Colab link:https://colab.research.google.com/drive/1GpsP-3l95M0hyZ8Dv8ptrjBJPuxGyj1n (https://colab.research.google.com/drive/1GpsP-3l95M0hyZ8Dv8ptrjBJPuxGyj1n)

Colab link models:
https://colab.research.google.com/drive/1JpXZ9XXCwny86wpeltn0psLlJtw1aD7K#scrollTo=ddtLlLDmJ7ng (https://colab.research.google.com/drive/1JpXZ9XXCwny86wpeltn0psLlJtw1aD7K#scrollTo=ddtLlLDmJ7ng)

## Introduction

This project will employ machine learning to classify different newsgroups data and explore additional trends and observations. The dataset used is the well-known 20 Newsgroups dataset, which has become quite frequency used in the field of machine learning.The 20 Newsgroups dataset is a collection of approximately 20,000 newsgroup documents, partitioned (nearly) evenly across 20 different newsgroups. It was originally collected by Ken Lang, probably for his Newsweeder: Learning to filter netnews paper, though he does not explicitly mention this collection. The reasoning behind choosing newsgroups data is mainly connected to also having real life applications. Through an advanced neural network it is possible to classify a certain piece of news simply by its content. This can be applied when uncertainty occurs about what category certain news might belong to. In this case the journalist can use the algorithm to help determine what specific category it might belong to. Additionally the algorithm can be used to suggest another category which can make the news fall into multiple categories. This can save time from various journalists/news publishers, who are no longer required to manually determine what category certain news belong to. This project will start by exploring the data with the use of a Glove embedding model, where the link between different words is being researched. Afterwards a network analysis will be constructed to explore how pair of words are connect with the use of Bigrams. After exploring the text, a baseline Random Forest model will be constructed to see how well it performs. Subsequently a neural network will be constructed with the goal of classlifying multiple topics. The neural network used in this project is "Long Short Term Memory", which is a superior neural network for sequential text data. Lastly the project will conclude upon the discoveries made.

## Data preparation

First, we read in the data and inspect it. The inspection has the purpose of checking that the data is as we expect, for this we use the glimpse command and find nothing unexpected. Next, we check the topic variable to both see if there are the 20 categories we expect, and to find if there is a somewhat even distribution between the 20 topics.

In [1]:
```
install.packages("tm")
install.packages("data.table")
install.packages("text2vec")
install.packages("quanteda")
install.packages("tidytext")
install.packages("textstem")
install.packages("dplyr")
install.packages("widyr")
install.packages("ggforce")  # Awesome plotting
install.packages("ggraph")
install.packages("uwot")
install.packages("igraph") # For network analysis
install.packages("ggraph")
install.packages("tidygraph")
install.packages("mapplots")
install.packages("tidyr")
library(tidyr)
library(tidyverse)
library(data.table)
library(tm)
library(mapplots)
library(ggforce)
library(ggraph)
library(uwot)
library(igraph)
library(ggraph)
library(tidygraph)
library(widyr)
library(dplyr)
library(textstem)
library(text2vec)
library(quanteda)
library(tidytext)

data=read_csv("https://www.dropbox.com/s/y29q10pyrkdtz67/Newspaper.csv?dl=1")
```

```
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
also installing the dependencies ‘NLP’, ‘slam’

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
also installing the dependencies ‘formatR’, ‘iterators’, ‘lambda.r’, ‘futile.
options’, ‘RcppParallel’, ‘foreach’, ‘irlba’, ‘futile.logger’, ‘mlapi’, ‘spar
sepp’

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
also installing the dependencies ‘coda’, ‘extrafontdb’, ‘Rttf2pt1’, ‘RcppEige
n’, ‘statnet.common’, ‘reticulate’, ‘ISOcodes’, ‘extrafont’, ‘fastmatch’, ‘gg
repel’, ‘network’, ‘RSpectra’, ‘sna’, ‘SnowballC’, ‘spacyr’, ‘stopwords’, ‘pr
oxyC’, ‘RcppArmadillo’

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
also installing the dependencies ‘hunspell’, ‘tokenizers’, ‘janeaustenr’

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
also installing the dependencies ‘zoo’, ‘dtt’, ‘sylly.en’, ‘sylly’, ‘syuzhe
t’, ‘english’, ‘mgsub’, ‘qdapRegex’, ‘koRpus.lang.en’, ‘koRpus’, ‘lexicon’,
‘textclean’, ‘textshape’

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
also installing the dependencies ‘farver’, ‘tweenr’, ‘polyclip’

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
also installing the dependencies ‘gridExtra’, ‘igraph’, ‘viridis’, ‘tidygrap
h’, ‘graphlayouts’

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
also installing the dependencies ‘sitmo’, ‘FNN’, ‘RcppAnnoy’, ‘RcppProgress’,
‘dqrng’

Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
Installing package into ‘/usr/local/lib/R/site-library’
(as ‘lib’ is unspecified)
```

```
── Attaching packages ─────────────────────────────────────── tidyverse 1.2.1
──
✓ ggplot2 3.2.1     ✓ purrr   0.3.3
✓ tibble  2.1.3     ✓ dplyr   0.8.3
✓ readr   1.3.1     ✓ stringr 1.4.0
✓ ggplot2 3.2.1     ✓ forcats 0.4.0
── Conflicts ────────────────────────────────────────── tidyverse_conflicts()
──
✗ dplyr::filter() masks stats::filter()
✗ dplyr::lag()    masks stats::lag()

Attaching package: ‘data.table’

The following objects are masked from ‘package:dplyr’:

    between, first, last

The following object is masked from ‘package:purrr’:

    transpose

Loading required package: NLP

Attaching package: ‘NLP’

The following object is masked from ‘package:ggplot2’:

    annotate

Loading required package: Matrix

Attaching package: ‘Matrix’

The following objects are masked from ‘package:tidyr’:

    expand, pack, unpack


Attaching package: ‘igraph’

The following objects are masked from ‘package:dplyr’:

    as_data_frame, groups, union

The following objects are masked from ‘package:purrr’:

    compose, simplify

The following object is masked from ‘package:tibble’:

    as_data_frame

The following object is masked from ‘package:tidyr’:

    crossing

The following objects are masked from ‘package:stats’:
    decompose, spectrum

The following object is masked from ‘package:base’:

    union


Attaching package: ‘tidygraph’

The following object is masked from ‘package:igraph’:

    groups

The following object is masked from ‘package:stats’:

    filter

Loading required package: koRpus.lang.en
Loading required package: koRpus
Loading required package: sylly
For information on available language packages for 'koRpus', run

  available.koRpus.lang()

and see ?install.koRpus.lang()


Attaching package: ‘koRpus’

The following object is masked from ‘package:readr’:

    tokenize


Attaching package: ‘text2vec’

The following object is masked from ‘package:igraph’:

    normalize

Package version: 1.5.2
Parallel computing: 2 of 2 threads used.
See https://quanteda.io for tutorials and examples.

Attaching package: ‘quanteda’

The following objects are masked from ‘package:koRpus’:

    tokens, types

The following objects are masked from ‘package:tidygraph’:

    as.igraph, convert

The following object is masked from ‘package:igraph’:
    as.igraph

The following objects are masked from ‘package:tm’:

    as.DocumentTermMatrix, stopwords

The following object is masked from ‘jupyter:irkernel’:

    View

The following object is masked from ‘package:utils’:

    View

Warning message:
“Missing column names filled in: 'X1' [1]”Parsed with column specification:
cols(
  X1 = col_double(),
  text_id = col_double(),
  text = col_character(),
  category = col_character()
)
```

In [0]:
```
data %<>% mutate(topic=category) %>% select(-category)
```

In [3]:
```
data %>% glimpse
```

```
Observations: 18,773
Variables: 4
$ X1      <dbl> 1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15, 16, 17, 1
8…
$ text_id <dbl> 1366, 6368, 7292, 3951, 7399, 5424, 2652, 2680, 5195, 4420,
 3…
$ text    <chr> "From: csfed@uxa.ecn.bgu.edu (Frank Doss)\r\n Subject: Re: S
c…
$ topic   <chr> "['alt.atheism']", "['sci.space']", "['talk.religion.misc']",
…
```

```
In [4]: data %>% head(1)
```

A tibble: 1 × 4

| X1 | text_id | text | topic |
|---|---|---|---|
| <dbl> | <dbl> | <chr> | <chr> |
| 1 | 1366 | From: csfed@uxa.ecn.bgu.edu (Frank Doss) Subject: Re: Science and theories Organization: Educational Computing Network Lines: 27 NNTP-Posting-Host: uxa.ecn.bgu.edu In article <C5u7Bq.J43@news.cso.uiuc.edu> cobb@alexia.lis.uiuc.edu (Mike Cobb) writes: >The examples he gave were quarks and continental plates. Are there Sounds like more of the same. Gods were used to describe almost everything in the past. As we come to understand the underpinnings of more and more, the less we credit to a god. Now, the not-so-well understood elements (at least by the author) includes quarks and tectonic drift. I guess that's better than describing the perceived patterns of stars in the sky as heroes being immortalized by the gods. Kinda sounds like old-earth creation--It seems that life did, indeed, evolve from a common ancestor. What caused that initial common ancestor? Are we going to hear another debate on causeless events? ;-) >explanations of science or parts of theories that are not measurable in and of >themselves, or can everything be quantified, measured, tested, etc.? > >MAC > Michael A. Cobb -- Frank Doss The above stated words are my opinions and do not reflect the opinions, attitudes, or policies of my employer or any affilliated organizations. | ['alt.atheism'] |

In [5]:
```
data %>% group_by(topic) %>% count()
```

A grouped_df: 40 × 2

| topic | n |
|---:|---:|
| <chr> | <int> |
| ['alt.atheism'] | 317 |
| ['comp.graphics'] | 389 |
| ['comp.os.ms-windows.misc'] | 394 |
| ['comp.sys.ibm.pc.hardware'] | 389 |
| ['comp.sys.mac.hardware'] | 381 |
| ['comp.windows.x'] | 394 |
| ['misc.forsale'] | 388 |
| ['rec.autos'] | 393 |
| ['rec.motorcycles'] | 398 |
| ['rec.sport.baseball'] | 396 |
| ['rec.sport.hockey'] | 393 |
| ['sci.crypt'] | 396 |
| ['sci.electronics'] | 390 |
| ['sci.med'] | 395 |
| ['sci.space'] | 391 |
| ['soc.religion.christian'] | 398 |
| ['talk.politics.guns'] | 364 |
| ['talk.politics.mideast'] | 376 |
| ['talk.politics.misc'] | 310 |
| ['talk.religion.misc'] | 251 |
| alt.atheism | 475 |
| comp.graphics | 579 |
| comp.os.ms-windows.misc | 591 |
| comp.sys.ibm.pc.hardware | 588 |
| comp.sys.mac.hardware | 567 |
| comp.windows.x | 590 |
| misc.forsale | 583 |
| rec.autos | 593 |
| rec.motorcycles | 598 |
| rec.sport.baseball | 591 |
| rec.sport.hockey | 600 |
| sci.crypt | 594 |
| sci.electronics | 588 |
| sci.med | 592 |

| topic | n |
|---:|---:|
| <chr> | <int> |
| sci.space | 593 |
| soc.religion.christian | 599 |
| talk.politics.guns | 546 |
| talk.politics.mideast | 564 |
| talk.politics.misc | 465 |
| talk.religion.misc | 374 |

Looking at the topic variable a glaring issue jumps out, there are duplicates of each category an issue we will address twice as we are using both R and python in our project. In each case we will redefine the variable so the categories are uniform, leaving us with a dataset that can be used in our different models further preprocessing will be required before the different models.

In [0]:
```
data$topic=ifelse(data$topic=="['alt.atheism']","atheism",data$topic)
data$topic=ifelse(data$topic=="alt.atheism","atheism",data$topic)

data$topic=ifelse(data$topic=="['comp.graphics']","compgraphics",data$topic)
data$topic=ifelse(data$topic=="comp.graphics","compgraphics",data$topic)

data$topic=ifelse(data$topic=="['comp.os.ms-windows.misc']","comp-windows",dat
a$topic)
data$topic=ifelse(data$topic=="comp.os.ms-windows.misc","comp-windows",data$to
pic)

data$topic=ifelse(data$topic=="['comp.sys.ibm.pc.hardware']","ibm-pc-hardware"
,data$topic)
data$topic=ifelse(data$topic=="comp.sys.ibm.pc.hardware","ibm-pc-hardware",dat
a$topic)

data$topic=ifelse(data$topic=="['comp.sys.mac.hardware']","mac-hardware",data$
topic)
data$topic=ifelse(data$topic=="comp.sys.mac.hardware","mac-hardware",data$topi
c)

data$topic=ifelse(data$topic=="['comp.windows.x']","windows-x",data$topic)
data$topic=ifelse(data$topic=="comp.windows.x","windows-x",data$topic)

data$topic=ifelse(data$topic=="['misc.forsale']","misc-forsale",data$topic)
data$topic=ifelse(data$topic=="misc.forsale","misc-forsale",data$topic)

data$topic=ifelse(data$topic=="['rec.autos']","autos",data$topic)
data$topic=ifelse(data$topic=="rec.autos","autos",data$topic)

data$topic=ifelse(data$topic=="['rec.motorcycles']","motorcycles",data$topic)
data$topic=ifelse(data$topic=="rec.motorcycles","motorcycles",data$topic)

data$topic=ifelse(data$topic=="['rec.sport.baseball']","baseball",data$topic)
data$topic=ifelse(data$topic=="rec.sport.baseball","baseball",data$topic)

data$topic=ifelse(data$topic=="['rec.sport.hockey']","hocey",data$topic)
data$topic=ifelse(data$topic=="rec.sport.hockey","hocey",data$topic)

data$topic=ifelse(data$topic=="['sci.crypt']","crypt",data$topic)
data$topic=ifelse(data$topic=="sci.crypt","crypt",data$topic)

data$topic=ifelse(data$topic=="['sci.electronics']","electronics",data$topic)
data$topic=ifelse(data$topic=="sci.electronics","electronics",data$topic)

data$topic=ifelse(data$topic=="['sci.med']","medicin",data$topic)
data$topic=ifelse(data$topic=="sci.med","medicin",data$topic)

data$topic=ifelse(data$topic=="['sci.space']","space",data$topic)
data$topic=ifelse(data$topic=="sci.space","space",data$topic)

data$topic=ifelse(data$topic=="['soc.religion.christian']","christianity",data
$topic)
data$topic=ifelse(data$topic=="soc.religion.christian","christianity",data$top
ic)
```

```
data$topic=ifelse(data$topic=="['talk.politics.guns']","guns",data$topic)
data$topic=ifelse(data$topic=="talk.politics.guns","guns",data$topic)

data$topic=ifelse(data$topic=="['talk.politics.mideast']","mideast",data$topic
)
data$topic=ifelse(data$topic=="talk.politics.mideast","mideast",data$topic)

data$topic=ifelse(data$topic=="['talk.politics.misc']","politics",data$topic)
data$topic=ifelse(data$topic=="talk.politics.misc","politics",data$topic)

data$topic=ifelse(data$topic=="['talk.religion.misc']","religion-misc",data$to
pic)
data$topic=ifelse(data$topic=="talk.religion.misc","religion-misc",data$topic)
```

In [7]:
```
data %>% group_by(topic) %>% count()
```

A grouped_df: 20 × 2

| topic | n |
|---|---|
| <chr> | <int> |
| atheism | 792 |
| autos | 986 |
| baseball | 987 |
| christianity | 997 |
| comp-windows | 985 |
| compgraphics | 968 |
| crypt | 990 |
| electronics | 978 |
| guns | 910 |
| hocey | 993 |
| ibm-pc-hardware | 977 |
| mac-hardware | 948 |
| medicin | 987 |
| mideast | 940 |
| misc-forsale | 971 |
| motorcycles | 996 |
| politics | 775 |
| religion-misc | 625 |
| space | 984 |
| windows-x | 984 |

## GLOVE

GloVe stands for Global Vector for Word Representation, and is an unsupervised learning algorithm, which we use for obtaining vector representations. The model rests on a simple idea that ratios of word-word co-occurrence probabilities have the potential for encoding some form of meaning which can be encoded as vector differences.

```
In [0]: data_coor <- data %>% corpus(docid_field = "topic", text_field = "text")
```

We tokenize the model, where punct and symbols are removed.

```
In [0]: data_toks <- tokens(data_coor, what = "word") %>%
          tokens_tolower() %>%
          tokens(remove_punct = TRUE,
                 remove_symbols = TRUE)
```

```
In [10]: feats <- dfm(data_coor, verbose = TRUE) %>%
           dfm_trim(min_termfreq = 5) %>%
           featnames()

         data_fcm <- fcm(data_coor,
                          context = "window",
                          count = "weighted",
                          weights = 1 / (1:5),
                          tri = TRUE)
```

```
Creating a dfm from a corpus input...
   ... lowercasing
   ... found 18,773 documents, 159,504 features
   ... created a 18,773 x 159,504 sparse dfm
   ... complete.
Elapsed time: 8.09 seconds.
```

When we create the document - feuture matrix and a feature-co-occurence matrix we can use it to train GloVe embeddings.

In [11]:
```
glove <- GlobalVectors$new(word_vectors_size = 50, vocabulary = featnames(data
_fcm), x_max = 10)
shakes_wv_main = glove$fit_transform(data_fcm, n_iter = 10, convergence_tol =
0.001)
```

```
INFO [2019-11-29 20:19:04] 2019-11-29 20:19:04 - epoch 1, expected cost 0.106
1
INFO [2019-11-29 20:19:08] 2019-11-29 20:19:08 - epoch 2, expected cost 0.075
0
INFO [2019-11-29 20:19:12] 2019-11-29 20:19:12 - epoch 3, expected cost 0.063
6
INFO [2019-11-29 20:19:17] 2019-11-29 20:19:17 - epoch 4, expected cost 0.056
5
INFO [2019-11-29 20:19:21] 2019-11-29 20:19:21 - epoch 5, expected cost 0.051
6
INFO [2019-11-29 20:19:25] 2019-11-29 20:19:25 - epoch 6, expected cost 0.048
0
INFO [2019-11-29 20:19:29] 2019-11-29 20:19:29 - epoch 7, expected cost 0.045
2
INFO [2019-11-29 20:19:34] 2019-11-29 20:19:34 - epoch 8, expected cost 0.042
9
INFO [2019-11-29 20:19:38] 2019-11-29 20:19:38 - epoch 9, expected cost 0.041
0
INFO [2019-11-29 20:19:42] 2019-11-29 20:19:42 - epoch 10, expected cost 0.03
93
```

Now we create the word vectors based on the GloVe model. We want to see which words are correlated to the different topics. The tree topics we look at are 'baseball', 'christianity' and 'atheims'.

In [0]:
```
data_topi = glove$components
topi_word_vectors = shakes_wv_main + t(data_topi)
```

In [13]:
```
baseball = topi_word_vectors["baseball", , drop = F]
atheism = topi_word_vectors["atheism", , drop = F]

# ham = shakes_word_vectors["hamlet", , drop = F]
cos_sim_bas = sim2(x = topi_word_vectors, y = baseball, method = "cosine", nor
m = "l2")
cos_sim_ath = sim2(x = topi_word_vectors, y = atheism, method = "cosine", norm
= "l2")

# head(sort(cos_sim_rom[,1], decreasing = T), 10)
head(sort(cos_sim_bas[,1], decreasing = T), 5)
head(sort(cos_sim_ath [,1],decreasing = T),5)
```

```
baseball   1
  hockey   0.850228735725524
 players   0.788695876617732
    team   0.746814678143087
    game   0.731861223445283

 atheism   1
    weak   0.650023959059722
  strong   0.627936921216048
   faith   0.597056542506727
     Its   0.593246026010867
```

The 5 most relevant words with the greatest correlation can be seen here. For baseball, it is seen that the word 'hokey' has the greatest correlation and then 'players'. For the subject of Atheism, the word 'strong' is the word with the highest relation. It is coded so that the top 5 words with the greatest correlation in a decreasing approach are displayed. This makes some sense in relation to the topic and what words are correlated with it.

In [14]:
```
data_word_vectors <- fit_transform(data_fcm, glove, n_iter = 10)
```

```
INFO [2019-11-29 20:19:48] 2019-11-29 20:19:48 - epoch 1, expected cost 0.046
0
INFO [2019-11-29 20:19:53] 2019-11-29 20:19:53 - epoch 2, expected cost 0.039
4
INFO [2019-11-29 20:19:57] 2019-11-29 20:19:57 - epoch 3, expected cost 0.036
9
INFO [2019-11-29 20:20:01] 2019-11-29 20:20:01 - epoch 4, expected cost 0.035
2
INFO [2019-11-29 20:20:05] 2019-11-29 20:20:05 - epoch 5, expected cost 0.033
8
INFO [2019-11-29 20:20:10] 2019-11-29 20:20:10 - epoch 6, expected cost 0.032
7
INFO [2019-11-29 20:20:14] 2019-11-29 20:20:14 - epoch 7, expected cost 0.031
8
INFO [2019-11-29 20:20:18] 2019-11-29 20:20:18 - epoch 8, expected cost 0.031
0
INFO [2019-11-29 20:20:22] 2019-11-29 20:20:22 - epoch 9, expected cost 0.030
3
INFO [2019-11-29 20:20:27] 2019-11-29 20:20:27 - epoch 10, expected cost 0.02
97
```

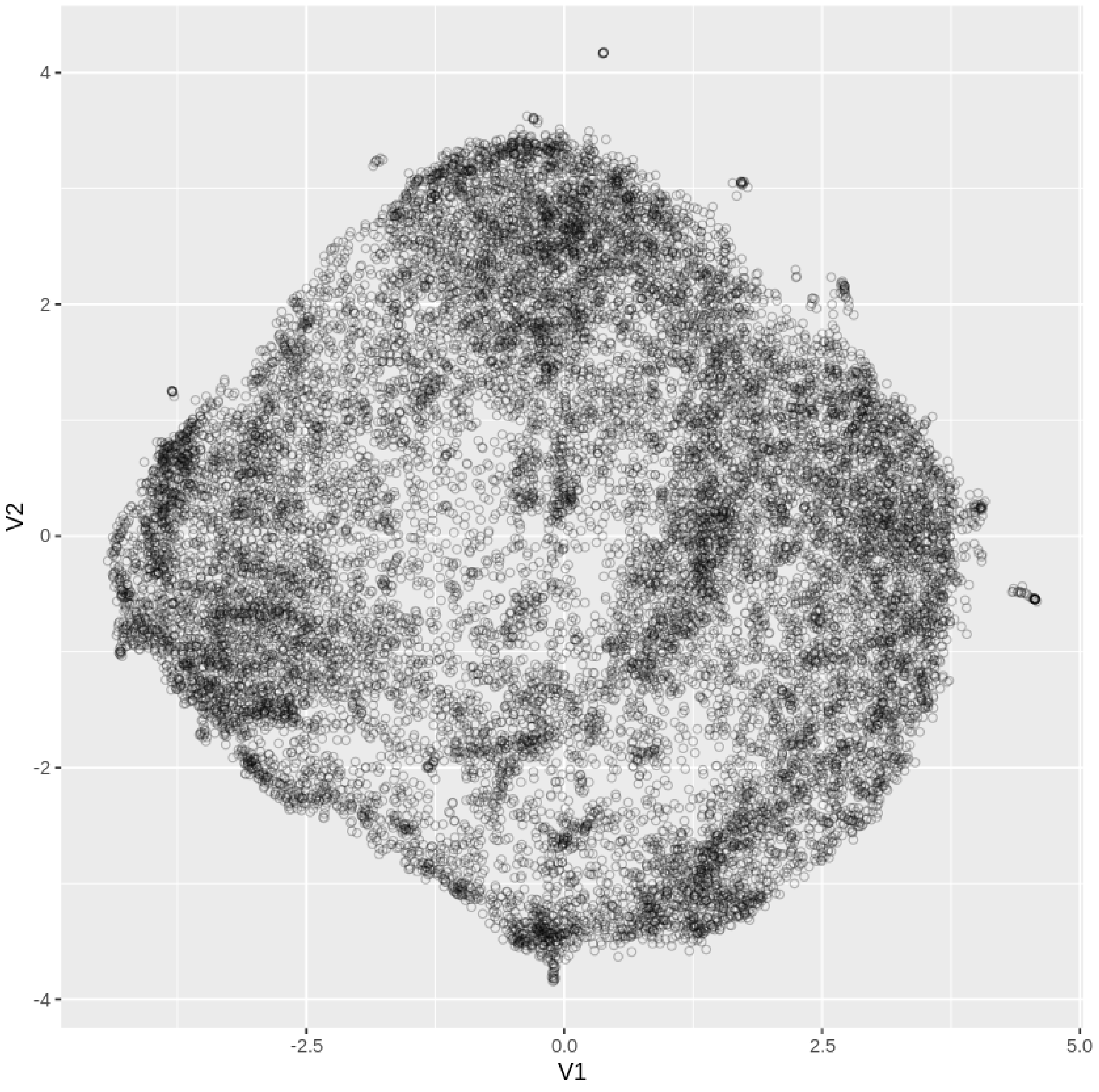
In [0]:
```
data_word_vectors %<>% as.data.frame() %>%
  rownames_to_column(var = "word") %>%
  as_tibble()

#out of this corpus we now create a tidy represation.
data_tidy2 <- data_toks %>%
  dfm() %>%
  tidy()

#and join them with the word vectors.
data_vectors <- data_tidy2 %>%
  inner_join(data_word_vectors, by = c("term" = "word"))


data_vectors %<>%
  select(-term, -count) %>%
  group_by(document) %>%
  summarise_all(mean)
```

In [16]:
```
data_vectors_umap <- umap(data_vectors %>% column_to_rownames("document"),
                       n_neighbors = 15,
                       metric = "cosine",
                       min_dist = 0.01,
                       scale = TRUE,
                       verbose = TRUE) %>%
  as.data.frame()
```

```
20:20:42 UMAP embedding parameters a = 1.896 b = 0.8006
20:20:42 Read 18773 rows and found 50 numeric columns
20:20:42 Scaling to zero mean and unit variance
20:20:42 Kept 50 non-zero-variance columns
20:20:42 Using Annoy for neighbor search, n_neighbors = 15
20:20:43 Building Annoy index with metric = cosine, n_trees = 50
0%   10   20   30   40   50   60   70   80   90   100%
[----|----|----|----|----|----|----|----|----|----|
**************************************************|
20:20:47 Writing NN index file to temp file /tmp/RtmpO2b0Xv/file8032262194
20:20:47 Searching Annoy index using 1 thread, search_k = 1500
20:20:52 Annoy recall = 100%
20:20:53 Commencing smooth kNN distance calibration using 1 thread
20:20:53 Initializing from normalized Laplacian + noise
20:20:54 Commencing optimization for 200 epochs, with 441142 positive edges
20:21:11 Optimization finished
```

We use the packages uwot to make the visualization.

In [17]:
```
data_vectors_umap %>%
  ggplot(aes(x = V1, y = V2)) +
  geom_point(shape = 21, alpha = 0.25)
```



## NETWORK

Network analysis is another way to gain insight into how the text is structured. We have previously used unnest_tokens as individual words, we now use n_grams which symbolize the order of words. We take a look at how often the word X is followed by the word Y and then build a model to see the relationship between them. When we put n = 2 we examine pairs of each other which are also called "Bigrams"

We create the bigrams by unnest_tokens.

In [0]:
```
bigramss <- data %>%
  unnest_tokens(bigram, text, token = "ngrams", n=2)
```

In [19]:
```
bigramss$bigram[1:2]
bigramss %>% head()
```

'from csfed' 'csfed uxa.ecn.bgu.edu'

A tibble: 6 × 4

| X1 | text_id | topic | bigram |
|---|---|---|---|
| <dbl> | <dbl> | <chr> | <chr> |
| 1 | 1366 | atheism | from csfed |
| 1 | 1366 | atheism | csfed uxa.ecn.bgu.edu |
| 1 | 1366 | atheism | uxa.ecn.bgu.edu frank |
| 1 | 1366 | atheism | frank doss |
| 1 | 1366 | atheism | doss subject |
| 1 | 1366 | atheism | subject re |

As we can see, the words overlap. The first token is "from csfed" and the second is "csfed uxa.ecn.bgu.edu". In the next step are the most common bigrams.

In [20]:
```
bigramss %>%
  count(bigram, sort = TRUE) %>% head()
```

A tibble: 6 × 2

| bigram | n |
|---|---|
| <chr> | <int> |
| of the | 22150 |
| in the | 15811 |
| subject re | 12423 |
| in article | 10002 |
| ax ax | 8806 |
| to the | 8806 |

The most common bigrams are words like "of the", "in the" and so on. These are very common and disinterested words for analysis. These words are cal "stop-words". We use now 'separate' from the packages tidyr to split the column into two column, "word" and "word2" and remove them if either is a stop-word.

```
In [21]: #
bigrams_separated <- bigramss %>%
  separate(bigram, c("word1", "word2"), sep = " ")

bigrams_filtered <- bigrams_separated %>%
  filter(!word1 %in% stop_words$word) %>%
  filter(!word2 %in% stop_words$word)

# new bigram counts:
bigram_counts <- bigrams_filtered %>%
  count(word1, word2, sort = TRUE)

bigram_counts %>% head()
```

A tibble: 6 × 3

| word1 | word2 | n |
|---|---|---|
| <chr> | <chr> | <int> |
| ax | ax | 8806 |
| nntp | posting | 8055 |
| posting | host | 8042 |
| organization | university | 2831 |
| distribution | world | 1687 |
| usa | lines | 1320 |

When the stop words are removed we now see that the common bigrams are now "ax ax", "posting host", "organization university" etc. We now want to combine two columns into a single column and this is done using the unite function from the packages tidyr. This function is the reverse of what we used earlier, 'separate'. Once we have seen how the words correlate with each other, it is interesting to visualize the relationship between the words.

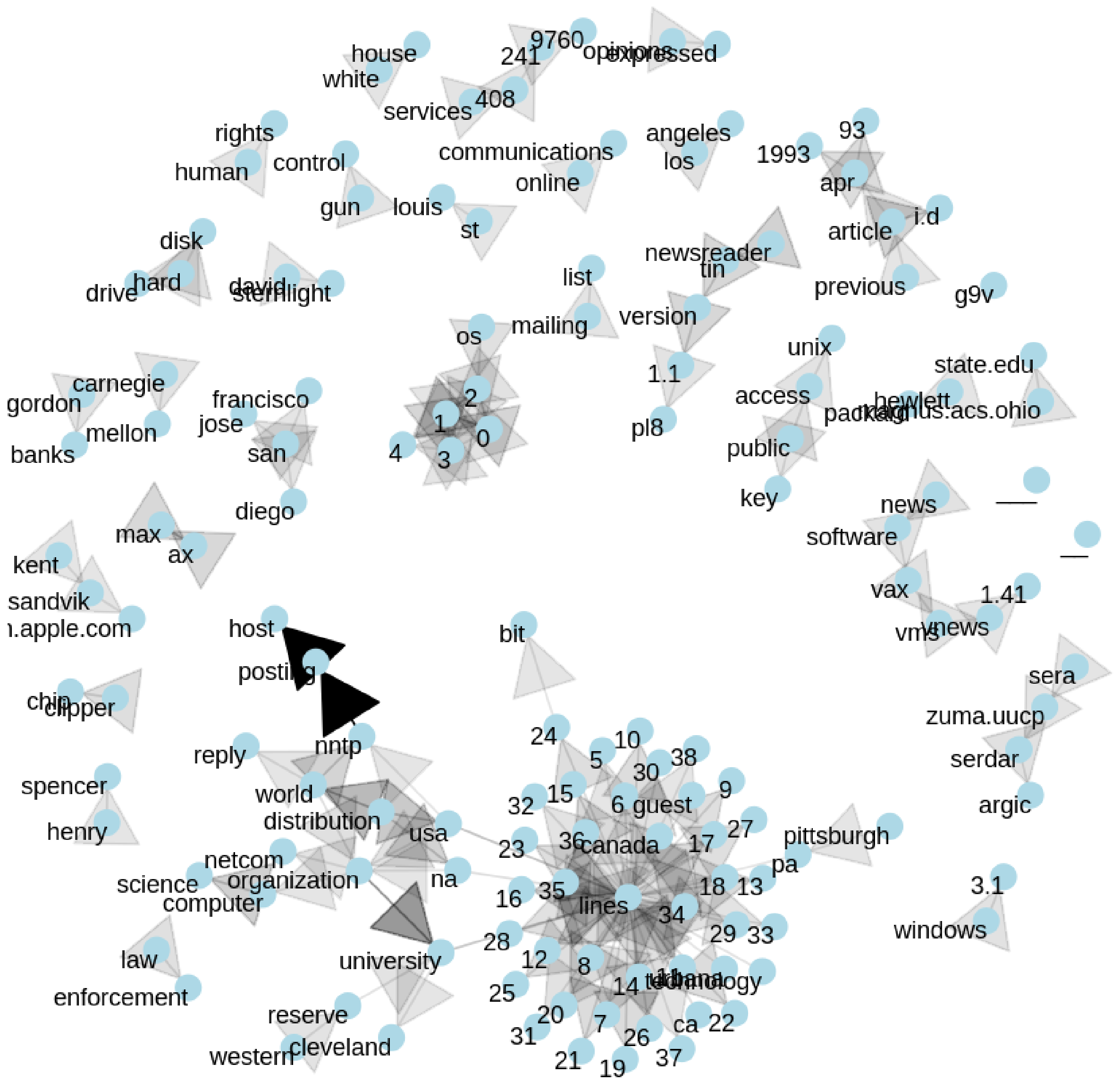
In [22]:
```
bigrams_united <- bigrams_filtered %>%
  unite(bigram, word1, word2, sep = " ")

bigrams_united %>% head()
```

A tibble: 6 × 4

| X1 | text_id | topic | bigram |
|---|---|---|---|
| <dbl> | <dbl> | <chr> | <chr> |
| 1 | 1366 | atheism | csfed uxa.ecn.bgu.edu |
| 1 | 1366 | atheism | uxa.ecn.bgu.edu frank |
| 1 | 1366 | atheism | frank doss |
| 1 | 1366 | atheism | doss subject |
| 1 | 1366 | atheism | theories organization |
| 1 | 1366 | atheism | organization educational |

The words are being arranged in a network. A graph can be constructed from a tidy object since it has three variables. From are the column where the node an edge is coming from, 'to' are the node and edge is going towards and 'weight' is a numeric value associated with each edge. The alpha value is plotted as the weight, so if the bigrams occur often then the colour would be darkere.

In [23]:
```
set.seed(2016)
  bigram_graph <- bigram_counts %>%
  filter(n > 200) %>%  #The occurence of the bigram is more than 15.
  graph_from_data_frame()

a <- grid::arrow(type = "closed", length = unit(.40, "inches"))

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                arrow = a, end_cap = circle(.07, 'inches')) +
  geom_node_point(color = "lightblue", size = 5) +
  geom_node_text(aes(label = name), vjust = 1, hjust = 1) +
  theme_void()
```



The plot shows how the words have a connection in the newsgroup. The arrows from one word to another indicate that there is a connection. As the words are lemmatized then it will not be grammatically correct, but it still makes sense to look at the connection. There are big and small connections, for example, are law -> enforcement, computer -> science and white -> house small connections, where san -> Diego -> francisco -> jose is a larger connection with several words combined to each other. At the same time, a much larger cluster is seen around the word 'lines' which is the keyword for all the numbers pointing towards each other. This makes sense since numbers are similar and therefore they are connected to each other.

In [24]:
```
bigram_tf_idf <- bigrams_united %>%
  count(topic, bigram) %>%
  bind_tf_idf(bigram, topic,n) %>%
  arrange(desc(tf_idf))

bigram_tf_idf %>% head()
```

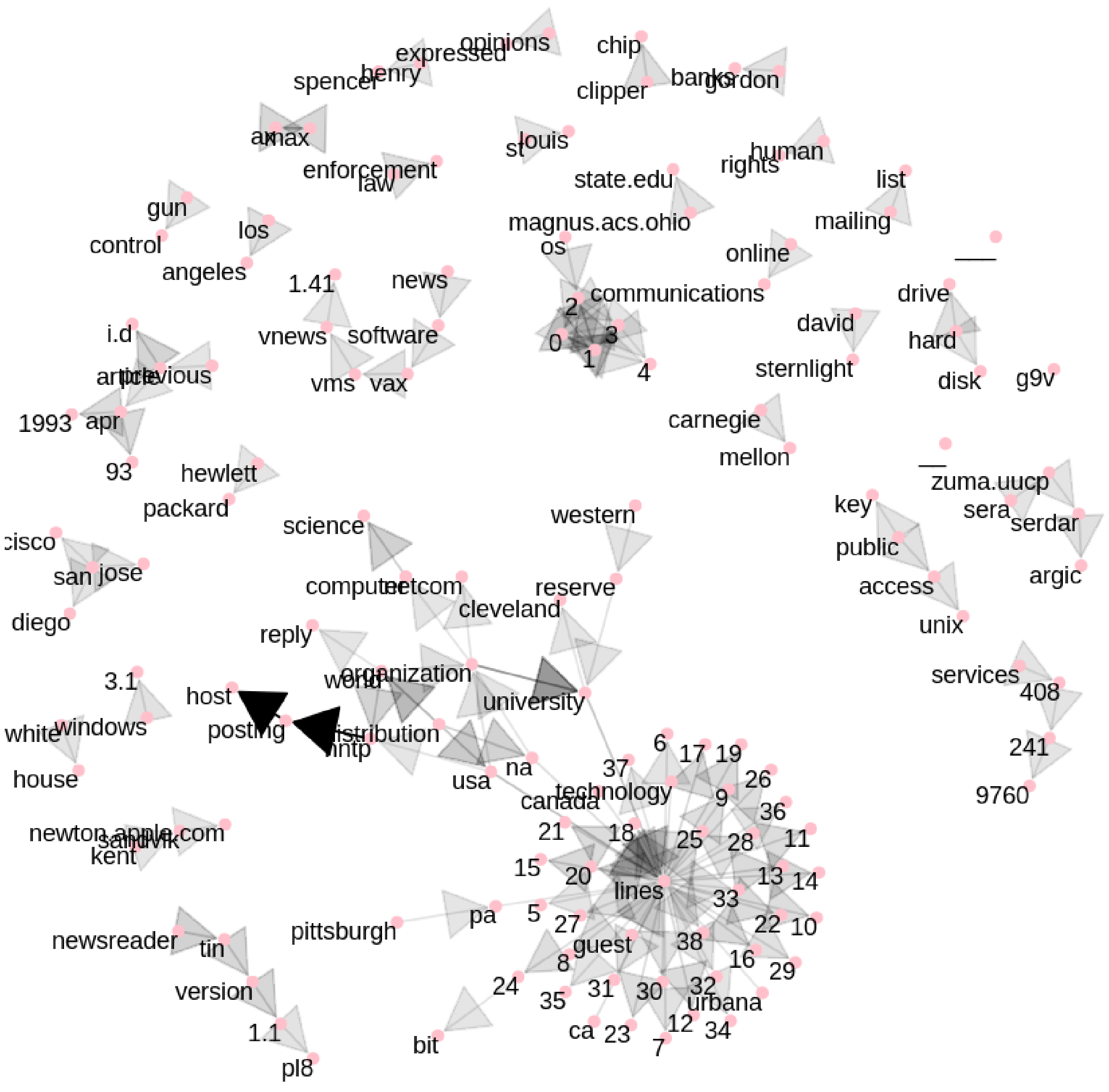
A tibble: 6 × 6

| topic | bigram | n | tf | idf | tf_idf |
|---|---|---|---|---|---|
| &lt;chr&gt; | &lt;chr&gt; | &lt;int&gt; | &lt;dbl&gt; | &lt;dbl&gt; | &lt;dbl&gt; |
| comp-windows | ax ax | 8806 | 0.113189116 | 2.995732 | 0.33908429 |
| comp-windows | max ax | 680 | 0.008740472 | 2.995732 | 0.02618411 |
| comp-windows | ax max | 678 | 0.008714765 | 2.995732 | 0.02610710 |
| mideast | serdar argic | 372 | 0.004354238 | 2.995732 | 0.01304413 |
| crypt | clipper chip | 450 | 0.006442653 | 1.897120 | 0.01222249 |
| comp-windows | g9v g9v | 281 | 0.003611872 | 2.995732 | 0.01082020 |

VIsualizing

--------------------------------------------------------

In [25]:
```
set.seed(123)
bigram_graph <- bigram_counts %>%
  filter(n > 200) %>%  #The occurence of the bigram is more than 15.
  graph_from_data_frame()

a<- grid::arrow(type = "closed",length = unit(.30,"inches"))

ggraph(bigram_graph, layout = "fr") +
  geom_edge_link(aes(edge_alpha = n), show.legend = FALSE,
                 arrow = a, end_cap = circle(.05,'inches'))+
  geom_node_point(color = "pink", size = 2) +
  geom_node_text(aes(label = name),vjust=1,hjust=1) +
  theme_void()
```



--------------------------------------

We import the data

```
In [0]: import pandas as pd
        import re
        import numpy as np
        import matplotlib.pyplot as plt
        import seaborn as sns
        import string
        import nltk
        import warnings

        from numpy.random import seed
        seed(1)

        from tensorflow import set_random_seed
        set_random_seed(2)
```

```
In [0]: data = pd.read_csv('https://www.dropbox.com/s/yvgkkt96fy0dp9k/File%20Name.csv?
        dl=1', low_memory=False)
```

```
In [0]: df = data
```

# Benchmark model

As a benchmark model we will be using simple NLP models, we will be using both a random forest model and a logistic clasifier model.

This model will be used to hold our neural network up against, to see how it performs. To do this we have to make a separate training and test split, run a tf-idf on both an then run it into the models.

## Random Forest

```
In [0]: from sklearn.feature_extraction.text import CountVectorizer
        bow_vectorizer = CountVectorizer(max_df=0.90, min_df=2, max_features=1000, sto
        p_words='english')
        # bag-of-words feature matrix
        bow = bow_vectorizer.fit_transform(df['text'])
```

```
In [0]: from sklearn.feature_extraction.text import TfidfVectorizer
        tfidf_vectorizer = TfidfVectorizer(max_df=0.90, min_df=2, max_features=1000, s
        top_words='english')
        # TF-IDF feature matrix
        tfidf = tfidf_vectorizer.fit_transform(df['text'])
        train_bow = bow[:31962,:]
```

In [0]:
```
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
```

In [0]:
```
X_train, X_test, y_train, y_test = train_test_split(train_bow, data['topic'],
random_state=42, test_size=0.3)
```

In [20]:
```
from sklearn.ensemble import RandomForestClassifier

RFmodel = RandomForestClassifier()

scores = cross_val_score(RFmodel, X_train, y_train, cv = 5)
print(scores)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245: Future
Warning: The default value of n_estimators will change from 10 in version 0.2
0 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245: Future
Warning: The default value of n_estimators will change from 10 in version 0.2
0 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245: Future
Warning: The default value of n_estimators will change from 10 in version 0.2
0 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245: Future
Warning: The default value of n_estimators will change from 10 in version 0.2
0 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245: Future
Warning: The default value of n_estimators will change from 10 in version 0.2
0 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

[0.66034156 0.65261959 0.65766451 0.66984369 0.66870229]
```

In [21]:
```
from sklearn.metrics import classification_report
RFmodel.fit(X_train, y_train)

y_pred_RF = RFmodel.predict(X_test)

print(classification_report(y_test, y_pred_RF))
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/ensemble/forest.py:245: Future
Warning: The default value of n_estimators will change from 10 in version 0.2
0 to 100 in 0.22.
  "10 in version 0.20 to 100 in 0.22.", FutureWarning)

                 precision    recall  f1-score   support

        atheism       0.62      0.67      0.65       225
          autos       0.59      0.74      0.66       277
       baseball       0.58      0.73      0.65       280
   christianity       0.68      0.85      0.75       304
   comp-windows       0.61      0.72      0.66       307
   compgraphics       0.44      0.53      0.48       276
          crypt       0.77      0.79      0.78       270
    electronics       0.44      0.43      0.43       287
           guns       0.69      0.76      0.72       275
          hocey       0.81      0.82      0.82       319
ibm-pc-hardware       0.53      0.44      0.48       309
   mac-hardware       0.68      0.60      0.64       305
        medicin       0.65      0.61      0.63       330
        mideast       0.89      0.83      0.86       285
   misc-forsale       0.76      0.74      0.75       293
    motorcycles       0.84      0.75      0.79       298
       politics       0.74      0.54      0.62       231
  religion-misc       0.66      0.34      0.45       188
          space       0.76      0.70      0.73       289
      windows-x       0.67      0.61      0.63       284

       accuracy                           0.67      5632
      macro avg       0.67      0.66      0.66      5632
   weighted avg       0.67      0.67      0.66      5632
```

The random forest model gives some nice results, getting a 67% accuracy with it having the most difficulty predicting the categories to do with elektronics and the easyest predicting the sports related categories.

## Logistic Regression

Now we will go on to a logistic reggresion, where the random forest relies on a series of reggresion trees for prediction, the logistic regression relies on linear estimations for predictions.

In [0]:
```
modelLogisticRegression = LogisticRegression()
```

In [23]:
```
#n-fold cross-validation (splitting the 80% into 5 part, using 4 to train and
 1 to evaluate)
scores = cross_val_score(modelLogisticRegression, X_train, y_train, cv = 5)
print(scores)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a s
olver to silence this warning.
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:469:
FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify
the multi_class option to silence this warning.
  "this warning.", FutureWarning)

[0.72220114 0.72930904 0.72879422 0.7384674  0.74732824]
```

In [24]:
```
modelLogisticRegression = LogisticRegression()
modelLogisticRegression.fit(X_train, y_train)
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:432:
FutureWarning: Default solver will be changed to 'lbfgs' in 0.22. Specify a s
olver to silence this warning.
  FutureWarning)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/logistic.py:469:
FutureWarning: Default multi_class will be changed to 'auto' in 0.22. Specify
the multi_class option to silence this warning.
  "this warning.", FutureWarning)
```

Out[24]:
```
LogisticRegression(C=1.0, class_weight=None, dual=False, fit_intercept=True,
                   intercept_scaling=1, l1_ratio=None, max_iter=100,
                   multi_class='warn', n_jobs=None, penalty='l2',
                   random_state=None, solver='warn', tol=0.0001, verbose=0,
                   warm_start=False)
```

In [25]:
```
print(modelLogisticRegression.score(X_test, y_test))
```

```
0.7356178977272727
```

```
In [26]: from sklearn.metrics import classification_report
         y_pred_log = modelLogisticRegression.predict(X_test)

         print(classification_report(y_test, y_pred_log))
```

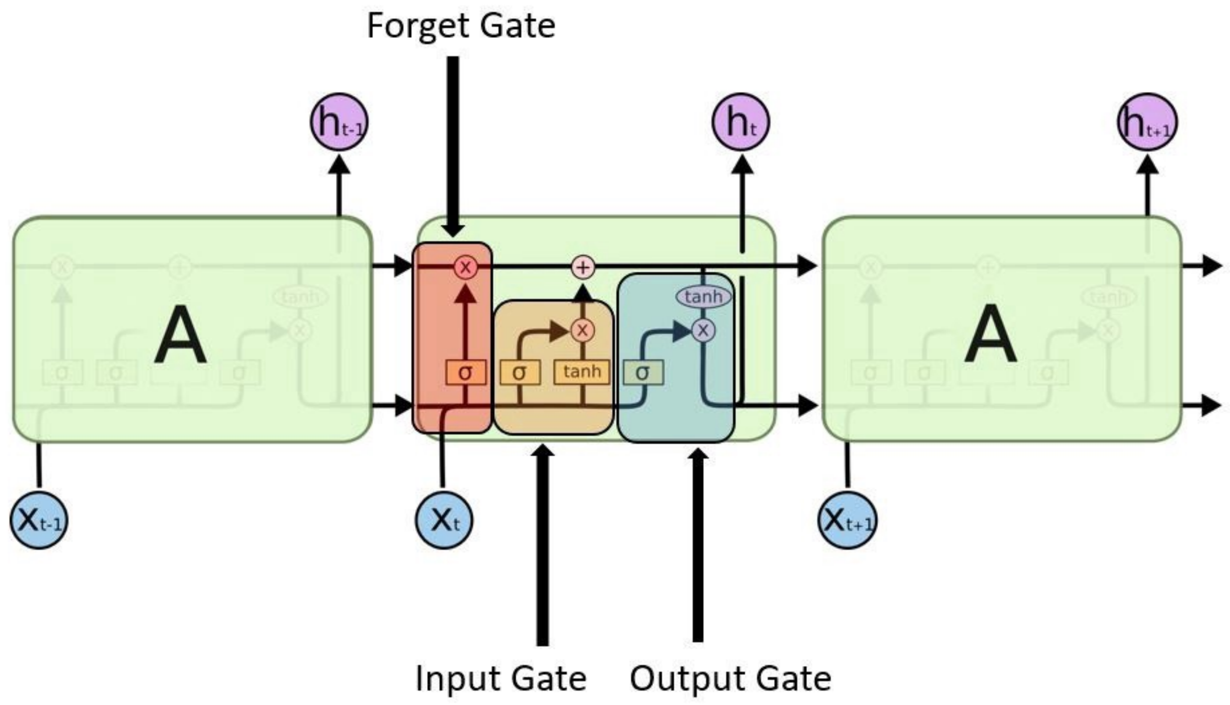
```
                 precision    recall  f1-score   support

        atheism       0.78      0.72      0.75       225
          autos       0.70      0.77      0.73       277
       baseball       0.77      0.81      0.79       280
   christianity       0.81      0.81      0.81       304
   comp-windows       0.70      0.73      0.71       307
   compgraphics       0.55      0.57      0.56       276
          crypt       0.83      0.82      0.83       270
    electronics       0.56      0.62      0.59       287
           guns       0.77      0.79      0.78       275
          hocey       0.87      0.87      0.87       319
ibm-pc-hardware       0.60      0.56      0.58       309
   mac-hardware       0.71      0.68      0.69       305
        medicin       0.72      0.73      0.73       330
        mideast       0.90      0.90      0.90       285
   misc-forsale       0.78      0.78      0.78       293
    motorcycles       0.82      0.78      0.80       298
       politics       0.74      0.67      0.70       231
  religion-misc       0.67      0.61      0.64       188
          space       0.77      0.77      0.77       289
      windows-x       0.66      0.66      0.66       284

       accuracy                           0.74      5632
      macro avg       0.74      0.73      0.73      5632
   weighted avg       0.74      0.74      0.74      5632
```

The logistic model seems to be performing better than the random forest, we get a 74% accuracy with the model performing better in most categories but its strong and weak categories are the same as in the random forest.

## LSTM

Long short-term memory (LSTM) is an artificial recurrent neural network (RNN) architecture used in the field of deep learning. Unlike standard feedforward neural networks, LSTM has feedback connections. It can not only process single data points (such as images), but also entire sequences of data (such as speech or video). For example, LSTM is commenly used for activities such as unsegmented, connected handwriting recognition, speech recognition and anomaly detection in network traffic or IDS's (intrusion detection systems).

In an LSTM network, three gates are present:



1. Input gate — This gate discovers which value from input should be used to modify the memory. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1.
2. Forget gate — discover what details to be discarded from the block. It is decided by the sigmoid function. it looks at the previous state(ht-1) and the content input(Xt) and outputs a number between 0(omit this)and 1(keep this)for each number in the cell state Ct−1.
3. Output gate — the input and the memory of the block is used to decide the output. Sigmoid function decides which values to let through 0,1. and tanh function gives weightage to the values which are passed deciding their level of importance ranging from-1 to 1 and multiplied with output of Sigmoid.

So to summarise the cell remembers values over arbitrary time intervals and the three gates regulate the flow of information into and out of the cell. LSTM models have been used in many different areas of machine learning.

For example in 2018 OpenAI developed bots able to humans in the game of Dota 2. OpenAI Five consists of five independent but coordinated neural networks. Each network is trained by a policy gradient method without supervising teacher and contains a single-layer, 1024-unit Long-Short-Term-Memory that sees the current game state and emits actions through several possible action heads. Also in 2019 Deepmind made a program called "AlphaStar" that used a deep LSTM core to play the complex game "Starcraft.

We import the data

In [4]:
```
from keras.preprocessing.text import Tokenizer
from keras.preprocessing.sequence import pad_sequences
```

```
Using TensorFlow backend.
```

We start by tokenizing our data.

- We start by tokenizing our data.
- Vectorize tweet text, by turning each text into either a sequence of integers or into a vector.
- Limit the data set to the top 50.000
- Set the max number of words to be 250.

In [5]:
```
# The maximum number of words to be used. (most frequent)
MAX_NB_WORDS = 50000
# Max number of words in each complaint.
MAX_SEQUENCE_LENGTH = 250
# This is fixed.
EMBEDDING_DIM = 100
tokenizer = Tokenizer(num_words=MAX_NB_WORDS, filters='!"#$%&()*+,-./:;<=>?@
[\]^_`{|}~', lower=True)
tokenizer.fit_on_texts(df['text'].values)
word_index = tokenizer.word_index
print('Found %s unique tokens.' % len(word_index))
```

```
Found 164987 unique tokens.
```

Next step is to truncate and pad the input sequences so that they are all in the same length for modeling.

In [6]:
```
X = tokenizer.texts_to_sequences(df['text'].values)
X = pad_sequences(X, maxlen=MAX_SEQUENCE_LENGTH)
print('Shape of data tensor:', X.shape)
```

```
Shape of data tensor: (18773, 250)
```

In [7]:
```
Y = pd.get_dummies(df['topic']).values
print('Shape of label tensor:', Y.shape)
```

```
Shape of label tensor: (18773, 20)
```

After tokenizing our data it's time to split our data into train and test.

```
In [0]: from sklearn.model_selection import train_test_split
```

```
In [9]: X_train, X_test, Y_train, Y_test = train_test_split(X,Y, test_size = 0.10, ran
        dom_state = 42)
        print(X_train.shape,Y_train.shape)
        print(X_test.shape,Y_test.shape)
```

```
(16895, 250) (16895, 20)
(1878, 250) (1878, 20)
```

It is now finally time to create our LSTM model.

```
In [0]: from keras.models import Sequential
        from keras.layers import Embedding
        from keras.layers import SpatialDropout1D
        from keras.layers import LSTM
        from keras.layers import Dense
        from keras.callbacks import ModelCheckpoint, EarlyStopping
```

```
In [0]: import keras
```

The LSTM model in this project is constructed as follows:

- The first layer is the embedded layer that uses 100 length vectors to represent each word.
- SpatialDropout1D performs variational dropout in NLP models.
- The next layer is the LSTM layer with 100 memory units.
- The output layer must create 20 output values, one for each outcome
- Activation function is softmax for multi-class classification.
- Because it is a multi-class classification problem, categorical_crossentropy is used as the loss function.
- 10 epochs have been used and gives the most accurate model without taking too much loading time. A more accurate model is perhaps possible with more epochs but it would increase the loading time significantly so this has been excluded and maintained at 10 epoch.

Various other inputs have been tested, but constructing our model with these speicific values has constructed the most accurate model, which the reader is free to test for themselves.

In [12]:
```
model = Sequential()
model.add(Embedding(MAX_NB_WORDS, EMBEDDING_DIM, input_length=X.shape[1]))
model.add(SpatialDropout1D(0.2))
model.add(LSTM(100, dropout=0.2, recurrent_dropout=0.2))
model.add(Dense(20, activation='softmax'))
model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['acc
uracy'])

epochs = 10
batch_size = 64

history = model.fit(X_train, Y_train, epochs=epochs, batch_size=batch_size,val
idation_split=0.1,callbacks=[EarlyStopping(monitor='val_loss', patience=3, min
_delta=0.0001)])
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:66: The name tf.get_default_graph is deprecated. Please
use tf.compat.v1.get_default_graph instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:541: The name tf.placeholder is deprecated. Please use
tf.compat.v1.placeholder instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:4432: The name tf.random_uniform is deprecated. Please
use tf.random.uniform instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:148: The name tf.placeholder_with_default is deprecate
d. Please use tf.compat.v1.placeholder_with_default instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:3733: calling dropout (from tensorflow.python.ops.nn_op
s) with keep_prob is deprecated and will be removed in a future version.
Instructions for updating:
Please use `rate` instead of `keep_prob`. Rate should be set to `rate = 1 - k
eep_prob`.
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/optimize
rs.py:793: The name tf.train.Optimizer is deprecated. Please use tf.compat.v
1.train.Optimizer instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:3576: The name tf.log is deprecated. Please use tf.mat
h.log instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow_cor
e/python/ops/math_grad.py:1424: where (from tensorflow.python.ops.array_ops)
is deprecated and will be removed in a future version.
Instructions for updating:
Use tf.where in 2.0, which has the same broadcast rule as np.where
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:1033: The name tf.assign_add is deprecated. Please use
tf.compat.v1.assign_add instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:1020: The name tf.assign is deprecated. Please use tf.c
ompat.v1.assign instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:3005: The name tf.Session is deprecated. Please use tf.
compat.v1.Session instead.

Train on 15205 samples, validate on 1690 samples
Epoch 1/10
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:190: The name tf.get_default_session is deprecated. Ple
ase use tf.compat.v1.get_default_session instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:197: The name tf.ConfigProto is deprecated. Please use
tf.compat.v1.ConfigProto instead.
```

```
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:207: The name tf.global_variables is deprecated. Please
use tf.compat.v1.global_variables instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:216: The name tf.is_variable_initialized is deprecated.
Please use tf.compat.v1.is_variable_initialized instead.

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/keras/backend/
tensorflow_backend.py:223: The name tf.variables_initializer is deprecated. P
lease use tf.compat.v1.variables_initializer instead.

15205/15205 [==============================] - 141s 9ms/step - loss: 2.7915 -
acc: 0.1237 - val_loss: 2.4287 - val_acc: 0.2207
Epoch 2/10
15205/15205 [==============================] - 133s 9ms/step - loss: 2.1354 -
acc: 0.3331 - val_loss: 1.8454 - val_acc: 0.4420
Epoch 3/10
15205/15205 [==============================] - 134s 9ms/step - loss: 1.4736 -
acc: 0.5240 - val_loss: 1.5684 - val_acc: 0.5041
Epoch 4/10
15205/15205 [==============================] - 134s 9ms/step - loss: 1.0488 -
acc: 0.6676 - val_loss: 1.4385 - val_acc: 0.5716
Epoch 5/10
15205/15205 [==============================] - 134s 9ms/step - loss: 0.7588 -
acc: 0.7628 - val_loss: 1.3721 - val_acc: 0.6030
Epoch 6/10
15205/15205 [==============================] - 133s 9ms/step - loss: 0.5713 -
acc: 0.8266 - val_loss: 1.4779 - val_acc: 0.5959
Epoch 7/10
15205/15205 [==============================] - 133s 9ms/step - loss: 0.4667 -
acc: 0.8606 - val_loss: 1.4281 - val_acc: 0.6343
Epoch 8/10
15205/15205 [==============================] - 134s 9ms/step - loss: 0.3217 -
acc: 0.9036 - val_loss: 1.4499 - val_acc: 0.6491
```

The model has been constructed. It is now time to interpret the results

In [13]:
```
print(model.summary())
```

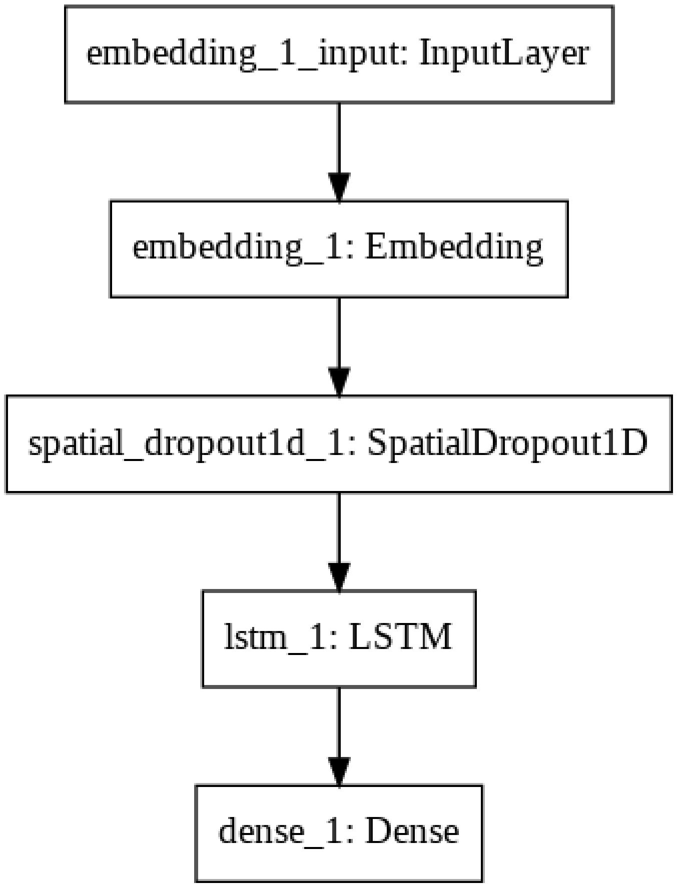
```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
embedding_1 (Embedding)      (None, 250, 100)          5000000   
_________________________________________________________________
spatial_dropout1d_1 (Spatial (None, 250, 100)          0         
_________________________________________________________________
lstm_1 (LSTM)                (None, 100)               80400     
_________________________________________________________________
dense_1 (Dense)              (None, 20)                2020      
=================================================================
Total params: 5,082,420
Trainable params: 5,082,420
Non-trainable params: 0
_________________________________________________________________
None
```

In [0]:
```
from keras.utils.vis_utils import plot_model
```

The Keras package has an included visualization package which helps explain to the reader how the model is constructed. This visualization will be presented below.

In [15]:
```
plot_model(model, to_file='model.png')
```

Out[15]:


embedding_1_input: InputLayer
embedding_1: Embedding
spatial_dropout1d_1: SpatialDropout1D
lstm_1: LSTM
dense_1: Dense

Now it's time for the true test. We are going to test the model on the actual test set.

In [24]:
```
accr = model.evaluate(X_test,Y_test)
print('Test set\n  Loss: {:0.3f}\n  Accuracy: {:0.3f}'.format(accr[0],accr[1
]))
```

```
1878/1878 [==============================] - 12s 6ms/step
Test set
  Loss: 1.564
  Accuracy: 0.619
```

We get an Accuracy of 61.9% which is quite amazing considering how many catogories there are and how little difference there is between them. The accuracy might seem low, but it is important to consider the model has to classify between 20 different variables which is quite problematic. Even a human would get massive errors if it had to classify a text between 20 different text topics. We will now further explore the model to see which topics are hard to classify.

In [0]:
```
from sklearn.metrics import classification_report
y_pred = model.predict(X_test)
```

In [0]:
```
y1 = np.argmax(y_pred, axis=1)
```

In [0]:
```
y2 = np.argmax(Y_test, axis=1)
```

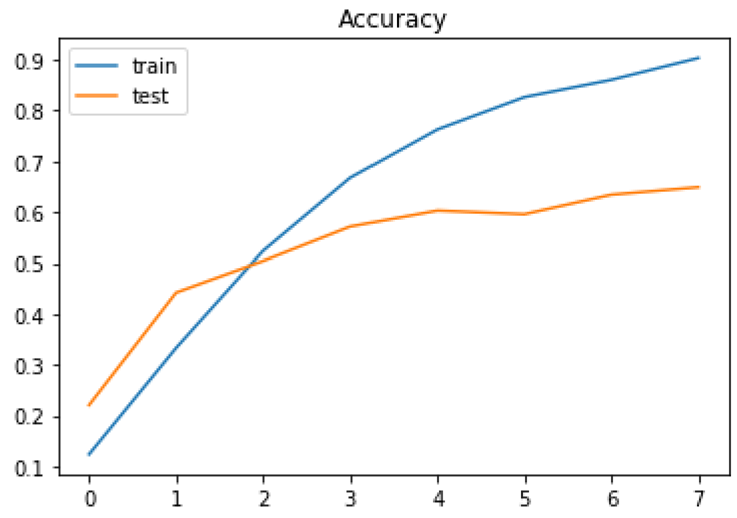
In [0]:
```
from sklearn.metrics import confusion_matrix
cm = confusion_matrix(y2,y1)
```

```
In [25]: print(classification_report(y2, y1, target_names=['Atheism', 'Autos', 'Basebal
         l', 'Christianity', 'Compgraphics', 'Comp-windows', 'Crypt', 'Electronics', 'G
         uns', 'Hockey', 'IBM-PC-hardware', 'Mac-hardware', 'Medicin', 'Mideast', 'Misc
         -forsale', 'Motorcycles', 'Politics', 'Religion-misc', 'Space', 'Windows-x',]
         ))
```

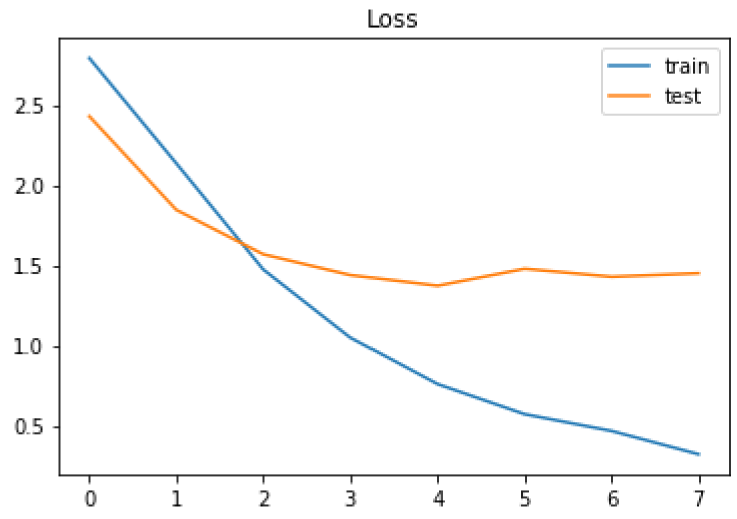
```
                 precision    recall  f1-score   support

        Atheism       0.64      0.74      0.69        58
          Autos       0.48      0.53      0.51        83
       Baseball       0.76      0.83      0.79       105
   Christianity       0.60      0.69      0.64        81
   Compgraphics       0.60      0.60      0.60       110
   Comp-windows       0.44      0.39      0.41        90
          Crypt       0.78      0.57      0.66       102
    Electronics       0.41      0.37      0.39        98
           Guns       0.63      0.65      0.64        88
         Hockey       0.83      0.82      0.83        91
IBM-PC-hardware       0.52      0.56      0.54       109
   Mac-hardware       0.58      0.57      0.58       110
        Medicin       0.62      0.62      0.62       108
        Mideast       0.60      0.82      0.69        87
   Misc-forsale       0.72      0.60      0.65        92
    Motorcycles       0.87      0.60      0.71        97
       Politics       0.49      0.61      0.54        82
  Religion-misc       0.46      0.43      0.44        75
          Space       0.69      0.62      0.66       101
      Windows-x       0.69      0.77      0.73       111

       accuracy                           0.62      1878
      macro avg       0.62      0.62      0.62      1878
   weighted avg       0.63      0.62      0.62      1878
```

We can observe from the above figure that the model can easier classify certain topics compared to others. For example the topic "Hockey" has a accuracy of 83%. The LSTM model experiences problems when having to classify Electronics, which can be explained through the other topics. There is so many topics related to electronics for example Windows X, MAC hardware, IBM-PC-hardware and compgraphics, which makes this topic and other topics hard to classify. IBM-PC-hardware and Mac-hardware accuracy is also quite low on 52% and 58%. Same for Comp windows on 44%, which indicates the model mixes the different topic with each other. Baseball is good example of a topic that is not related to many other topics. This is also where the model achieves very high accuracy. Same for Motorcycles and so on. Overall we can conclude the LSTM neural network is quite accurate, but more accurate for certain topics.

In [22]:
```
plt.title('Accuracy')
plt.plot(history.history['acc'], label='train')
plt.plot(history.history['val_acc'], label='test')
plt.legend()
plt.show();
```



In [23]:
```
plt.title('Loss')
plt.plot(history.history['loss'], label='train')
plt.plot(history.history['val_loss'], label='test')
plt.legend()
plt.show();
```



The plots suggest that the model has a little over fitting problem, more data may help, but more epochs will not help using the current data and will only increase only loading times. More data for all topics could help reduce this over fitting problem.

## Conclusion

For this conclusion we will be focusing primarily on our benchmark models and the neural network, the Glove model helps us with identifying words that are important in each topic. We find that electronics are connected as well as sport topics, and that some combinations of words are used in specific categories.

Looking at the 3 models, we see that the logistic and random forest models are outperforming our neural network. While not expected, this makes some sense as the classification models are based on a bag of word approach making the words everything and as there will be words used exclusively in each topic. The neural network takes sequences into a count, the sequence of words is often a personal thing and will therefore work well in identifying a person from writing but less in identifying a topic. Further though our model has been run for some time it still has room for improvements, both by adding epochs and more layers and units in each layer and therefore there are room for improvements.