a linear function (called discriminant) to "discriminate" between more than two classes and make separation. It overcomes the Logistic Regression problems such as instability when the classes are well separated or when the data set is limited. Instability means we have poor results obtaining a model that has coefficients that do not converge well.

```
# set the validation scheme, 10-fold cross-validation
control <- trainControl(method = "cv", number = 10, savePredictions = "final",
                        classProbs = TRUE)

# train the dataset with the lda method on the column Brand, using the dtm_train dataset
set.seed(seed)
lda <- train(Brand ~ . , data = dtm_train, method = "lda", trControl = control,
             metric= "Accuracy", tuneLength = 5)

# check the model
lda
```

```
Linear Discriminant Analysis

2939 samples
 804 predictor
   2 classes: 'Apple', 'Samsung'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 2645, 2646, 2645, 2645, 2645, 2645, ...
Resampling results:

  Accuracy   Kappa
  0.7727195  0.5429092
```

The results we obtain from the training are summarized by simply typing the name of the variable in which we saved the procedure. We can see the number of classes, number of samples, predictor variables (basically the words from the dtm after preprocessing), the cross-validation type, the Accuracy and Kappa evaluation metrics. Accuracy is more useful when you have a two classes, while Kappa (or Cohen's Kappa), is better when you have imbalance in classes.

In order to predict the test dataset we use the function **predict()**. The system can also give us more information regarding the probabilities with which it assigns a label.

```
# based on the lda classification model, classify the dtm_test dataset
lda.predict <- predict(lda, newdata = dtm_test)
```

A confusion matrix is created by putting together the values of the classification through the model with the ground-truth values. This is assessed with the help of the confrontation with the data in the Test Set which is set to 20%. There are several ways to measure how well a model performs, common measurement types are Accuracy, Sensitivity and Specificity.

The **caret** package focuses on Accuracy which basically calculates how many correct answers we obtain from the total number of answers.

LDA confusion

```r
# check the performance on the test set
cfm <- confusionMatrix(lda.predict, as.factor(dtm_test$Brand))
print(cfm)

# adding the "prob" attribute will give the classification probabilities for
# each brand
lda.predict.p <- predict(lda, newdata = dtm_test, type = "prob")

# display the first six documents, function round will round up to two decimals,
# hence the 2 while the multiplication by 100 yields the probability expressed
# in percentage for a better reading
head(round(lda.predict.p*100,2))
```

	Apple <dbl>	Samsung <dbl>
4	90.13	9.87
5	92.70	7.30
8	99.99	0.01
11	67.03	32.97
16	93.08	6.92
20	98.90	1.10

6 rows

Classification measurement types

Checking the probability with which the prediction based on the model assigned the text to what brand gives us further information on how “sure” the algorithm is of making that decision. This is another tool in understanding that misclassification can be due to ambiguities and commonalities in the terms used to define a specific brand.

Be aware that the model fit can be evaluated through some other overall measures such as Accuracy and Error Rate and some Class-Specific Measures such as Precision, Recall and F-Measures. The accuracy measures the proportion of correct predictions over the total number of predictions. The error rate is the proportion of incorrect predictions over the total number of predictions. It is basically the inverse of accuracy and can be calculated like this: Error rate = 1 - Accuracy. Here we can introduce another measure, called Cohen’s kappa which evaluates the agreement between the expected accuracy and the calculated accuracy of a model. When we deal with class-specific measures we focus on specific classes and how well they are predicted. Precision is similar to accuracy, it is the proportion of correctly predicted over the total number of predicted items but for a given class. Recall calculates the proportion between the number of correctly predicted and the total number of items that are in that specific class. F-measure combines precision and recall trying to balance them all. The formula is $F_i = 2 \times \frac{precision \times recall}{precision + recall}$ where i is the class. To have a better understanding we need the confusion matrix.

Confusion matrix

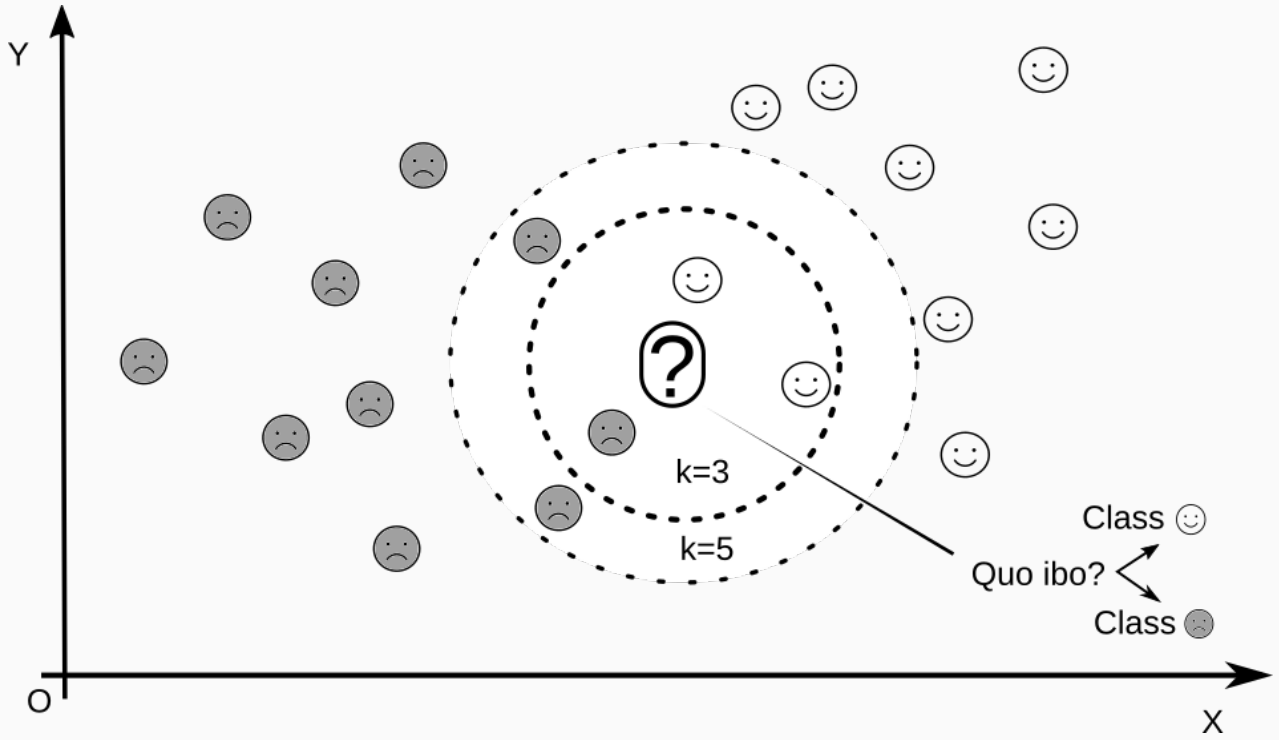
	Actual Apple	Actual Samsung	Total Predicted
Predicted Apple	325	108	372
Predicted Samsung	47	257	365
Total Actual	433	304	737

With the numbers we obtained, let us calculate the Accuracy: $\frac{325+257}{737} = 0.7897$. The error rate is $1 - 0.7897 = 0.2103$. Precision for the Apple tablets is $\frac{325}{372} = 0,8737$ and Precision for the Samsung tablets is $\frac{257}{365} = 0.7041$. Recall for the Apple products is $\frac{325}{433} = 0.7506$ while for the Samsung products is $\frac{108}{304} = 0.3553$. Calculating the F-measure is straightforward by applying the formula given in the previous paragraph.

K-NN or K Nearest Neighbors

A non linear algorithm is K-Nearest Neighbor, also known as KNN. This algorithm is used for classification and regression. While the first one tries to predict to what class an entity should be assigned to, the second one tries to predict a number. We will be focusing on the latter. KNN is a peculiar case of algorithm. The model, which is the thing we look for in order to make predictions, is the complete data set. So, there is no learning required because the predictions use the training data set by looking up the most similar K elements in the set. Neighboring is computed as a distance so this is a distance-based algorithm. It is computed in a similar way to what you have seen in Chapter 5 in Hierarchical Clustering. Depending on your data type, there are several ways to calculate the distance, such as Hamming, Manhattan, Jaccard, Mahalanobis or cosine. The best value for K can be found by the algorithm by trying several values. KNN classifies an element by choosing the k-most similar elements. The algorithm itself performs well when it deals with a limited number of predictors but the performance drops if their number increases. For further information on KNN.

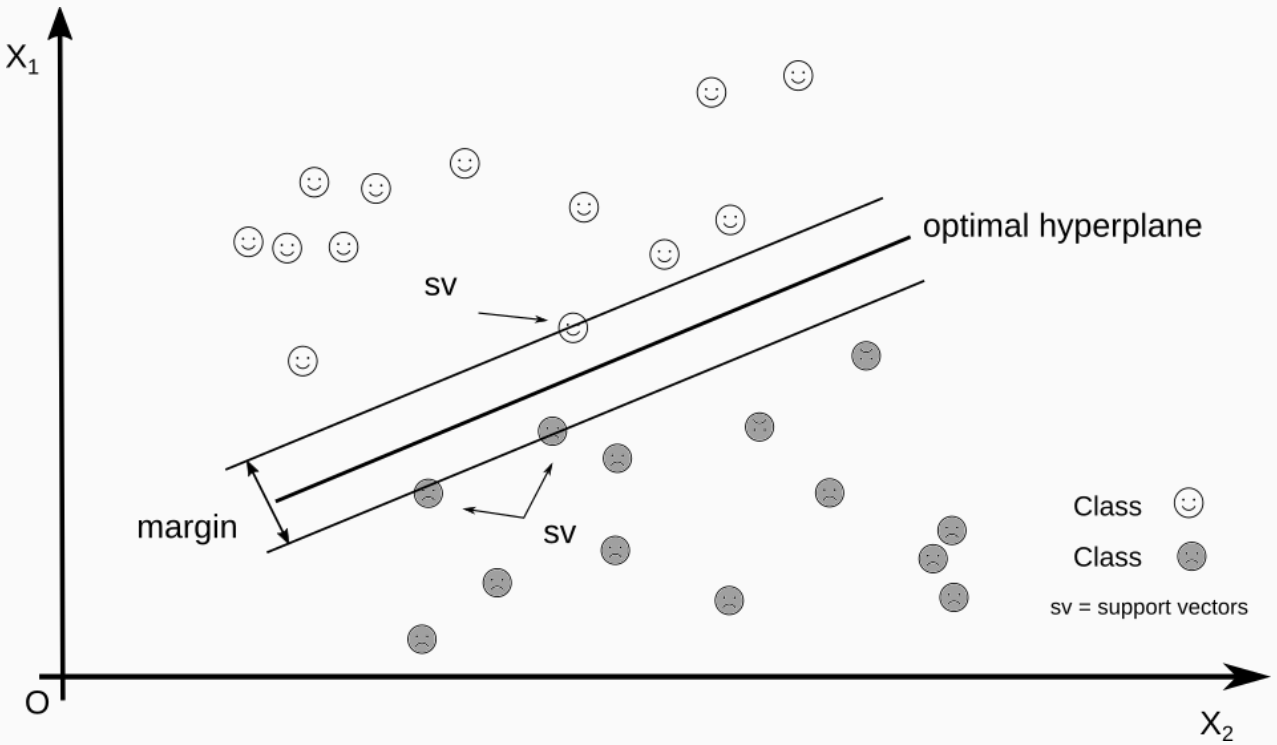
K-NN or K Nearest Neighbors



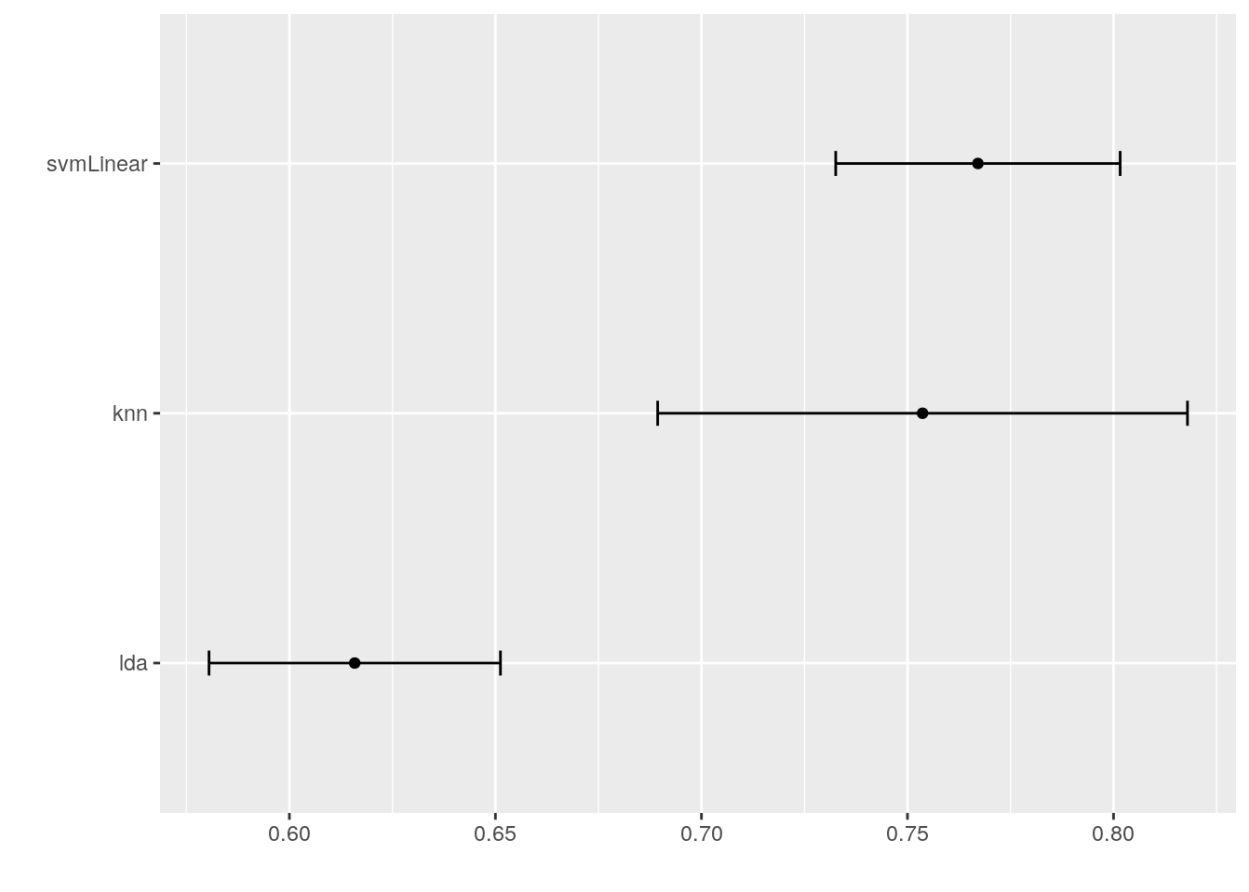
K-NN or K Nearest Neighbors

k	Model	Word Root	Accuracy
7	BOW	Stem	0.7842
5	BOW	Lem	0.7914
9	Tf-Idf	Stem	0.6189
5	Tf-Idf	Lem	0.6250

Support Vector Machine

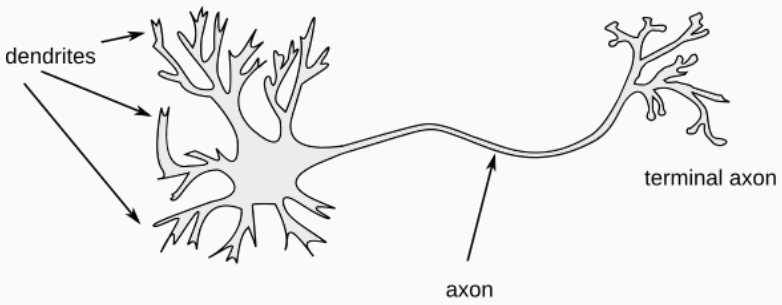


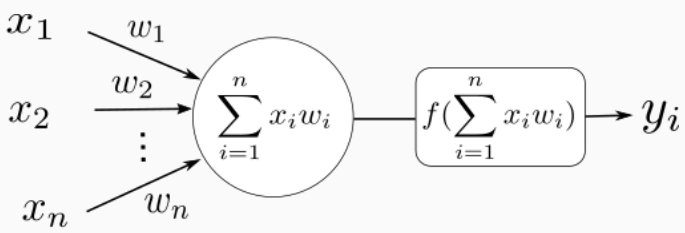
Compare Performances

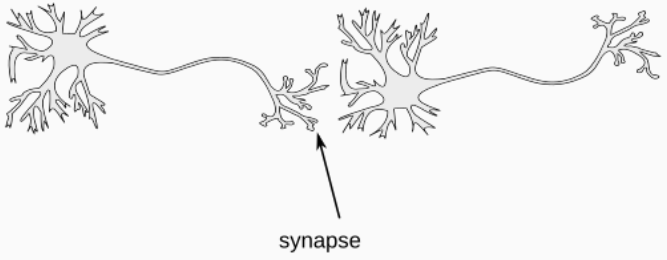


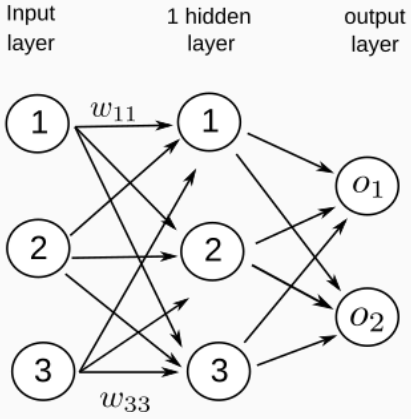
The fascination of how our brain thinks and what can be done in computer terms in order to mimic and outperform our reasoning found a lot of support in the academic media. The transition from a human neural cell of a Natural Neural Network into an algorithmic program consisting of one or more functions of an Artificial Neural Network (ANN) was a fantastic evolution. ANNS are a specific subfamily of machine learning algorithms that uses linked nodes situated on various layers to process data, trying to copy the behavior of the human brain with its neurons and synapses. These algorithms can be very powerful but they are not always the best choice in terms of performance. This can depend of several situations, such as the complexity of the problem to solve, the dataset dimension, the computing power, etc. All in all, ANNs accomplishing tasks such as image recognition or image classification, text understanding or text generation, fraud detection, medical diagnosis and many other.

Neurons and Neural Networks









Neural Networks Performance

```
          Reference
Prediction Apple Samsung
   Apple     336      42
   Samsung    36     323

               Accuracy : 0.8942
                 95% CI : (0.8697, 0.9154)
    No Information Rate : 0.5047
    P-Value [Acc > NIR] : <2e-16

                  Kappa : 0.7883

 Mcnemar's Test P-Value : 0.5713

            Sensitivity : 0.9032
            Specificity : 0.8849
         Pos Pred Value : 0.8889
         Neg Pred Value : 0.8997
             Prevalence : 0.5047
         Detection Rate : 0.4559
   Detection Prevalence : 0.5129
      Balanced Accuracy : 0.8941

       'Positive' Class : Apple

         NNPredictions
          Apple Samsung
  Apple     336      36
  Samsung    42     323
```

Ensemble Classification

One can build three types of ensemble algorithms and they have a name: Bagging, Boosting and Stacking. Bagging would put together several models, each of them taking different subsamples of the training data set. Boosting, acts like a chain, each model tries to improve on the previous one. The first two methods use the same type of models while Stacking can put together different kinds plus a model on top of that. The supervisor model works like an orchestra director who would blend the prediction of the initial models to obtain the best result. All the ensembles have pros and cons, you choose the best of them depending on the situation.

Here we will focus on Stacking. Using the **caret** package one would put together several models under a model that would stay on top and combine their predictions to yield optimal results. Similarly to what was done previously, we set up the control attributes for the training. Then we add to a list a series of Machine Learning algorithms that we did before, i.e. LDA, KNN and SVM in its linear version. Through the **caretList()** we train the models in the list. Then we check the prediction accuracy of them all.

Ensemble Classification

We notice that in this case, the ensemble classification is no better than ANN result, but it is not always like that.

```
A glm ensemble of 3 base models: lda, knn, svmLinear

Ensemble results:
Generalized Linear Model

5878 samples
   3 predictor
   2 classes: 'Apple', 'Samsung'

No pre-processing
Resampling: Cross-Validated (10 fold)
Summary of sample sizes: 5290, 5290, 5290, 5290, 5290, 5290, ...
Resampling results:

  Accuracy   Kappa
  0.8075867  0.6133461
```

Named Entity Recognition (NER)

Named Entity Recognition (NER) is a natural language processing (NLP) technique used to identify and extract named entities from text. Named entities refer to words or phrases that represent specific entities or concepts, such as names of people, places, organizations, and products.

The process of NER involves analyzing the text to identify and classify named entities into predefined categories such as person, location, organization, date, time, etc. This can be done using machine learning algorithms that are trained on large annotated data sets or rule-based approaches that use a set of rules to identify named entities.

The output of NER is a structured representation of the text that includes the identified named entities along with their corresponding categories. NER is used for information retrieval, text classification, sentiment analysis, and machine translation. It is also used in many real-world applications such as news analysis, social media monitoring, and customer relationship management (CRM).

NER

```r
library(nametagger)
model <- nametagger_download_model("english-conll-140408", model_dir = tempdir())
x <- data.frame(doc_id      = c(1, 1, 2),
                sentence_id = c(1, 2, 1),
                text        = c("I\nlive\nin\nNew\nYork\nand\nI\nwork\nfor\nApple\nInc.",
                                "Why\ndon't\nyou\ncome\nvisit\nme",
                                 "Good\nnews\nfrom\nAmazon\nas\nJohn\nworks\nthere\n."))
head(predict(model, x))
```

```
##   doc_id sentence_id term_id term entity
## 1      1           1       1    I      O
## 2      1           1       2 live      O
## 3      1           1       3   in      O
## 4      1           1       4  New  B-LOC
## 5      1           1       5 York  I-LOC
## 6      1           1       6  and      O
```

Text embeddings

Text embeddings, the very basics

To understand any modern generative NLP task performs, one needs to understand the most important concept: word embeddings. It involves transforming textual data into numerical vectors, which are sequences of numbers that encapsulate the information within words.

Such example can be the word love:

$$\text{love} = \left[0.23, -0.67, 0.15, 0.91, -0.34\right]$$

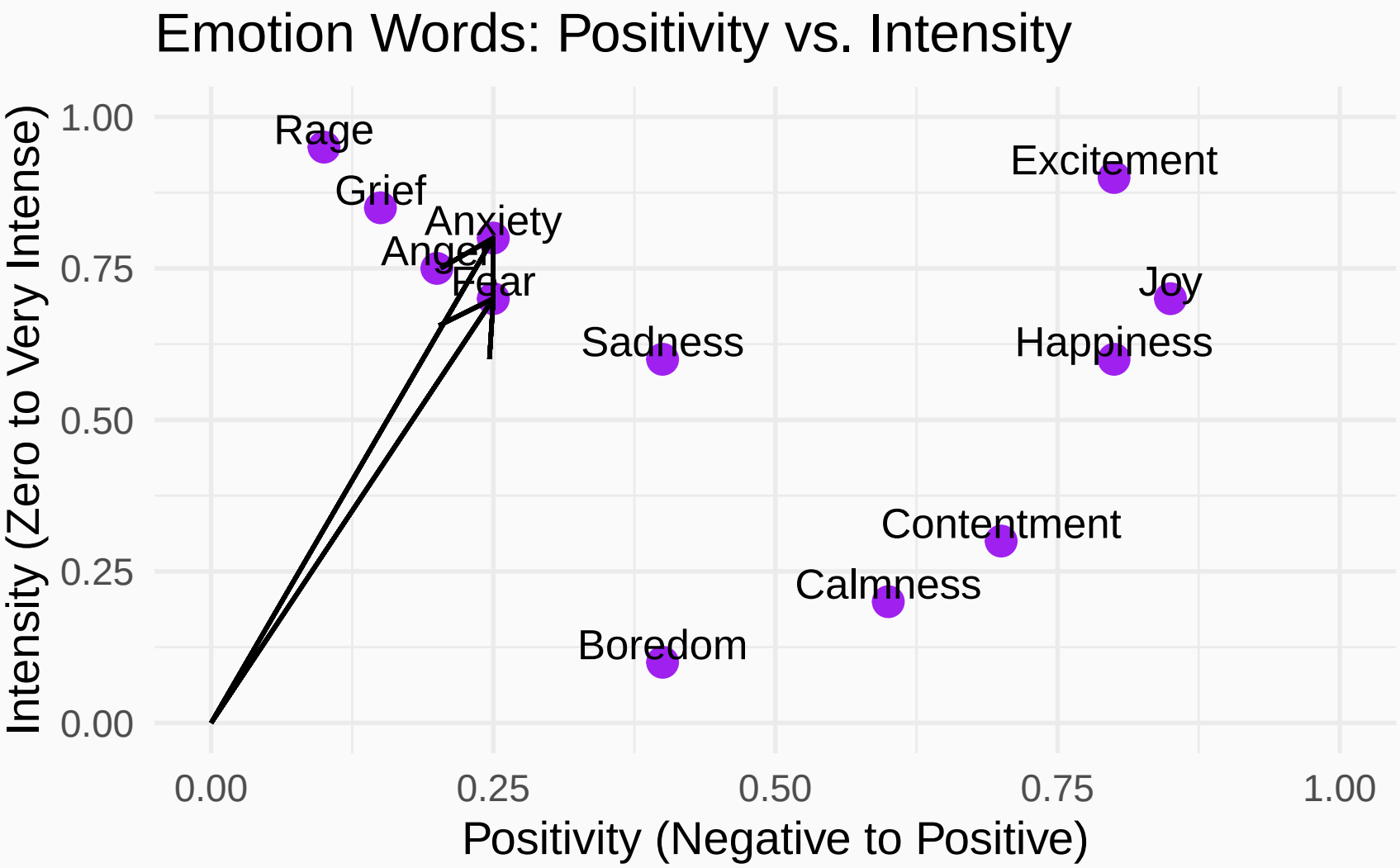
In real-world applications, word vectors are typically generated by algorithms like Word2Vec, GloVe, or FastText, and they might have several dimensions, ranging from 50 to 300. We will see how to implement Word2Vec and understand what these dimensions are. But first...

A Vector Space (Emotional)

We can use the words that describe emotions and encode them in a very empirical way, into numbers. This encoding is our subjective secret: negative is 0, and positive is 1, with a range of values in between. Similarly, for intensity of the emotions, zero is 0, and very intense is 1. The emotion "Joy" can be represented by two subjective numbers: [0.9, 0.7]. We can plot these numbers on a 2D plane as a point. On the x-axis, positivity = 0.9, and on the y-axis, intensity = 0.7. In a 2D space, and indeed in 3D, 4D, or any n-dimensional space, points correspond to vectors that start at the origin. While points and vectors are not the same, points can represent vectors. The coordinates [0.9, 0.7], which mark the endpoint of a vector, actually describe a vector. All these vectors form a space, in our case, an emotional vector space.

Example

Word	Positivity	Intensity
Joy	0.85	0.70
Happiness	0.80	0.60
Contentment	0.70	0.30
Excitement	0.80	0.90
Anger	0.20	0.75
Rage	0.10	0.95
Sadness	0.40	0.60
Grief	0.15	0.85
Fear	0.25	0.70
Anxiety	0.25	0.80
Calmness	0.60	0.20
Boredom	0.40	0.10


Emotion Words: Positivity vs. Intensity
Intensity (Zero to Very Intense)
1.00
0.75
0.50
0.25
0.00
Rage
Grief
Anxiety
Anger
Fear
Sadness
Boredom
Excitement
Joy
Happiness
Contentment
Calmness
0.00
0.25
0.50
0.75
1.00
Positivity (Negative to Positive)

Similarity

In the emotional space, we can represent emotions not only by points but also by vectors, as demonstrated with Anxiety and Fear. In real life, these emotions are often perceived as very similar, and this is reflected mathematically by their closeness on the 2D representation. This closeness is indicative of their similarity.

You might have noticed that we need to use vectors and not just points. Vectors do not provide only a magnitude (distance from the origin), but also direction, so they can capture the orientation and, as such, they are informative when it comes to understanding relationships between data points. As it is hard to apply mathematics to words, it is a lot easier to apply it to the number transformations of these words. After the computations, we retransform the numbers into words. Similarity in space is equivalent to similarity in meaning. But this similirarity is assessed through the cosine similarity measure. It two vectors overlap, the angle between them is zero.

More about similarity

Cosine similarity is a measure of similarity between two vectors, based on the angle between them. When the angle between the vectors is zero, the cosine similarity is 1, indicating that the vectors are identical. If the vectors are perpendicular, the angle is 90 degrees, resulting in a cosine similarity of 0, indicating no similarity. If the vectors are opposite, the cosine similarity is -1, and it means that they are dissimilar or opposite.

To illustrate this with an example, let us consider two words, "Fear" and "Anxiety," with their vector representations: Fear [0.25, 0.7] and Anxiety [0.25, 0.8]. Suppose the angle between these two vectors is denoted by θ. The cosine similarity between the vectors $\mathbf{F}$ and $\mathbf{A}$ is given by the formula:

$$\cos(\theta) = \frac{\mathbf{F} \cdot \mathbf{A}}{\|\mathbf{F}\| \times \|\mathbf{A}\|}$$

The numerator represents the dot product of the vectors: $\mathbf{F} \cdot \mathbf{A} = (0.25 \times 0.25) + (0.7 \times 0.8)$, while the denominator is the product of the magnitudes of the vectors: $\|\mathbf{F}\| \times \|\mathbf{A}\| = \sqrt{(0.25)^2 + (0.7)^2} \times \sqrt{(0.25)^2 + (0.8)^2}$.

Plugging these values into the formula, we obtain a cosine similarity of approximately 0.999195.

Many features, not only two

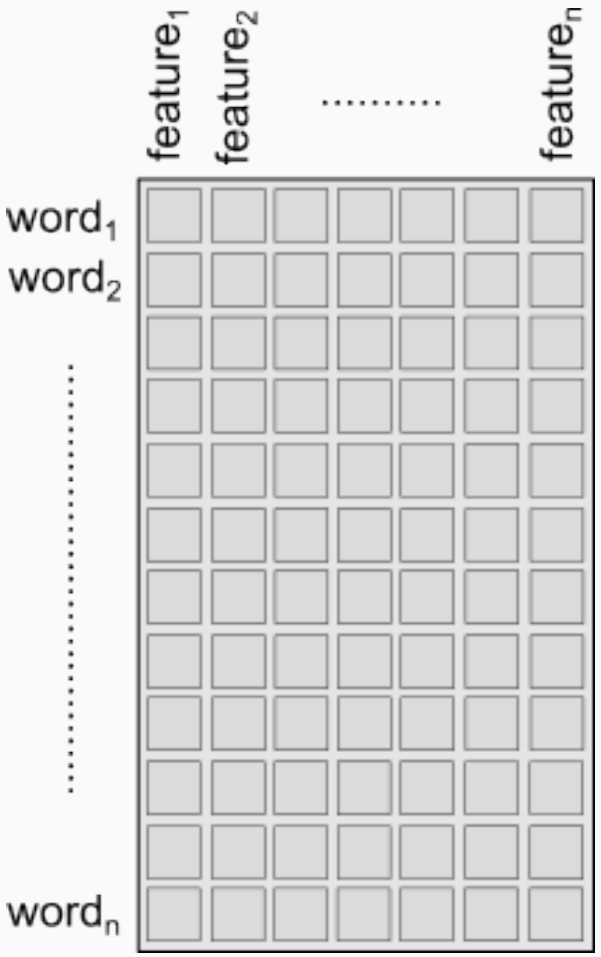
Emotion Characteristics Table

Word	Positivity	Intensity	Arousal	Valence	Dominance
Joy	0.85	0.70	0.80	0.90	0.70
Happiness	0.80	0.60	0.70	0.85	0.70
Contentment	0.70	0.30	0.40	0.75	0.50
Excitement	0.80	0.90	0.90	0.85	0.80
Anger	0.20	0.75	0.80	0.15	0.40
Rage	0.10	0.95	0.95	0.10	0.30
Sadness	0.40	0.60	0.50	0.35	0.40
Grief	0.15	0.85	0.70	0.20	0.30
Fear	0.25	0.70	0.90	0.30	0.40
Anxiety	0.25	0.80	0.90	0.30	0.40
Calmness	0.60	0.20	0.30	0.65	0.50
Boredom	0.40	0.10	0.20	0.50	0.35

Analogies

If you take the numbers in the corresponding positions, they are very similar. Similar in numbers, similar in significance. Because we have vectors nobody stops us from calculating analogies, following the classical example, king-man+woman=queen, we check: if Anxiety stands to Fear, Joy stands to Happiness. If you do the computations, you obtain a vector that is not exactly Happiness, but very close to it. You can see now that word embeddings are excellent for identifying conceptual analogies.

Word Embeddings, Organization



Word Embeddings

Vertically, we have the number of words: 10.000, 50.000, etc. Horizontally, we have the number of features that represent the number of neurons in the hidden layer.

Word Embeddings Creation

You shall know a word by the company it keeps. J.R. Firth. Distributional theory.

To generate word2vec, there are two main procedures, Continuous Bag of Words (CBOW) and Skip-gram. Both models try to predict context words surrounding a target word. We will make an example and then demystify the algorithmic part.

Another task, not only the analogies, could be the detection of the next word in a sequence of words.

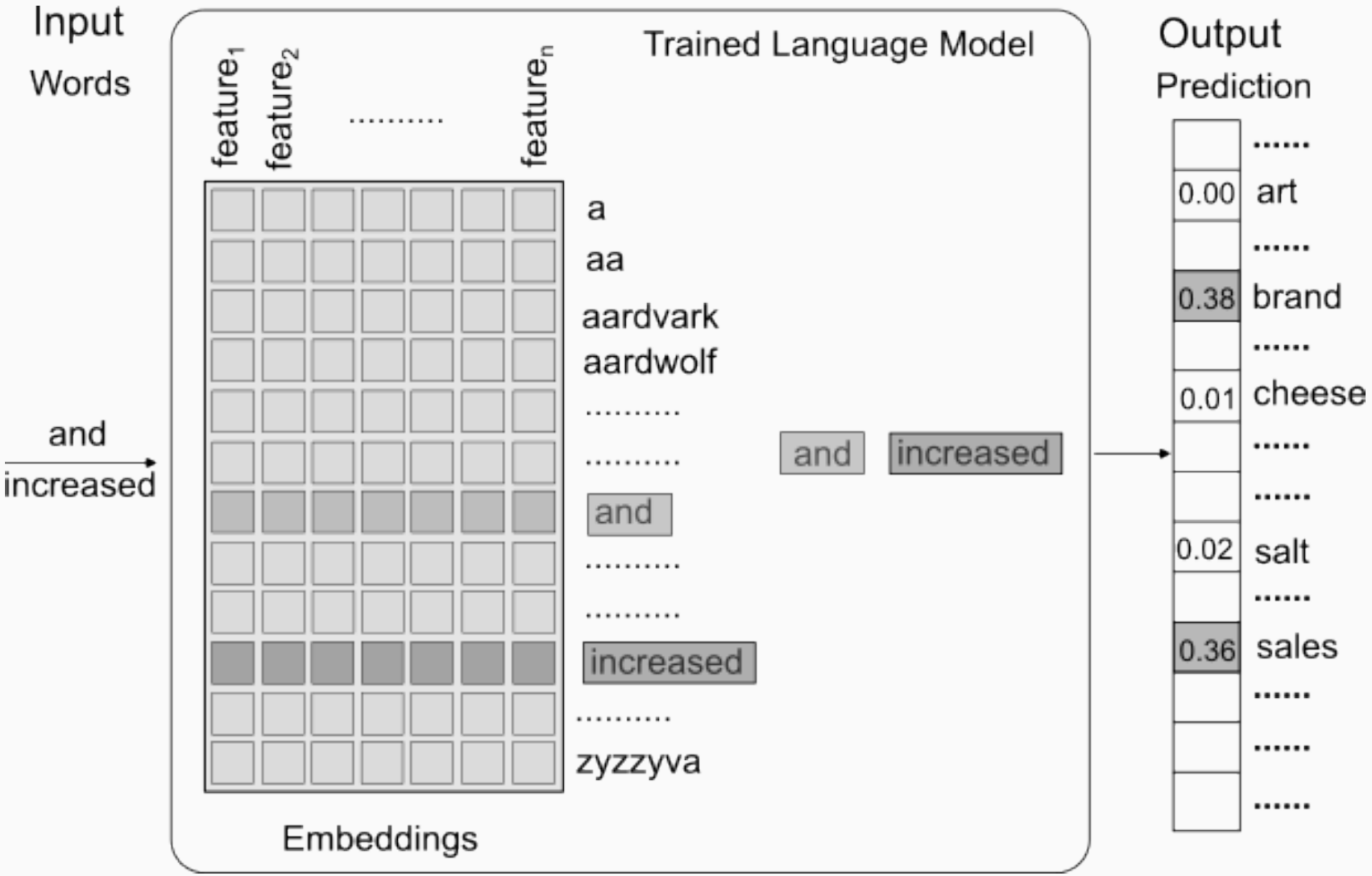
Guess Next Word

The embedding numbers are created through training a neural network. Fear not, we are going to explain these things stepwise. Like all the task in artificial intelligence, the purpose of this operation is to fulfill a prediction task. For example, what word comes after this word, given that so far there have been these words? A direct example is your instant messaging app that tries to guess what would be your next word to write, given that you have already said something. Let us say that you have in input two words such as x and y in a series of 10 000 words, which word will appear with greater probability?

Which Word Would Be?

Let us say that you want to send through your smartphone this message to your boss (you work for a marketing company), with your favorite messaging system, Signal or Telegram: “The summer sale campaign went viral on social media and increased brand awareness.” What words would the system predict for you to type after increased? Sales? Traffic? Engagement?

A Trained Model Would...



Word Embeddings Training

Training, Sliding Window

The summer sale campaign went viral on social media awareness...

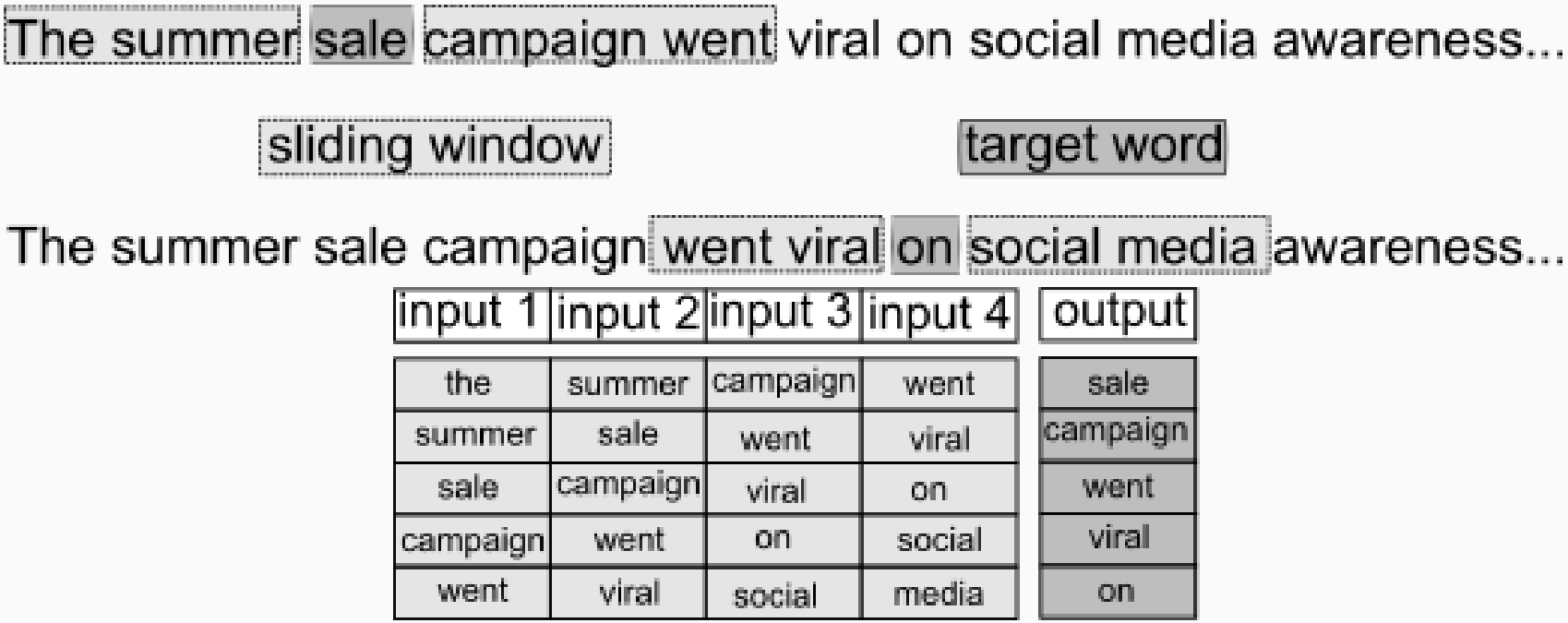
sliding window

the	summer	sale	campaign	went	viral	on	...
the	summer	sale	campaign	went	viral	on	...
the	summer	sale	campaign	went	viral	on	...
the	summer	sale	campaign	went	viral	on	...
the	summer	sale	campaign	went	viral	on	...

input 1	input 2	output
the	summer	sale
summer	sale	campaign
sale	campaign	went
campaign	went	viral
went	viral	on

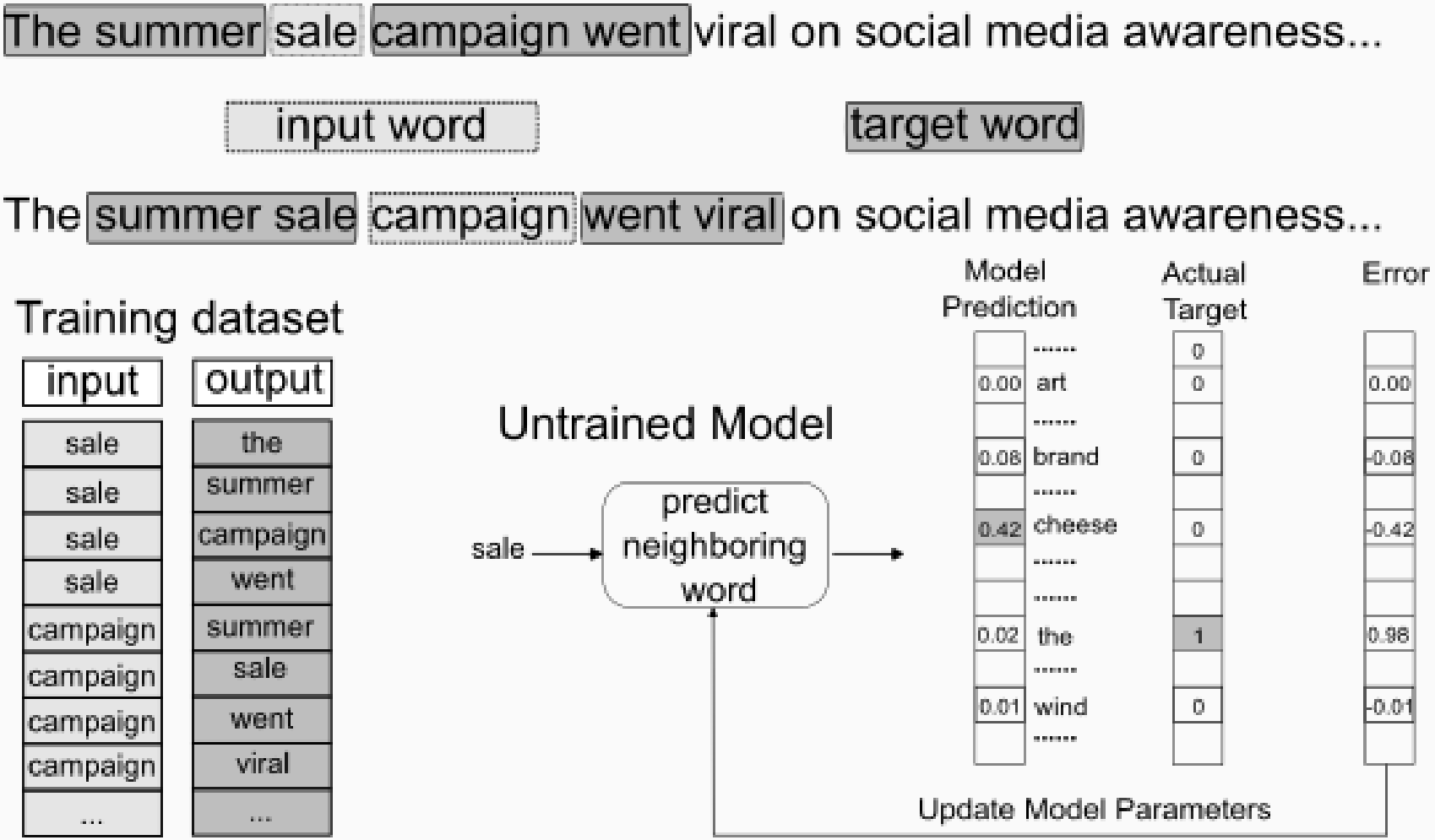
Word Embeddings Training

Training, Cbow



Word Embeddings Training

Training, Big Picture



Training Process

Training, Switching Tasks

How do we get high quality embeddings? Switching the model's task from predicting a neighboring word to a model that takes the input and output word and outputs a score indicating if they'r neigbors or not (1, or 0). This switch, changes to model from a neural network to a logistic regression model, which is simpler and faster to calculate. The structure of our dataset changes, as we have a new target. Not the next word, but if the words are neighbors or not! This can be done very fast but with one caveat. If all the examples are all positive, meaning that target:1, we have a model that always returns 1, 100% accuracy but it learns nothing and generates garbage embeddings. To solve this we introduce "negative samples" to the dataset, samples of words that are not neighbors. The model needs to return 0 for those samples. This is something that is done pretty quickly. What do we rill in as output words? We randomly sample words from our vocabulary. So we randomly select words that are not neighbors.

Negative Sampling

Training dataset

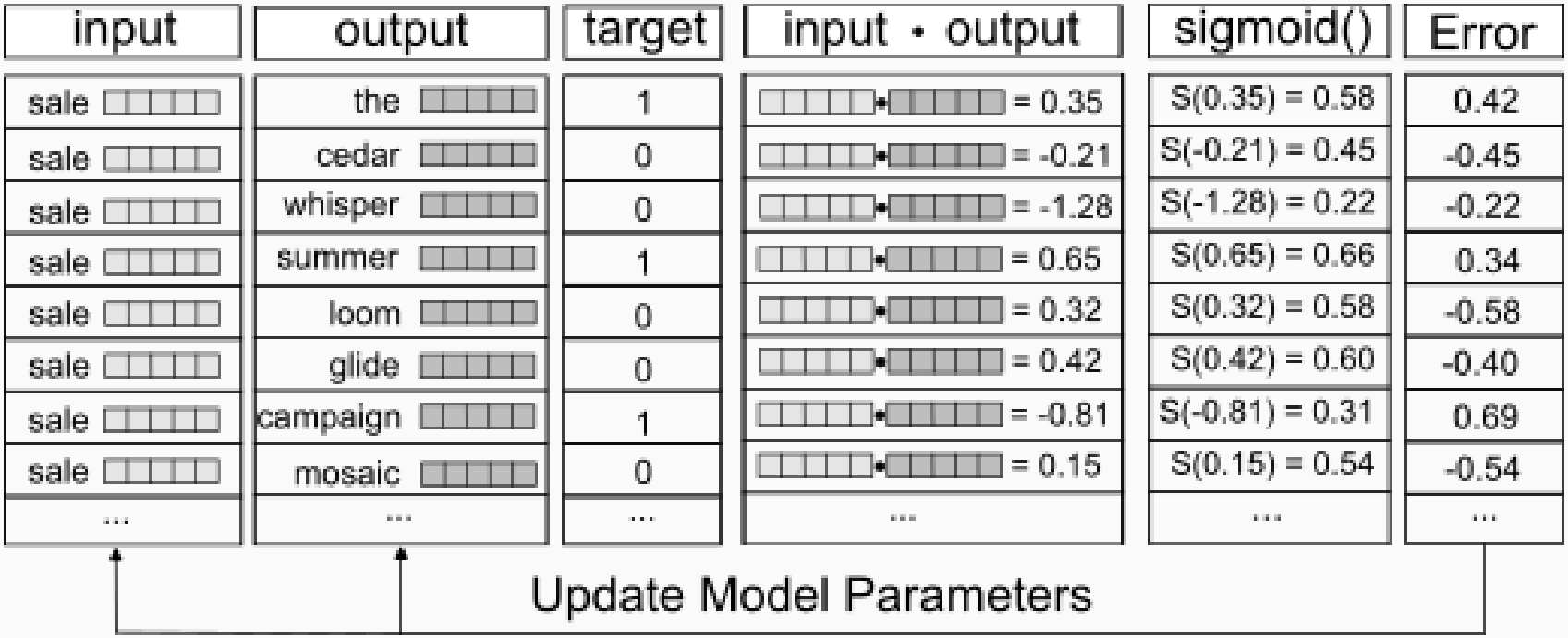
input	output	target
sale	the	1
sale	summer	1
sale	campaign	1
sale	went	1
campaign	summer	1
campaign	sale	1
campaign	went	1
campaign	viral	1
...	...	...

Negative sampling

input	output	target
sale	the	1
sale	cedar	0
sale	whisper	0
sale	summer	1
sale	loom	0
sale	glide	0
sale	campaign	1
sale	mosaic	0
...	...	...

Skipgram with Negative Sampling

Training, Embedding and Context Matrix



Word2vec Training Process

Parameters Update

The update to the parameters (both embedding and context matrices) is typically done using the gradient descent method. The update rule can be expressed as:

$$\theta = \theta - \eta \cdot \nabla_\theta J(\theta)$$

This represents a gradient descent update rule, where θ is the parameter being updated, η is the learning rate, $\nabla_\theta J(\theta)$ is the gradient of the cost function $J(\theta)$ with respect to the parameters θ.

Parameters Update

If we denote the embedding vector of the input word as v and the context vector of the output word as u, and the error calculated as E, the updates can be mathematically expressed as:

1. For the first equation:

$$v_{new} = v_{old} - \eta \cdot \frac{\partial E}{\partial v_{old}}$$

2. For the second equation:

$$u_{new} = u_{old} - \eta \cdot \frac{\partial E}{\partial u_{old}}$$

These equations represent gradient descent updates for the parameters v and u respectively, where η is the learning rate, and $\frac{\partial E}{\partial v}$ and $\frac{\partial E}{\partial u}$ are the gradients of the error function E with respect to v and u.

Parameters Update

The partial derivatives $\frac{\partial E}{\partial v}$ and $\frac{\partial E}{\partial u}$ depend on the specific form of the error function used, which often involves the sigmoid function of the dot product of v and u, compared against the actual target (1 or 0 in binary classification tasks).

Simple Example

Let's define our error function as:

$$E = t - \sigma(v \cdot u)$$

Now, let's follow the steps:

Define the error function:

$$E = t - \sigma(v \cdot u)$$

where σ is the sigmoid function, and t is the target (0 or 1).

Calculate the partial derivatives:

$$\frac{\partial E}{\partial v} = -\sigma(v \cdot u)(1 - \sigma(v \cdot u)) \cdot u$$

$$\frac{\partial E}{\partial u} = -\sigma(v \cdot u)(1 - \sigma(v \cdot u)) \cdot v$$

Simple Example

Update the vectors:

$$v_{\text{new}} = v_{\text{old}} - \eta \cdot (-\sigma(v_{\text{old}} \cdot u_{\text{old}})(1 - \sigma(v_{\text{old}} \cdot u_{\text{old}})) \cdot u_{\text{old}})$$

$$u_{\text{new}} = u_{\text{old}} - \eta \cdot (-\sigma(v_{\text{old}} \cdot u_{\text{old}})(1 - \sigma(v_{\text{old}} \cdot u_{\text{old}})) \cdot v_{\text{old}})$$

Numerical Example

$v = \left[0.5, 0.3\right], u = \left[0.1, 0.2\right]$

$t = 1$ (positive example), $\eta = 0.1$ (learning rate)

Calculate:

$$v \cdot u = 0.5 \cdot 0.1 + 0.3 \cdot 0.2 = 0.11$$

$$\sigma(0.11) \approx 0.5275$$

Error:

$$E = 1 - 0.5275 = 0.4725$$

Gradient for v:

$$-\sigma(0.11) \cdot (1 - \sigma(0.11)) \cdot \left[0.1, 0.2\right] \approx \left[-0.0249, -0.0498\right]$$

Update v:

$$v_{\text{new}} \approx \left[0.5, 0.3\right] - 0.1 \cdot \left[-0.0249, -0.0498\right] \approx \left[0.50249, 0.30498\right]$$

Transformers and more

Transformers

A transformer, in AI, is an important neural network type. It uses word embeddings to learn context and implicitly meaning, by detecting relationships between words in sentences. Transformers appear as the main architectural frame in unsupervised learning models such as BERT or GPT. The link with word embeddings in the previous section is that the input tokens, which are words or subwords, are transformed into arrays of numbers but with a twist. Unlike in Word2Vec, transformers add a position encoding to the vectors to ensure that the model captures the position of each word in a sentence. This dual information is passed on to the actual transformer model. Transformers rely on the self-attention mechanism to process input sequences in parallel.

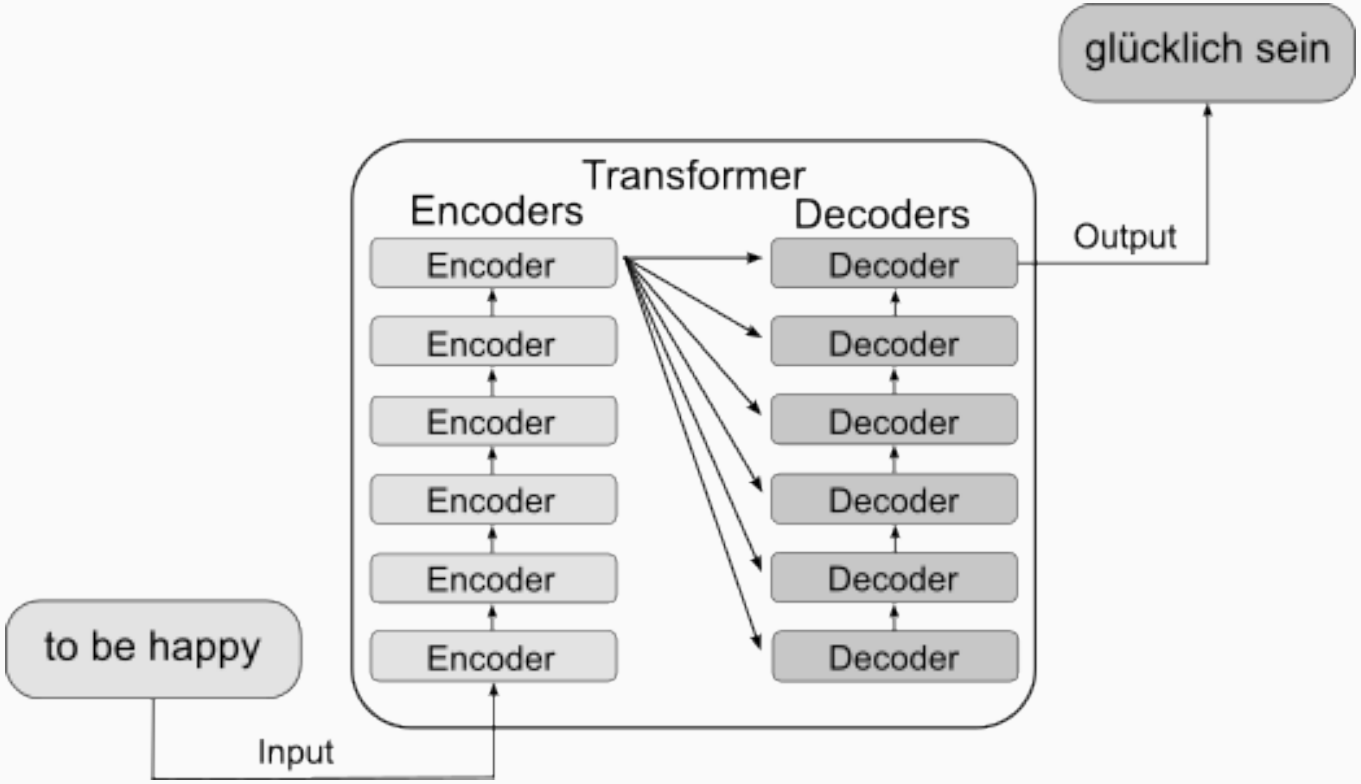
Transformers

The transformer architecture includes an encoder, which processes the input sequence; sometimes (but not always) a decoder, which generates the output sequence; and multi-head self-attention layers, which run in parallel and focus on different aspects of the input. After the attention layer, feedforward layers are applied, and positional encodings are used. Three types of transformers can be identified such as encoder-only, which are designed to understand text, e.g., BERT; decoder-only, which are designed to generate text, e.g., GPT; and encoder-decoder sequences, useful for language translation. The theory presentation in the Transformers examples are inspired from Jay Alammar's online source [1].

[1] https://jalammar.github.io/illustrated-transformer/

Encoder-decoder

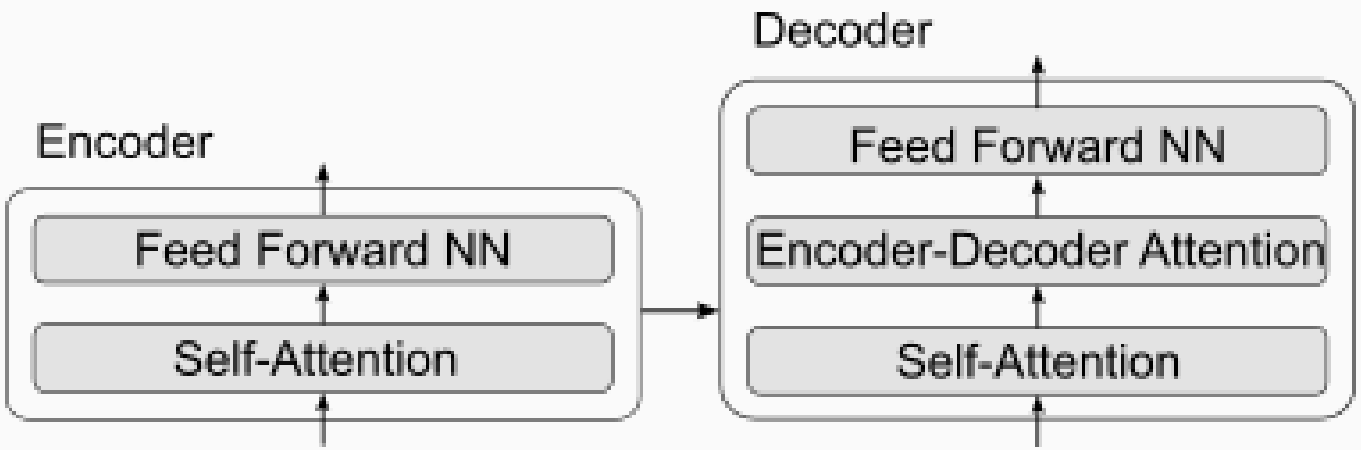
The phrase “to be happy” is sent to the encoder made of six layers, each one representing a processing state. The output from the decoder leads to the correct output translation in German, “glücklich sein.”



Transformer, encoder and decoder

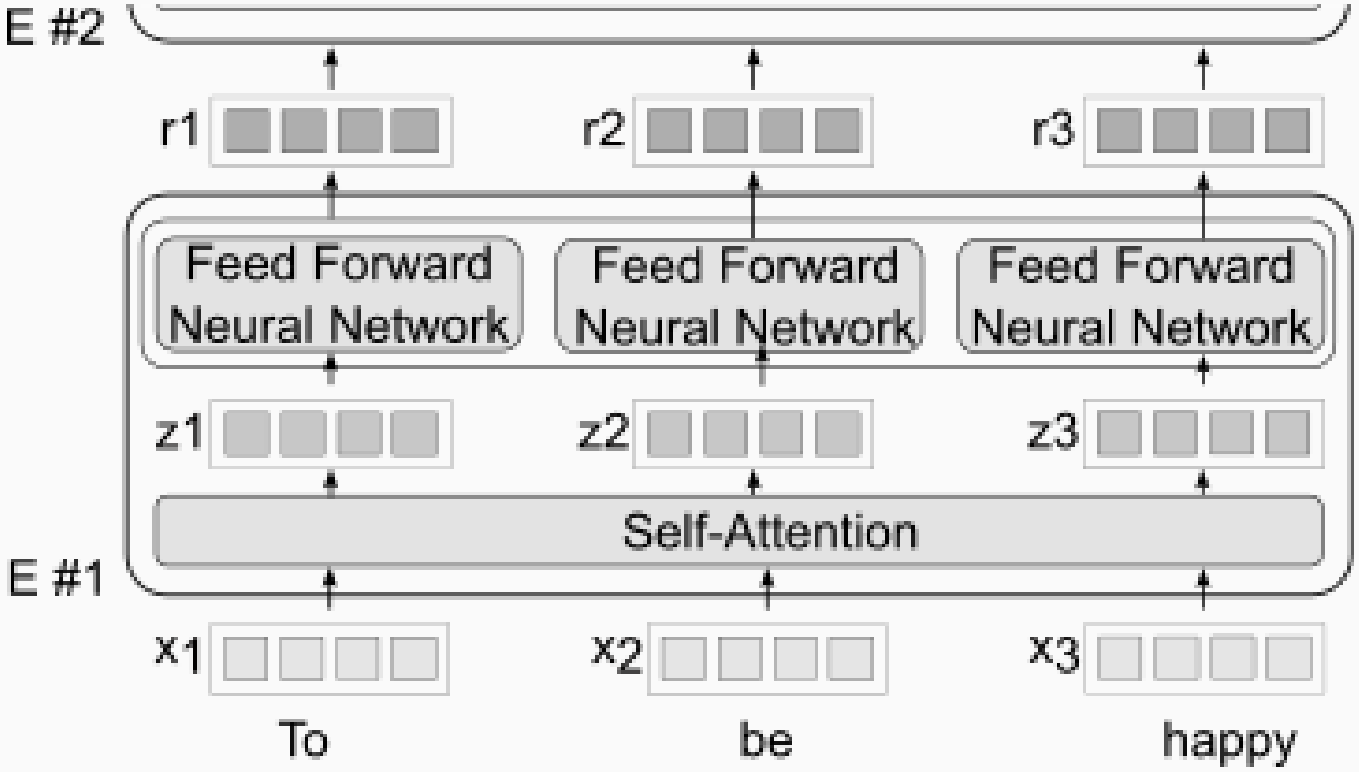
Encoder-decoder

The encoder has a Feed Forward Neural Network that processes data through a series of transformations and a Self-Attention mechanism to evaluate the importance of different sections of the input, aiming at capturing dependencies and relationships in the phrase. The Decoder has its own Feed Forward Neural Network, which is used for the output, an Encoder-Decoder Attention part, which allows the decoder to focus on the important parts of the input sequence, and the Self-Attention mechanism to consider the previous output when generating the next token.



Transformer, encode and decoder detail

The many layers are important so that at each of them, the vector representation can be refined for precision.



Encoder detail

Encoder-decoder

After processing through all the layers, the final output of the encoder stack will contain a context-aware representation of the input sequence and serves as the bridge between the Encoder and Decoder. The main idea is that a word can be translated accurately, but a phrase would get immensely more precision if the context is included. At the end of the encoder for the sentence "to be happy," what's produced is a set of contextualized representations for each word in the sentence. In input, we have "to be happy" (3 tokens). In output we must have three vectors, one for each word, typically of dimension d_{model} (e.g., 512 or 768 as in many implementations).

Vector for "to" can become:

$$\vec{v}_{to} = \left[v_{to,1}, v_{to,2}, \dots, v_{to,d_{model}}\right]$$

Vector for "be" can become:

$$\vec{v}_{be} = \left[v_{be,1}, v_{be,2}, \dots, v_{be,d_{model}}\right]$$

And vector for "happy" can become:

$$\vec{v}_{happy} = \left[v_{happy,1}, v_{happy,2}, \dots, v_{happy,d_{model}}\right]$$

Encoder-decoder

Each of the vectors $\vec{v}_{to}, \vec{v}_{be}, \vec{v}_{happy}$ contains information not only about the word itself, but also about its context inside the sentence. In this layered approach, the self-attention mechanism yields a result and feed-forward layers in the encoder process the relationships between the words. The numerical representation expressed as a vector for the word “to” contains information about its relationship to “be” and “happy,” the one for “be” contains information about its relationship to “to” and “happy”, and the one for “happy” contains information about its relationship to “to” and “be.” These final output vectors capture semantic and syntactic information about each word in the context of the whole sentence. Transformers can be used to accomplish several tasks, such as classification, translation, or even language generation.

What are LLMs?

LLMs are advanced AI models capable of processing, understanding, and generating large amounts of text. These models are abie to learn from vast datasets and they can be used in applications such as chatbots, content creation, education and tutoring, information extraction and data analysis, search engines, linguistic research, healthcare, writing code, etc.

Example of content creating with GPT

Language Models, a bit of a history

Rule-Based Systems (1950s - 1980s): Handcrafted rules to interpret and generate language. ELIZA program (1966), which simulated conversation by matching user inputs to pre-defined patterns and responding with scripted responses.

Statistical Methods (1980s - 2000s): Analyzing large text datasets to find patterns and make predictions about language. Key technique was the n-gram model. It tried to predict the probability of a word based on its preceding 'n-1' words.

Early Machine Learning (Late 1990s - 2000s): Decision trees, support vector machines, and hidden Markov models were used for tasks like speech recognition and part-of-speech tagging.

Neural Networks and Word Embeddings (2000s - Early 2010s): Word embeddings such as Word2Vec (2013) and GloVe (2014) represent words in continuous vector spaces, capturing semantic meaning and enabling more effective language processing (Previous lecture).

Sequence-to-Sequence Models and Attention Mechanisms (Mid-2010s): These models processed sequential data (like sentences) more effectively. The development of attention mechanisms (2014) allowed models to focus on specific parts of the input sequence, improving tasks like translation and text summarization.

Transformers and Pre-trained Language Models (Late 2010s - Present): The Transformer model (2017) enabled more parallel processing and effective handling of long-range dependencies in text. This led to the development of large pre-trained models like GPT (2018), BERT (2018), and their subsequent iterations. These models learn general language representations from vast amounts of text and are fine-tuned for specific tasks.

How do LLMs work

Main principle: learn from huge amounts of text data to understand and generate human-like language.

Training on Huge Datasets: books, websites, articles, etc. The topics are diverse, as well as styles, structures, basically everything. As such there are many examples, rare words, linguistic flavors, etc.

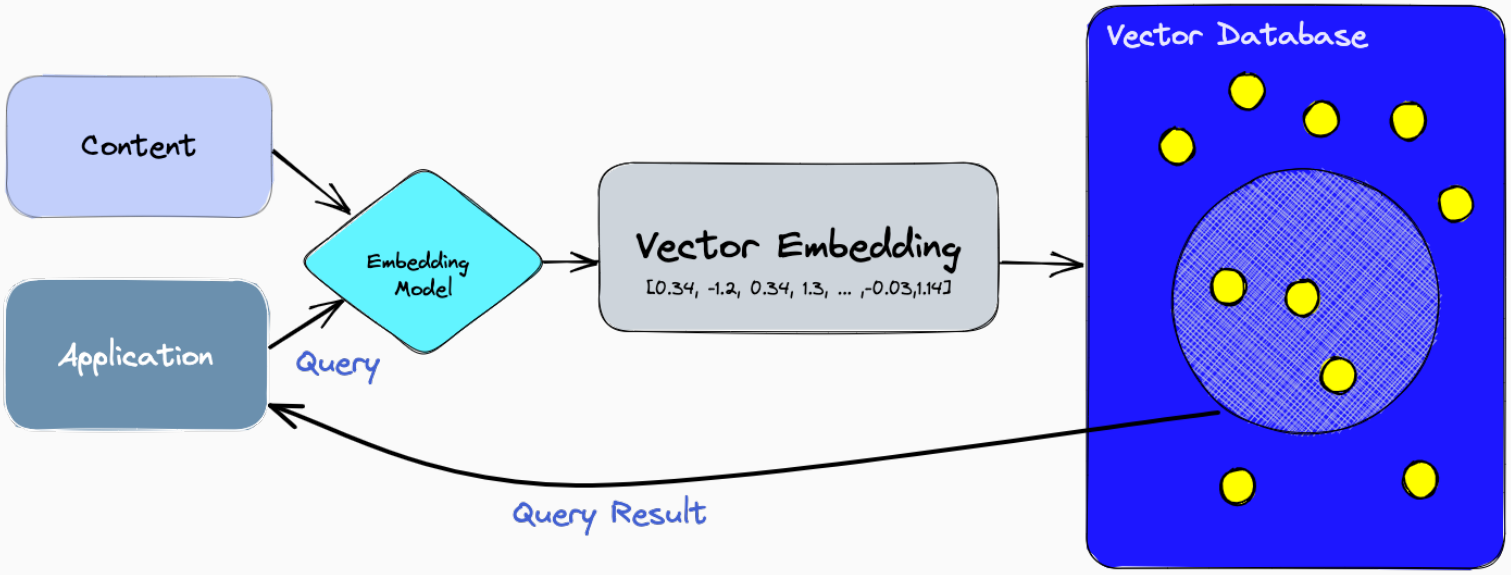
Learning Patterns and Linguistic Structures: Like many AI algorithms, LLMs are systems that recognize patterns (in different contexts). Those based on GPT (i.e. Transformers) are based on the surrounding text. They also learn the grammatical rules and the linguistic structure. There use no pre-programmed rules, but learn them in the training process.

What do they use to parse and generate text?

First, they use a neural network called Transformer. The text input is broken down into tokens (can be anything from words to parts of words or characters) so that the model can process. Then it learns representations for these tokens. They capture their meaning and usage. **The Attention Mechanism** is a crucial feature of the Transformer architecture. This allows the model to weight different parts of the input text differently. In the training process, the model adjusts the internal parameters (weights) to minimize the difference between outputs and expected outputs. This is no different than any other AI algorithm used for this such as backpropagation or stochastic gradient descent.

"Attention Is All You Need" by Vaswani et al. in 2017

In a nutshell. **Self-Attention Mechanism**. It is a variant of the attention mechanism. It weighs the important of different words in a sentence when processing each word. The model creates a contextual representation by considering how it relates to every other word in the sentence. How does it work? For each word in a sentence, the self-attention mechanism computes three vectors: a query vector, a key vector and a value vector. They are all produced from word embeddings. Then an attention score is calculated by taking the dot product of the query vector with the key vectors of all words in the sentence. The scores determine how much focus to put on other parts of the sentence when encoding the given word. This is ot done sequentially but simultaneously.



Attention mechanism

The attention mechanism is the one that allows the model to selectively focus on different parts of the input text when processing each word. In the attention mechanism various level of importance will be assigned to different words based on the computed relevance to the current context. As such, long-range dependencies are captured to have a deeper understanding of the hole text.

The attention scores are computed by comparing each word in the sequence to all other words. Each score says how much each word contributes to the representation of other words. Obviously, the higher the score, the stronger the influence on the model's understanding of the context.

For example in ChatGPT, the attention mechanism helps the model to understand what a user types in input, collect relevant information from he conversation history, and, generate coherent and contextually appropriate responses.

Attention example

I really enjoy Ashley and Ami salon she do a great job be friendly and professional I usually get my hair do when I go to MI because of the quality of the highlight and the price the price be very affordable the highlight fantastic thank Ashley i highly recommend you and ill be back

love this place it really be my favorite restaurant in Charlotte they use charcoal for their grill and you can taste it steak with chimichurri be always perfect Fried yucca cilantro rice pork sandwich and the good tres lech I have had.The desert be all incredible if you do not like it you be a mutant if you will like diabeetus try the Inca Cola

Vector Databases

Vector databases 1 hyperlink

Vector databases 2 hyperlink

Chatbots, Implementation

Chatbots, Conversational Interfaces

Intro

Since very recently, chatbots, expecially ChatGPT are revolutionizing how we interact with knowledge. The AI-powered conversational agents are really transforming customer service, personal assistance, knowledge building, and information retrieval in general. The human-like interactions bring a personalized experience. One of the amazing things is that they keep track of the past conversations (up to a point though).

What is a Chatbot? A chatbot is an AI softare designed to conversate with humans in a human-like manner through text or voice. Chatbots can mainly provide information, but can also automate tasks.

Chatbot types

- There are rule-based chatbots that follow predefined scripts (old style, obsolete)
- AI-powered chatbots use deep learning for dynamic interactions

Components of a Chatbot

1. Natural Language Processing (NLP). The latest trends in NLP allow chatbots to understand, interpret and generate human language. There is no “real” understanding under the hood, but a series of contexts and situations.
2. Dialog Management The dialog management takes care of the flow of conversation, determining (hopefully) appropriate responses based on user input.
3. Knowledge Base A repository of information that the chatbot uses to formulate responses and provide (accurate) information.
4. User Interface The front-end component that allows users to interact with the chatbot through text or voice. All in all, there is no true audio conversation, it is always a voice to text and a text to voice component.

Language Understanding

NLP algorithms analyze user input to extract intent and entities. This process involves tokenization, parsing, and semantic analysis.

Language Generation

NLP generates human-like responses based on the understood intent. It considers **context**, **tone**, and **user preferences**

Machine Learning The continuous improve of the training dataset, adapting to language patterns and user behaviors.

Dialog Management

Conversational flows Predefined pathways guide user interactions, ensuring coherent and purposeful conversations

Retrieval-Augmented Generation (RAGs) RAGs combine information retrieval with language generation. Chatbots answer in a “human-like” manner, but with up-to-date knowledge.

Dyanmic Adaptation Advanced chatbots adjust conversational flows in real-time, based on user inputs and context.

Knowledge Bases and Data Sources

Structured Databases Organized repositories of information enable quick and accurate retrieveal of specific data points.

Unstructured Data Text documents, articles, and web content provide rich sources of information for more complex queries.

API integrations Connections to external services allow chatbots to access real-time data and perform actions.

Machine Learning Models Pre-trained modlels enhance the chatbot's ability to understand context and generate relevant responses

User Interface and Interaction Design

Chat Interface Intuitive chat windows with message bubbles, typing indicators, answer feedback, and emoji support enhance user engagement.

Voice Integration Voice reognition and synthesis allow for hands-free interactions, improving accessibility and convenience.

AR integration Augmented reality interfaces create immersive experiences, blending digital assistants with real world.

Chatbot Integration and Deployment

Cloud Deployment Cloud-based solutions, best for scalability and flexibility, not many problems with hosting and management

Mobile Integration Native applications integrations, tentative IOs

Web Embedding Easy to integrate into websites and create apps and instant support for visitors and internal personnel alike

API Access Enable developers to integrate chatbot functionality into applications or services (RStudio, Python programming, etc.)

Measuring Chatbot Performance

Measuring Chatbot Performance Metrics

Metric	Description	Importance
User Satisfaction	Feedback scores and ratings	High
Task Completion Rate	Successful resolutions	High
Response Time	Speed of chatbot replies	Medium
Conversation Length	Number of interactions	Medium
Fallback Rate	Frequency of unhandled queries	High

Ethical Considerations in Chatbot Design

Transparency Clearly commmunicate the chatbot's artificial nature to users. Avoid deception or misrepresentation of capabilities.

Data Privacy Implmement robust data protection measures. Obtain user consent for data collection and usage.

Bias Mitigation Regularly audit and address potential biases in chatbot responses and decision-making processes.

Accessibility Design chatbots to be inclusive, accommodation users with diverse needs and abilties.

Transformation of Chatbots into Chatbot Agents

Understand input, process information, provide services and virtual assistance (goal oriented)

Examples?

Customer Service Handle inquiries, complaints and troubleshooting

Healthcare Offering preliminary diagnosis, schedule appointments, other health-related services

E-commerce Product recommendation, buy, track order

Education and Training Assist learners with several tasks

Chatbot components

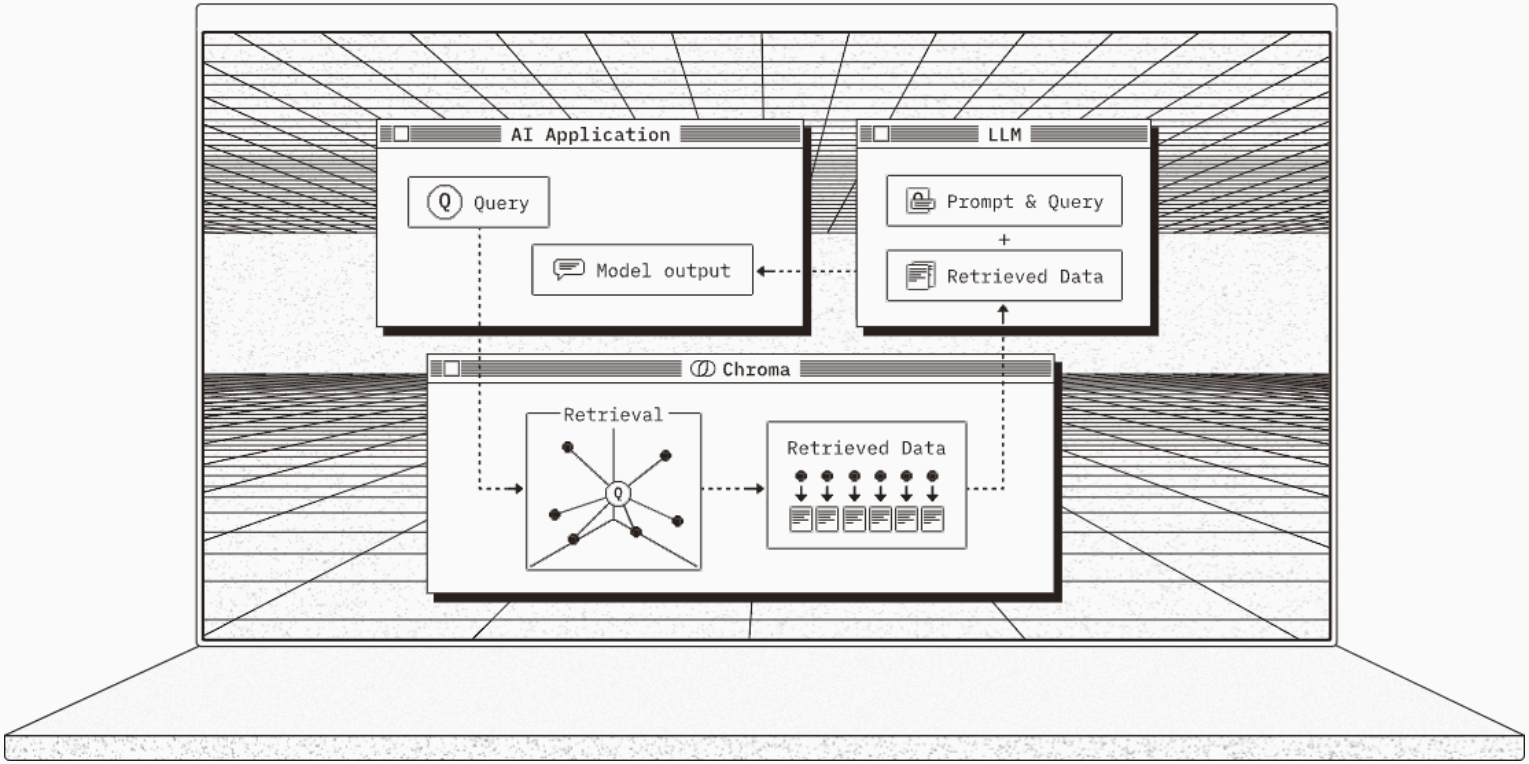
Packages to make a chatbot work

There are several packages needed for building a chatbot

- ChromaDB: Vector store for document embeddings
- Sentence-Transformers: For generating document embeddings (option
- Streamlit: For building a frontend interface
- PyPDF2 (or PyMuPDF): For extracting text from PDFs
- LangChain: Framework for building LLM applications
- RAGs: Retrieval-Augmented Generation.
- LLaMA3.1 (or other models) via Ollama framework: For question-answering

What is ChromaDB?

ChromaDB There must be a place where to store your data. This is an open-source vector database, used for semantic search applications. The best part is that querying and retreiveal of semantical documents is based on embeddings. https://www.trychroma.com/



ChromaDB

ChromaDB features

SIMPLE AND POWERFUL

From notebook to production, prototype, iterate, and deploy your application with Chroma.

Getting started is as easy as pip install...

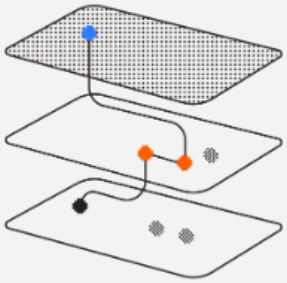
```
> pip install chromadb
```

```
import chromadb

client = chromadb.Client()

c = client.create_collection("te

# add embeddings and documents
c.add(...)

# get back similar ones
c.query(...)
```

FULL FEATURED

Everything you need for retrieval:

Vector search
Full-text search
Document storage
Metadata filtering
Multi-modal retrieval

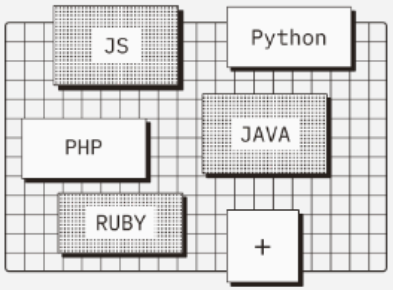


FREE

Open source under Apache 2.0

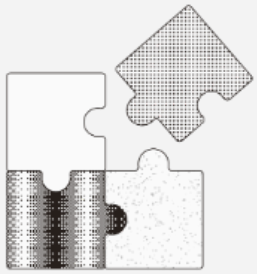
SPEAKS YOUR LANGUAGE

Clients for JS, Python, Ruby, PHP, Java and more.



INTEGRATED

Embedding models from HuggingFace, OpenAI, Google, and more, built in. Integrated with Langchain, Llamaindex, with more tools and frameworks coming soon.



ChromaDB, get started

INSTALL

Get up and running instantly.

Get started

```
$ # install chroma
$ pip install chromadb
$
$ # run chroma
$ chroma run
```

DEPLOY

Run Chroma on AWS, GCP, Azure and more.

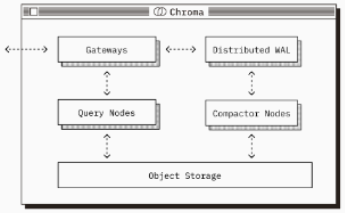
View guides

```
terraform {
  required_providers {
    aws = {
      source  = "hashicorp/a
      version = "~> 5.0"
    }
  }
}

# Define provider
# ...
```

SCALE

Chroma's distributed architecture is horizontally scalable, multi-tenant, and cloud native - powered by object storage.

Learn more



CLOUD

Chroma's developer cloud is fully-managed, elastically scalable, with serverless billing. With the same API.

Join the waitlist

```
# one line migration

# local client (before)
client = chromadb.HttpClient

# cloud client (after)
client = chromadb.CloudClien

# that's it!
```

Get Started

Sentence transformers

Sentence-Transformers Known as SBERT, is a Python library to access, use, and train high-quality sentence embeddings. It is built on Hugging Face Transformers and PyTorch. It can generate dense, fix-length embeddings to capture **semantic meaning** rater than encoding individual word meaning.

SBERT can be used to detect semantic similarity, clustering, information retrival, and paraphrase mining. There are more than 5.000 pre-trained Sentence Transformers models on Hugging Face. Many of these models are found on the Massive Texte Embedding Benchmark. MTEB spans 8 embedding tasks covering a total of 213 datasets and 113 languages (05/11/2024). (MTEB) website https://huggingface.co/spaces/mteb/leaderboard.

Easy to train or finetune your own models using Sentence Transformers. You can create custom models for your specific use cases.

Finetune Sentence Transformers

Finetuning heavily improves the performance of the model!

Source: https://sbert.net/docs/sentence_transformer/training_overview.html

2 different articles:

"Apple launches the new iPad"

"NVIDIA is gearing up for the next GPU generation"

We may have different notions of similarity:

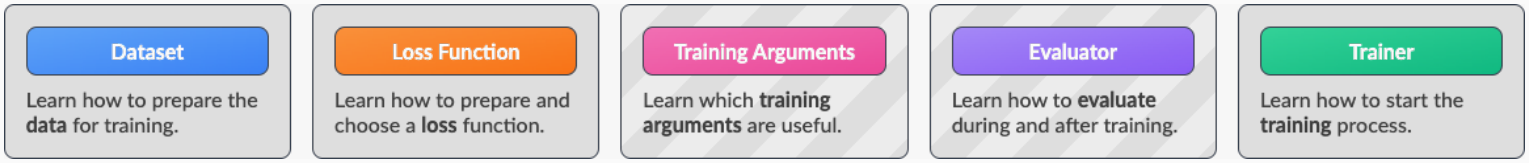
A model for **classification** of news articles as Economy, Sports, Technology, Politics, etc., should produce similar embeddings for these texts.

A model for **semantic textual similarity** should produce **dissimilar embeddings** for these texts, as they have different meanings.

A model for **semantic search** would **not need a notion for similarity** between two documents, as it should only compare queries and documents.

Training components

Training Sentence Transformer models involve between 3 to 5 components:



Get Started

Key Training Arguments for improving training performance

Key Training Arguments for improving training performance

Column1	Column2	Column3	Column4
learning_rate	lr_scheduler_type	warmup_ratio	num_train_epochs
max_steps	per_device_train_batch_size	per_device_eval_batch_size	auto_find_batch_size
fp16	bf16	gradient_accumulation_steps	gradient_checkpointing
eval_accumulation_steps	optim	batch_sampler	multi_dataset_batch_sampler

Key Training Arguments for observing training performance

Column1	Column2	Column3	Column4
eval_strategy	eval_steps	save_strategy	save_steps
save_total_limit	load_best_model_at_end	report_to	log_level
logging_steps	push_to_hub	hub_model_id	hub_strategy
hub_private_repo			

Evaluators that come with Sentence Transformers

Evaluator Types and Required Data

Evaluator	Required_Data
BinaryClassificationEvaluator	Pairs with class labels
EmbeddingSimilarityEvaluator	Pairs with similarity scores
InformationRetrievalEvaluator	Queries (qid => question), Corpus (cid => document), and relevant documents (qid => set[cid])
MSEEvaluator	Source sentences to embed with a teacher model and target sentences to embed with the student model. Can be the same texts.
ParaphraseMiningEvaluator	Mapping of IDs to sentences & pairs with IDs of duplicate sentences.
RerankingEvaluator	List of {'query': '...', 'positive': [...], 'negative': [...]} dictionaries.
TranslationEvaluator	Pairs of sentences in two separate languages.
TripletEvaluator	(anchor, positive, negative) pairs.

PyPDF2 (or PyMuPDF)

PyPDF2 and PyMuPDF (aka fitz), Python libraries used for extracting text from PDF files.

PyPDF2 does a good job in text PDF processing. It has limited support for complex PDF files, images, tables, or complex formatting. The quality of the outcome depends on the PDF structure.

PyMuPDF supports complex PDFs. Extraction of text, images, tables provide accurate rendering. Good for extracting image data, can help provide context-sensitive responses. It requires more resources, and some more installation steps than PyPDF2.

Other tools, by file type:

PDF PyPDF2, PyMuPDF, PDFMiner, pdfplumber **docx** python-docx **multiple formats** textract (PDF, DOCX, XLSX, PPTX, JPG, and some more), tika Apache Tika Python Wrapper (PDFs, Word, HTML, etc.)

LangChain

LangChain is a framework for building applications that involve Large Language Models and it has tools and modules and integrations to perform several workflows.

What are the “chains?”

The framework, LangChain allows developers to create prompts in a chain-like manner, meaning that the output of one LLM is used as the input to another, creating complex structures.

An important characteristic of LangChain is the possibility to implement an **Agent** that can interact with eternal tools, APIs, etc. to deal with multi-step queries, and actions.

Memory (or cache memory), allows LLMs to “keep in mind” the previous interactions.

Integration with external data, allowing the LLM to work with domain-specific information. (RAGs)

Ollama

Platform that provides access to language models that are already fine-tuned and, very important, can run on your computer. You can download, manage and run specific LLMs (Gamma2, for example) without the need to connect to external servers. Why is this important? Privacy reasons, latency, cost, etc.

Several types of models for several purposes:chat, code generation, etc.

Ollama can work together with LangChain. Example. Build a chatbot that uses LangChain and runs a series of propmts and responses (employing Langchai's memory). It can use a model runninc locally on Ollama as teh main LLM.

Retrieval-Augmented Generation.

RAG is a technique that combines retrieval of relevant information with generation of text to improve the quality and relevance of LLM outputs. It leverages two main components, Retriever and Generation. The first one, finds the most relevant piece of information, such as the one from your pdf files. The Generator part generates a response including the information found from the pdf files.

Primarily, there are two models of RAGs: RAG-Sequence and RAG-Token. The input query and the found information are concatenated to form a long single input which is fed to the generator. Good when you have several sources of information. Rag-Token does not use a single concatenated sequence. It generates each output token one at a time while retreiving passages to inform each token generation.This improves response accuracy

Uses: Question-Answering Systems, Customer Support Chatbots, Personalized Content Generation, Medical and Legal Consultation, etc.

Streamlit

An open-source Python library for building interactive web applications

Converts Python scripts into shareable apps very quickly. It has interactive widgets, buttons, sliders, for real-time interactivity. Supports libraries, such as matplotlib, plotly, seaborn. It uses caching to optimize heavy computation. Very easy to deploy, on your own machine, streamlit cloud, etc.

Example: https://streamlit.io/gallery?category=llms

Snakes?

Step 1

Install Anaconda or Miniconda or something that allows you to work with Python. Open your terminal or Anaconda Prompt and run:

```
# Create the environment
conda create --name my_chatbot python=3.12

# Activate the Conda environment
conda activate my_chatbot
```

Other things to install

0. Could be useful, install NPM (Node Package Management) Node.js (check for your operating system) (but it is not necessary to have it)

```
node -v
npm -v
```

1. Install Ollama from the ollama download page: Ollama. This IS useful! In a console verify if ollama is installed, type:

```
ollama
```

2. Download some language models. In a console type

```
ollama pull llama3.1
# or
ollama pull gemma2
```

Be aware, we are talking about several GB of data!!!

Other things to install

Install the ollama python package to integrate ollama into the python environment

```
pip install ollama
```

Install all the other dependencies

```
pip install chromadb sentence-transformers streamlit PyPDF2
pip install langchain jupyter
```

Install IPython kernel for your environment

```
python -m ipykernel install --user --name=my_chatbot
```

Start Jupyter Notebook from the console (or terminal)

```
jupyter notebook
```

Extract text from your pdf files

Use PyPDF2 to extract text from your pdf documents. This is typically done only once, before running your chatbot. Place all your pdf files in a directory, e.g., ./mypdfs/.

PyPDF2 is a Python library, you can use it to manipulate pdfs. For example, you can extract selected pages, merge multiple pdfs, split documents, etc.

Generate embeddings

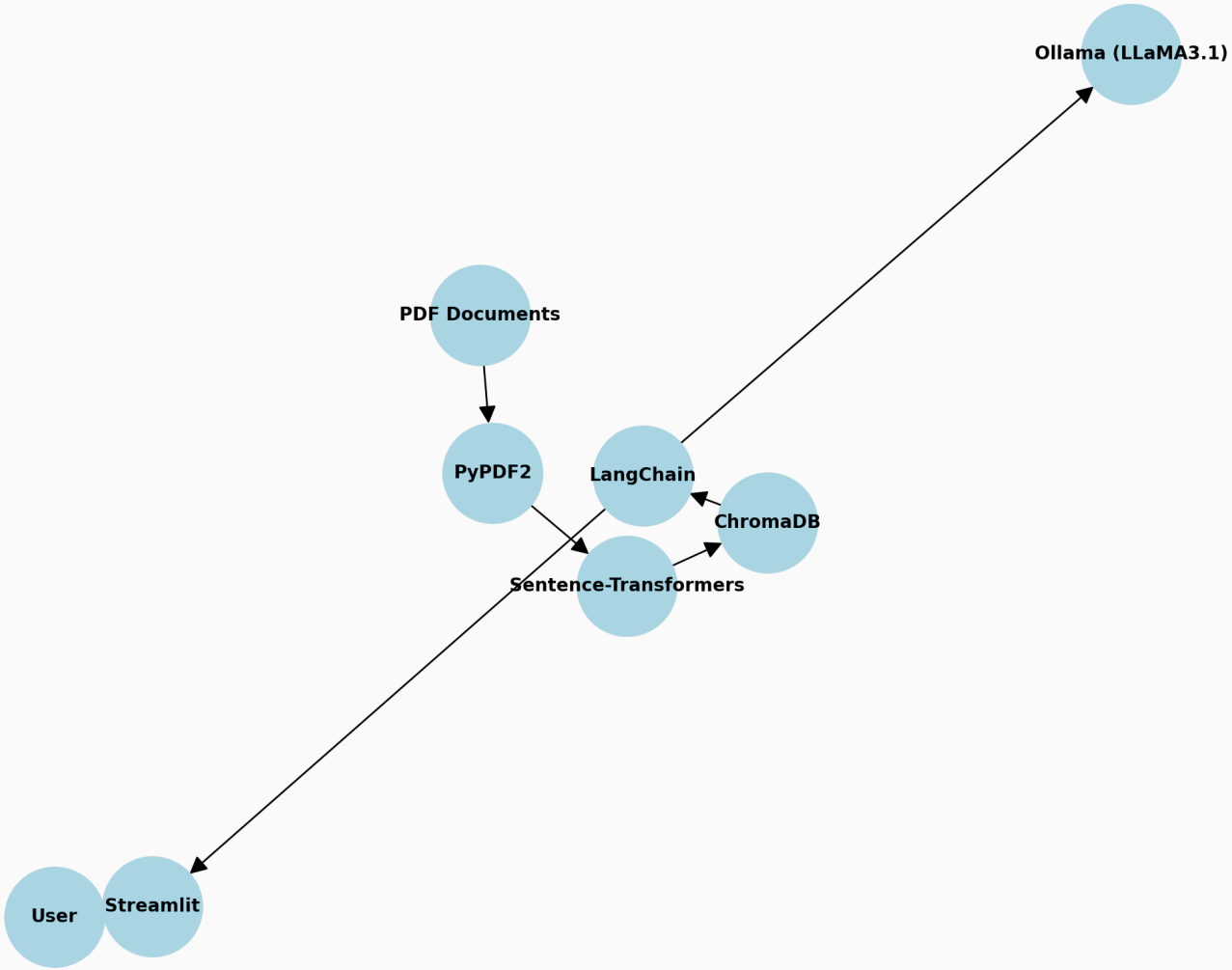
Use Sentence-Transformers to generate embeddings for the extracted text. This creates the famous vector representations of your documents. This step can be explicit, with various way to implement it. Here we will deal with simple things.

Store Embeddings in ChromaDB

We will use ChromaDB to store the generated embeddings but we could also use FAISS, another scalable alternative. This creates a searchable vector database of the documents tranformed from your mypdfs folder. This operation will create another folder with the vectorized texts.

LangChain and Ollama

The framework we will use to create a retrieval system that can query ChromaDB and format prompts for the LLM is LangChain. Configure Ollama to run the LLaMA3.1 model locally.



Make things more interactive

Develop a Streamlit app (code in Python) that will serve as the frontend for your chatbot

In your Streamlit app, integrate ChromaDB for retrieval, LangChain for managing the flow, and Ollama for generating responses. You can find all in the code provided.

Run your Streamlit app to deploy the chatbot.