A novel story from exploration analysis of toxic comments: train a Word2Vec model

A large part of text mining and NLP modeling use bag of words or bag of n-grams methods: they are simple to understand and have a good performance on text categorization and classification tasks.
But, the first step in text mining modelling (and also data science problems) should always the exploration analysis to discover all hidden (and not) patterns on our data.

In this vignette, let's briefly review some of the steps in a typical text analysis pipeline: from tokenization to cleaning and record linkage phase.
For the scope, we will use the Toxic Comments dataset (https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge) witch present a lot of Wikipedia comments. The challenge is to predict the toxicity grade of a text (toxic, severe_toxic, obscene ,threat ,insult ,identity_hate) using only the sentences and words.

The researcher usually begins by cleaning the text from special characters and common noises sentences. There is no magic and automatic recipe: the phase of exploration and cleaning is fundamental and can greatly improve our analysis models. Following this article, we will try to speak with you about R and H2O (https://www.h2o.ai). To setup the environment for the exploration analysis take a look at Docker and H2O to enter leaderboard in Kaggle competitions (http://www.mauropelucchi.com/blog/docker-and-h2o/). We also take a look at Word2Vec (follow this link for the full paper https://arxiv.org/pdf/1301.3781.pdf).

The full script is available at https://github.com/mauropelucchi/machine-learning-course/blob/master/text-mining/h2o_toxic_exploration.r

Disclaimer: the dataset from this Kaggle competition contains text that may be considered profane, vulgar, or offensive.

Toxic Comment Classification Challenge

Identify and classify toxic online comments

Discussing things you care about can be difficult. The threat of abuse and harassment online means that many people stop expressing themselves and give up on seeking different opinions. Platforms struggle to effectively facilitate conversations, leading many communities to limit or completely shut down user comments.

Several teams of data scientist are working on tools to help improve online conversation. One area of focus is the study of negative online behaviors, like toxic comments, in other words comments that are rude, disrespectful or otherwise likely to make someone leave a discussion. There are a lot of publicly available models served through API to predict toxicity of a text. But the current models still make errors, and they don't allow users to select which types of toxicity they're interested in finding.

In Kaggle competition, we're involved to build a multi-headed model that's capable of detecting different types of toxicity like threats, obscenity, insults, and identity-based hate.
We will be using a dataset of comments from Wikipedia's talk page edits. Each comment has been labeled by human raters for toxic behavior. The types of toxicity are:

- toxic
- severe_toxic
- obscene
- threat
- insult
- identity_hate

At this page, you can download the entire dataset https://www.kaggle.com/c/jigsaw-toxic-comment-classification-challenge/data:
- train.csv - the training set, contains comments with their binary labels;
- test.csv - the test set, you must predict the toxicity probabilities for these comments.

Stopwords

From Wikipedia, with stop words (https://en.wikipedia.org/wiki/Stop_words) are words which are filtered out before or after processing of natural language data (text). Though "stop words" usually refers to the most common words in a language, there is no single universal list of stop words used by all natural language processing tools, and indeed not all tools even use such a list. Some tools specifically avoid removing these stop words to support phrase search.
To clean, Toxic Comment Dataset from stop words we use a custom dictionary of stop words. You can download our dictionary from https://github.com/mauropelucchi/machine-learning-course/blob/master/text-mining/stopwords-en.txt. It refers to common stop words for English language.

Check the toxicity

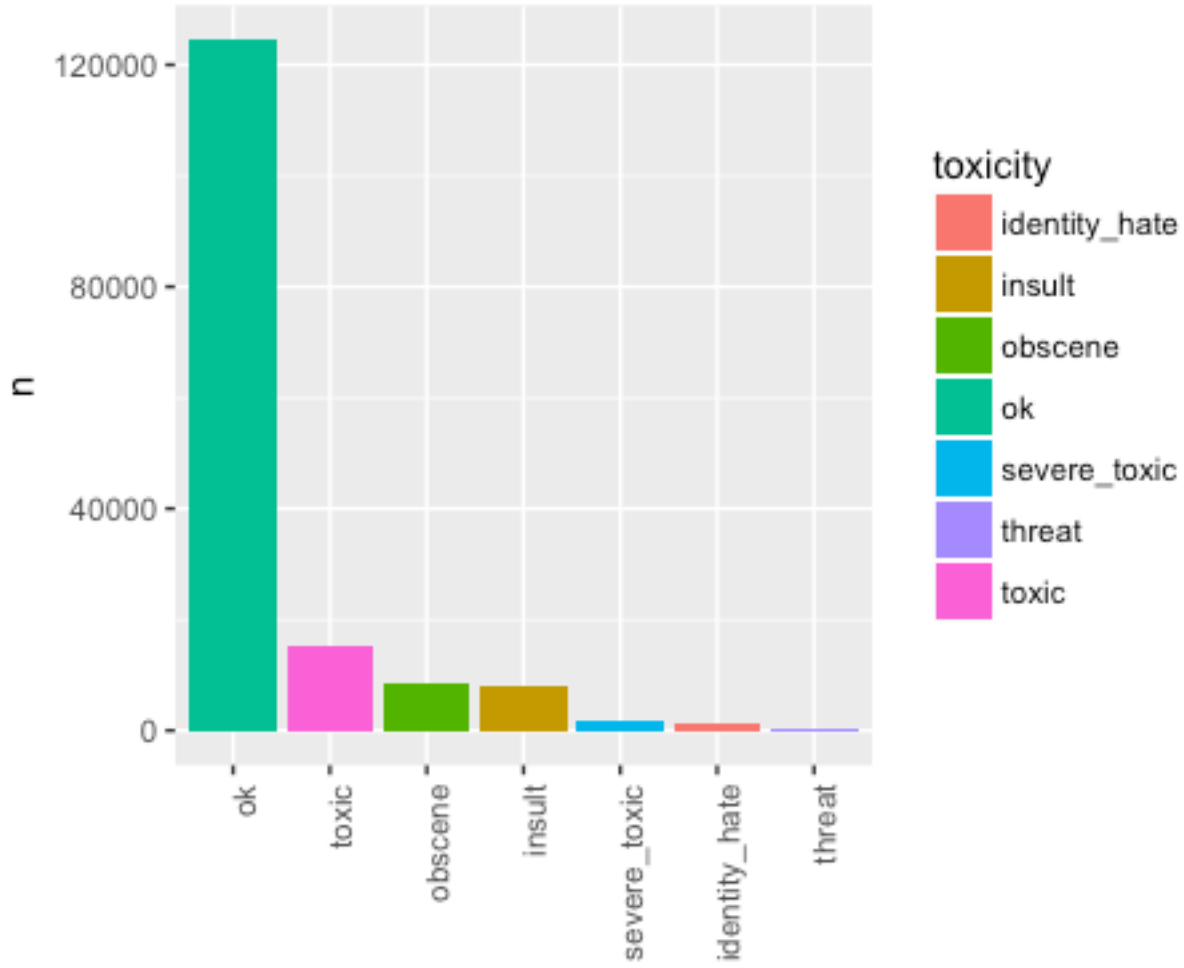
The first step is to check the frequency of target labels in Kaggle dataset.
We have to create a R script and load the CSV files and also the txt file with the list of our stop words.

```
library(data.table)
library(dplyr)
library(h2o)
library(stringr)
library(ngram)
packageVersion("h2o")

h2o.init(ip = "localhost", port = 4444, nthreads = -1, max_mem_size = "8g", strict_version_check = FALSE)

train <- fread("/Users/mauropelucchi/Desktop/Toxic_comments/train.csv", key=c("id"))
test <- fread("/Users/mauropelucchi/Desktop/Toxic_comments/test.csv", key=c("id"))
stopwords.en <- fread("/Users/mauropelucchi/Desktop/Toxic_comments/stopwords-en.txt")
```

```
train <- train %>% mutate(filter="train")
test <- test %>% mutate(filter="test")

all_comments <- train %>% bind_rows(test)
nrow(all_comments)
```

We have loaded 312,735 comments, but how are the toxicity distributed among the data? On train dataset, we can count the frequency for each target class.

```
## Count frequency for each class (on train)
nrow(train)
library(ggplot2)
library(tidyr)
names(tmp)
tmp <- train %>% summarise(toxic=sum(toxic),
                severe_toxic=sum(severe_toxic),
                obscene=sum(obscene) ,
                threat=sum(threat) ,
                insult=sum(insult) ,
                identity_hate=sum(identity_hate),
                total=n()) %>%
  mutate(ok=total-toxic-severe_toxic-obscene-threat-insult-identity_hate) %>%
  select(ok,toxic,obscene,insult,severe_toxic,identity_hate,threat)
# transpose all but the first column (name)
df.freq <- as.data.frame(t(tmp[,]))
colnames(df.freq ) <- t(tmp[0, ])
df.freq$toxicity <- factor(row.names(df.freq))
colnames(df.freq)[1] <- "n"

df.freq %>%
  ggplot(aes(x=reorder(toxicity,-n), y=n, fill=toxicity)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90, hjust=1), axis.title.x = element_blank())#
```



The dataset is imbalanced. Imbalanced data (all real dataset are imbalanced) typically refers to a problem with classification problems where the classes are not represented equally.
As you can see in this picture (from Nitin R. Patel), some models are sensible to rare events. In this case, you could use under-sampling and over-sampling techniques to balancing the dataset.

Comparison of Data Mining techniques Nitin R. Patel

	Multiple Linear Regression	Logistic Regression	Discriminant Analysis	Naïve Bayes	Neural Nets	Trees	k-Nearest Neighbors
Accuracy	M	M	M	HM	H	M	HM
Intepretability	H	H	M	H	L	H	L
Speed-Training	H	H	H	HM	L	HM	H
Speed-Deployment	H	H	H	H	H	HM	L
Effort in choice and transformation of indep.Vars.	HM	HM	HM	HM	L	L	ML
Effort to tune performance parameters	L	L	L	ML	H	ML	ML
Robustness to Outliers in indep vars	ML	ML	ML	ML	HM	H	HM
Robustness to irrelevant variables	H	H	HM	H	L	ML	L
Ease of handling of missing values	M	M	M	H	M	H	ML
Natural handling both categorical and continuous variables	H	H	ML	M	H	H	L

H: high, M:medium, L:low.

53

Significant words

Now you can explore the dataset and, for each toxicity grade, find the most common words. First we apply a standard text processing pipeline composed by the following steps:
- Removing punctuation and special charaters
- Lowercase
- Cleaning from words that contains number
- Removing words less than 2 char
- Removing stop words and rare words.

We will use a combination of two pipeline: the first has the scope to configure our custom stop words and rare words dictionary, the second apply the text processing to summarize each comment.

```
# Remove all special chars
all_comments <- all_comments %>% mutate(clean_text=str_replace_all(comment_text,
"[^[:alnum:]]", " "))
head(all_comments)
# Go to H2O
comments.hex <- as.h2o(select(all_comments, c("id", "clean_text", "filter", "toxic", "severe_toxic",
"obscene","threat" ,"insult" ,"identity_hate")), destination_frame = "all_comments.hex",
col.types=c("String"))
```

```
# Tokenize, lowercase, remove text with len less than 3 chars
tokenize <- function(text) {
  tokenized <- h2o.tokenize(text, "\\\\W+")
  tokenized.lower <- h2o.tolower(tokenized)
  # remove (less than 2 chars)
  tokenized.lengths <- h2o.nchar(tokenized.lower)
  tokenized.filtered <- tokenized.lower[is.na(tokenized.lengths) || tokenized.lengths > 2,]
  # remove words that contain numbers
  tokenized.words <- tokenized.filtered[h2o.grep("[0-9]", tokenized.filtered, invert = TRUE,
output.logical = TRUE),]
}


# Get a list of terms and create stop_words and rare_words lists
words.hex <- tokenize(comments.hex$clean_text)
h2o.head(words.hex)
words <- as.data.frame(words.hex)
words <- words %>% group_by(C1) %>% mutate(wc=n())
tmp <- words %>% tally(wc) %>% top_n(200)
head(tmp)
tmp <- words %>% tally(wc) %>% top_n(-200)
tmp
stop_words <- words %>% tally(wc) %>% top_n(200) %>% select(C1) %>% ungroup()
rare_words <- words %>% filter(wc < 10) %>% select(C1) %>% ungroup()
rm(words, tmp)
h2o.rm(words.hex)
#
stop_words_chr <- c(stop_words$C1) %>% union(stopwords.en$C1) %>% c('wikipedia','username')
rare_words_chr <- c(rare_words$C1)
```

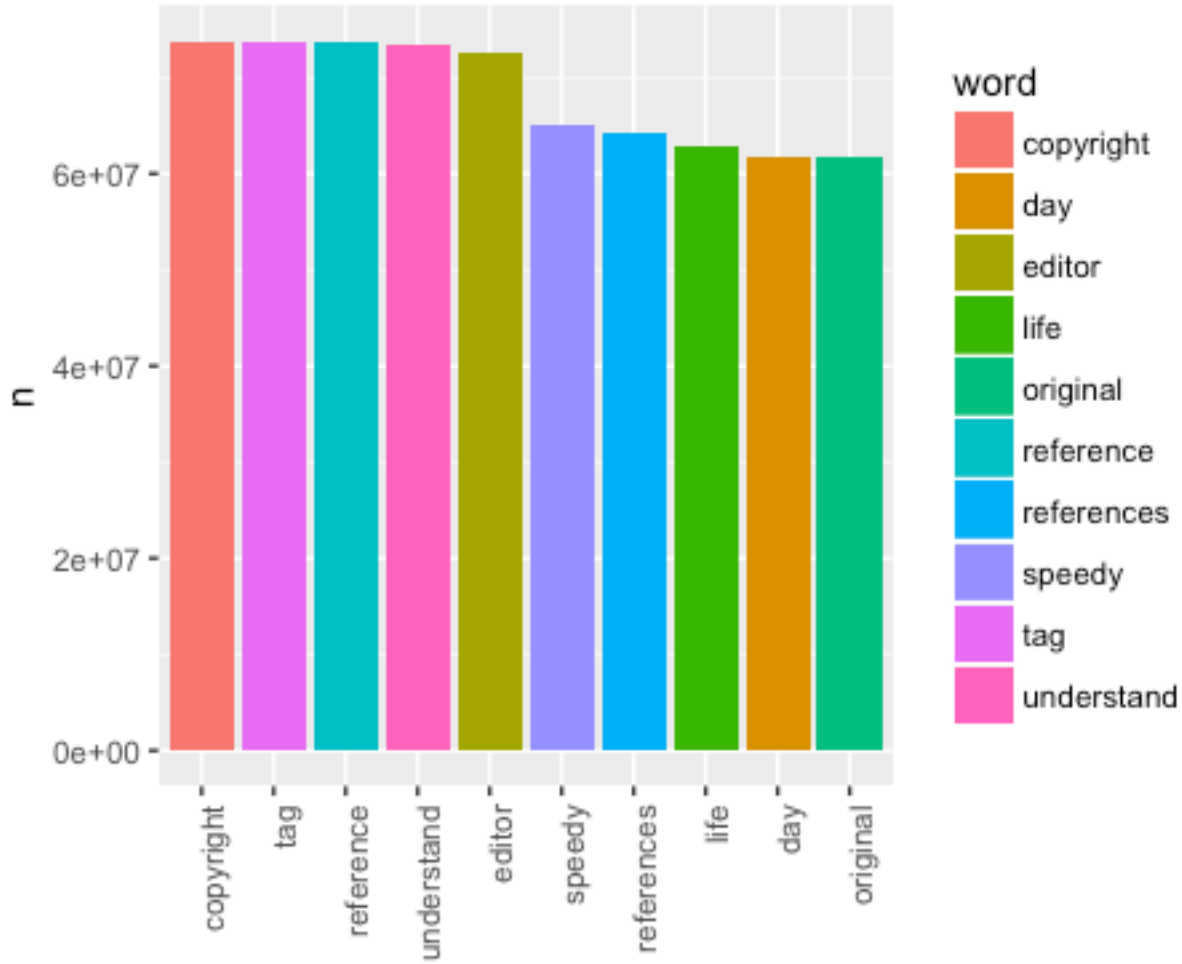
Our custom stop words and rare words dictionary is composed by about 473k entries.
Now we redefine our tokenize function to clean the text.

```
# Tokenize, lowercase, remove text with len less than 3 chars
clean_token <- function(text, STOP_WORDS, RARE_WORDS) {
  tokenized <- h2o.tokenize(text, "\\\\W+")
  tokenized.lower <- h2o.tolower(tokenized)
  # remove (less than 2 chars)
  tokenized.lengths <- h2o.nchar(tokenized.lower)
  tokenized.filtered <- tokenized.lower[is.na(tokenized.lengths) || tokenized.lengths > 2,]
  # remove words that contain numbers
  tokenized.words <- tokenized.filtered[h2o.grep("[0-9]", tokenized.filtered, invert = TRUE,
output.logical = TRUE),]
  tokenized.words <- tokenized.words[is.na(tokenized.words) || (! tokenized.words %in%
STOP_WORDS),]
  tokenized.words <- tokenized.words[is.na(tokenized.words) || (! tokenized.words %in%
RARE_WORDS),]
```

```
  tokenized.words[!is.na(tokenized.words), ]
}
```

Now we can explore the most common words used in the comments from Kaggle dataset.

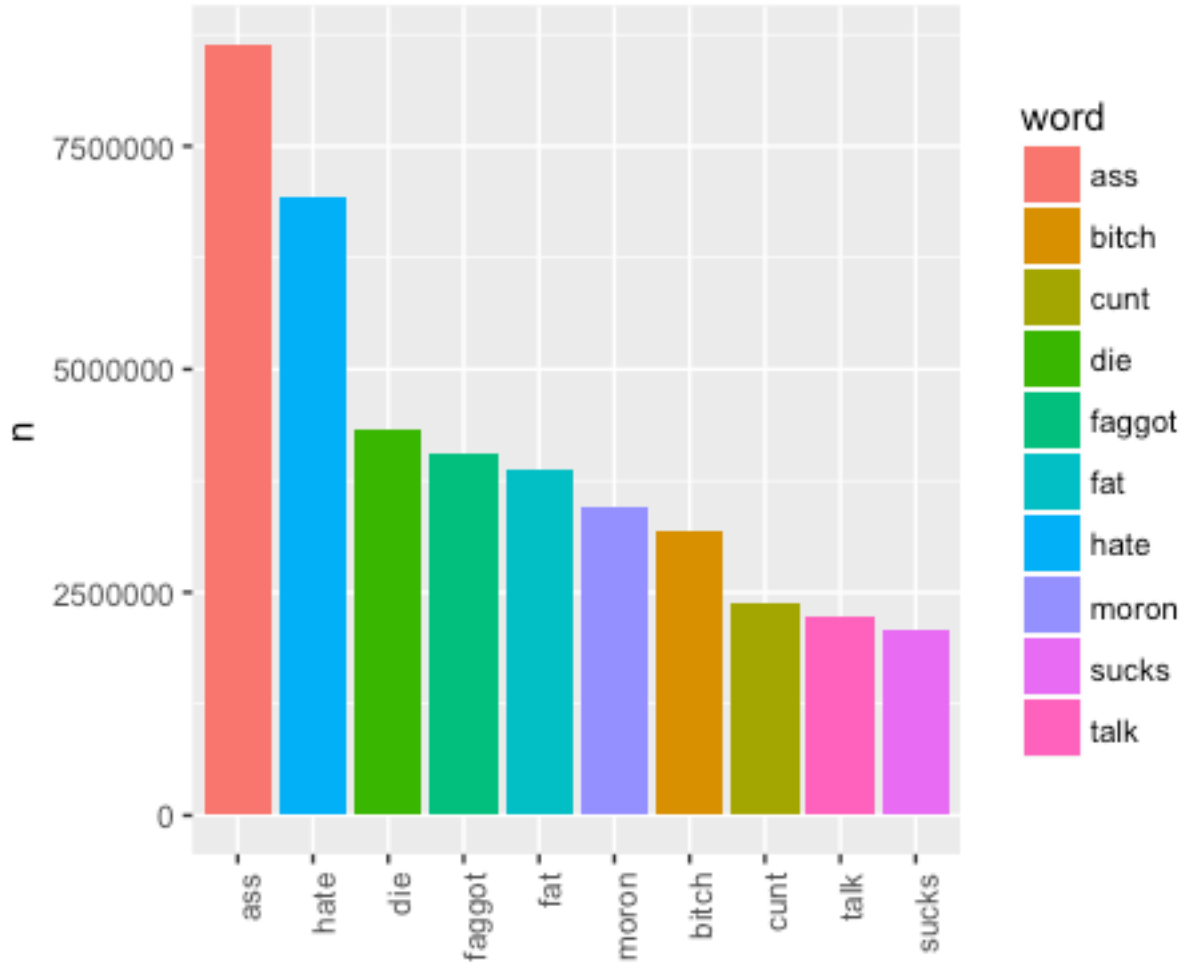
```
## Explore Common words
words.hex <- clean_token(comments.hex$clean_text, stop_words_chr, rare_words_chr)
h2o.head(words.hex)
nrow(words.hex) #6128533
words <- as.data.table(words.hex)
words <- words %>% group_by(C1) %>% mutate(wc=n())
top.words <- words %>% tally(wc) %>% top_n(10) %>% ungroup()
head(top.words, 10)
names(top.words)[1] <- "word"
names(top.words)[2] <- "n"


top.words %>%
  ggplot(aes(x=reorder(word,-n), y=n, fill=word)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90, hjust=1), axis.title.x = element_blank())#
```



Explore Toxic comment:

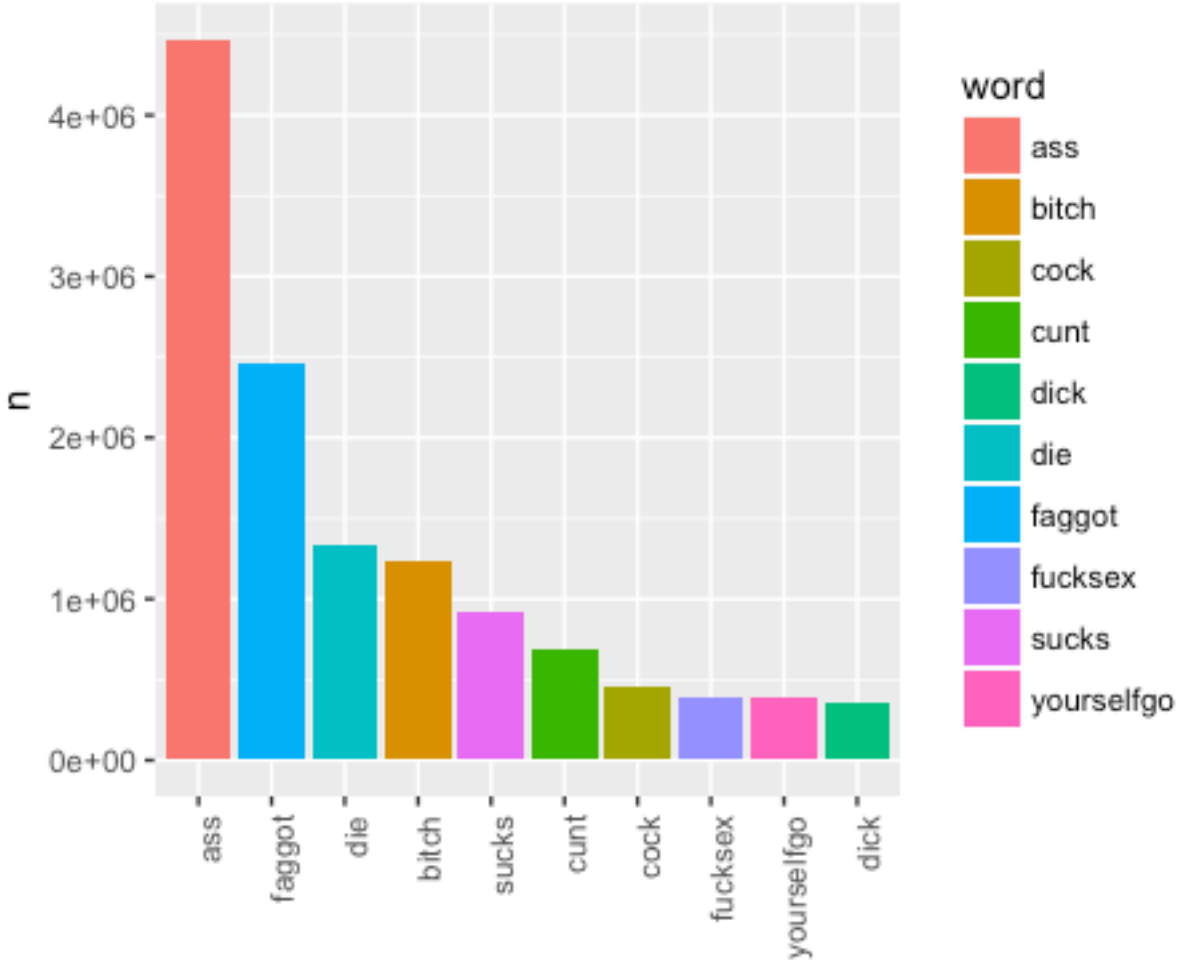
```
## Explore Toxic comment
toxic.set <- comments.hex[, "toxic"] == 1
toxic.hex <- comments.hex[toxic.set, ]
toxic.words.hex <- clean_token(toxic.hex$clean_text, stop_words_chr, rare_words_chr)
h2o.head(toxic.words.hex)
nrow(toxic.words.hex) #251819
toxic.words <- as.data.table(toxic.words.hex)
toxic.words <- toxic.words %>% group_by(C1) %>% mutate(wc=n())
top.toxic_words <- toxic.words %>% tally(wc) %>% top_n(10) %>% ungroup()
head(top.toxic_words, 10)
names(top.toxic_words)[1] <- "word"
names(top.toxic_words)[2] <- "n"
```

```
top.toxic_words %>%
  ggplot(aes(x=reorder(word,-n), y=n, fill=word)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90, hjust=1), axis.title.x = element_blank())#
```



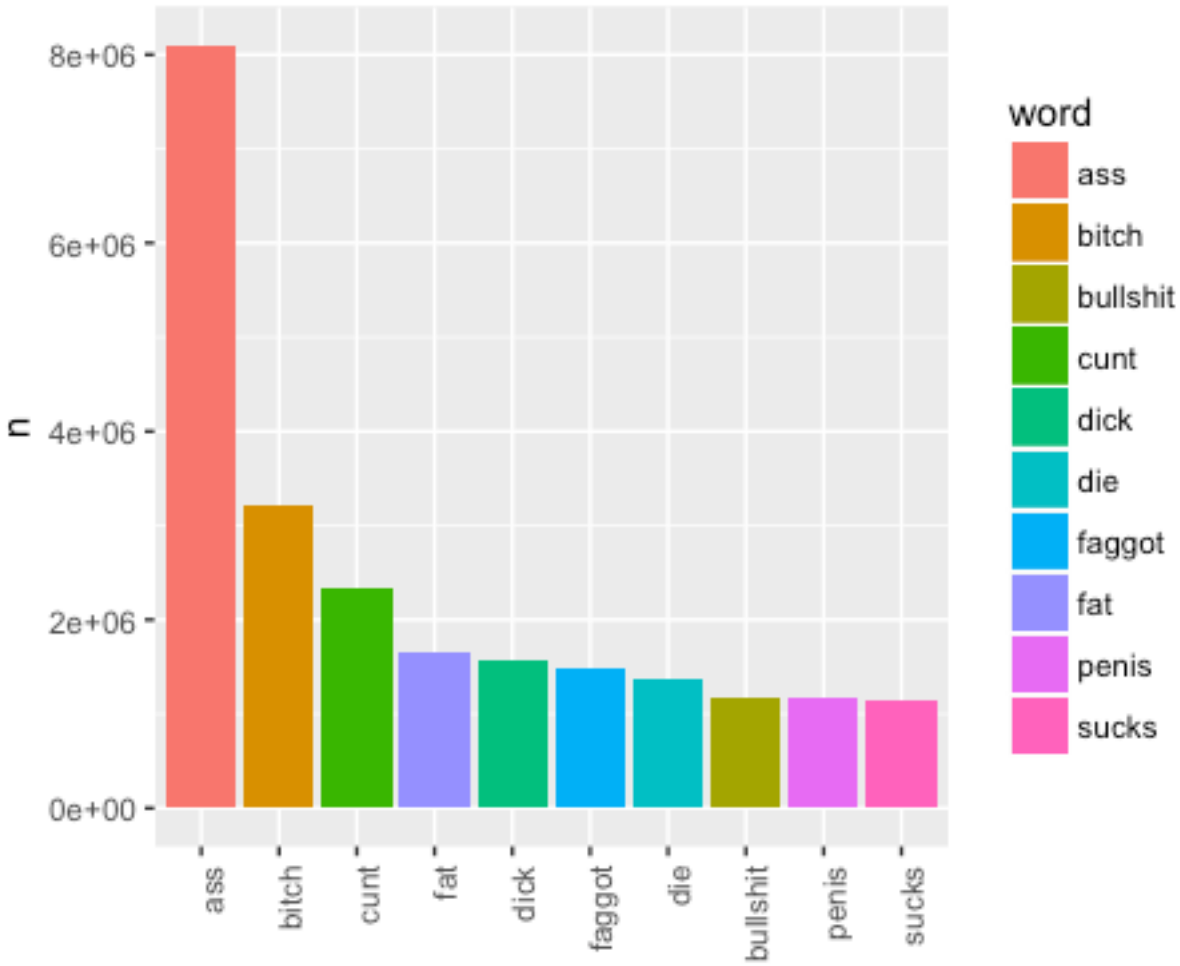
Severe toxic comments:

```
## Explore Severe toxic comment
severe_toxic.set <- comments.hex[, "severe_toxic"] == 1
severe_toxic.hex <- comments.hex[severe_toxic.set, ]
severe_toxic.words.hex <- clean_token(severe_toxic.hex$clean_text, stop_words_chr,
rare_words_chr)
h2o.head(severe_toxic.words.hex)
nrow(severe_toxic.words.hex) #46267
severe_toxic.words <- as.data.table(severe_toxic.words.hex)
severe_toxic.words <- severe_toxic.words %>% group_by(C1) %>% mutate(wc=n())
top.severe_toxic_words <- severe_toxic.words %>% tally(wc) %>% top_n(10) %>% ungroup()
head(top.severe_toxic_words, 10)
names(top.severe_toxic_words)[1] <- "word"
names(top.severe_toxic_words)[2] <- "n"


top.severe_toxic_words %>%
  ggplot(aes(x=reorder(word,-n), y=n, fill=word)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90, hjust=1), axis.title.x = element_blank())#
```



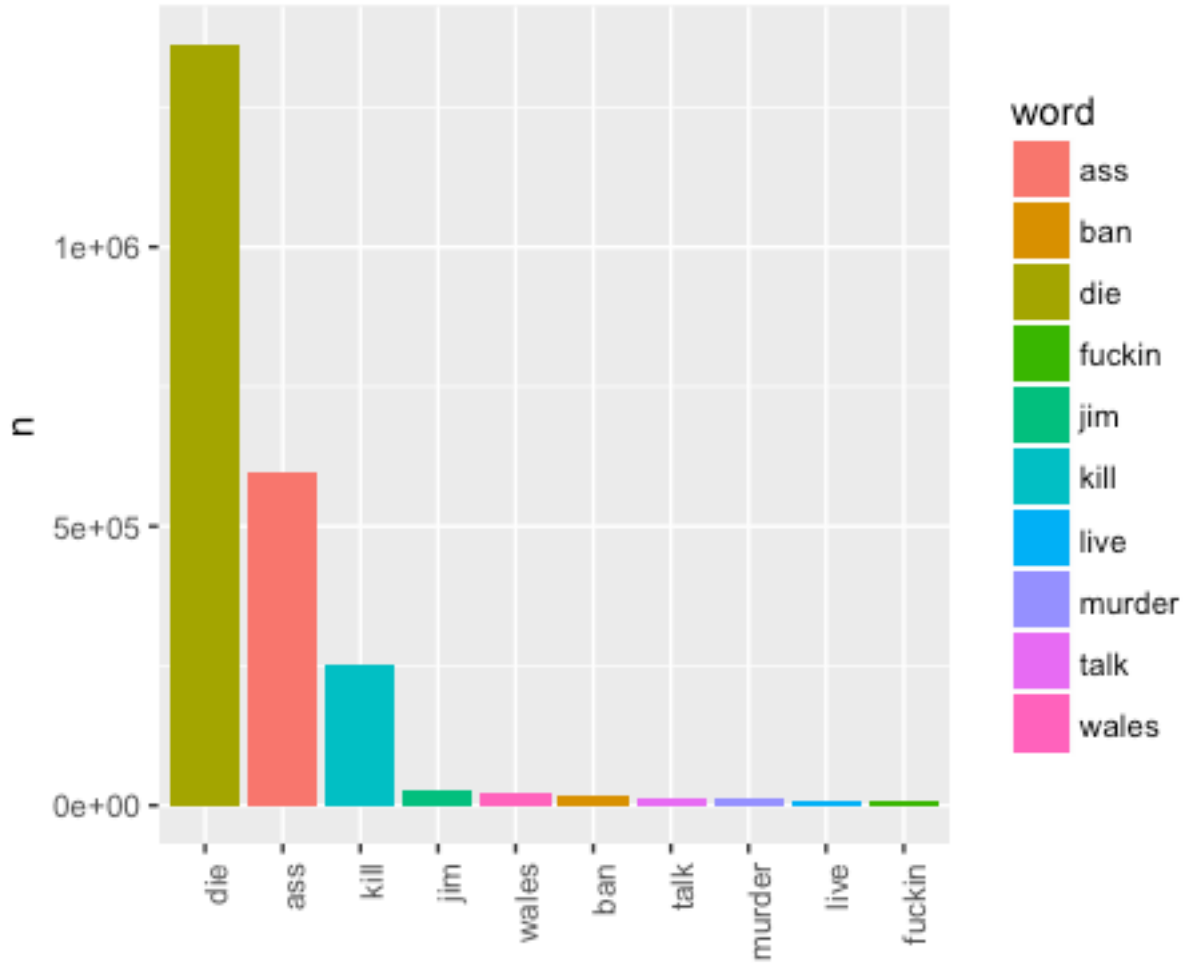
Explore obscene comments:

```
## Explore Obscene
obscene.set <- comments.hex[, "obscene"] == 1
obscene.hex <- comments.hex[obscene.set, ]
obscene.words.hex <- clean_token(obscene.hex$clean_text, stop_words_chr, rare_words_chr)
h2o.head(obscene.words.hex)
nrow(obscene.words.hex) #137138
obscene.words <- as.data.table(obscene.words.hex)
obscene.words <- obscene.words %>% group_by(C1) %>% mutate(wc=n())
top.obscene_words <- obscene.words %>% tally(wc) %>% top_n(10) %>% ungroup()
head(top.obscene_words, 10)
names(top.obscene_words)[1] <- "word"
names(top.obscene_words)[2] <- "n"


top.obscene_words %>%
  ggplot(aes(x=reorder(word,-n), y=n, fill=word)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90, hjust=1), axis.title.x = element_blank())#
```



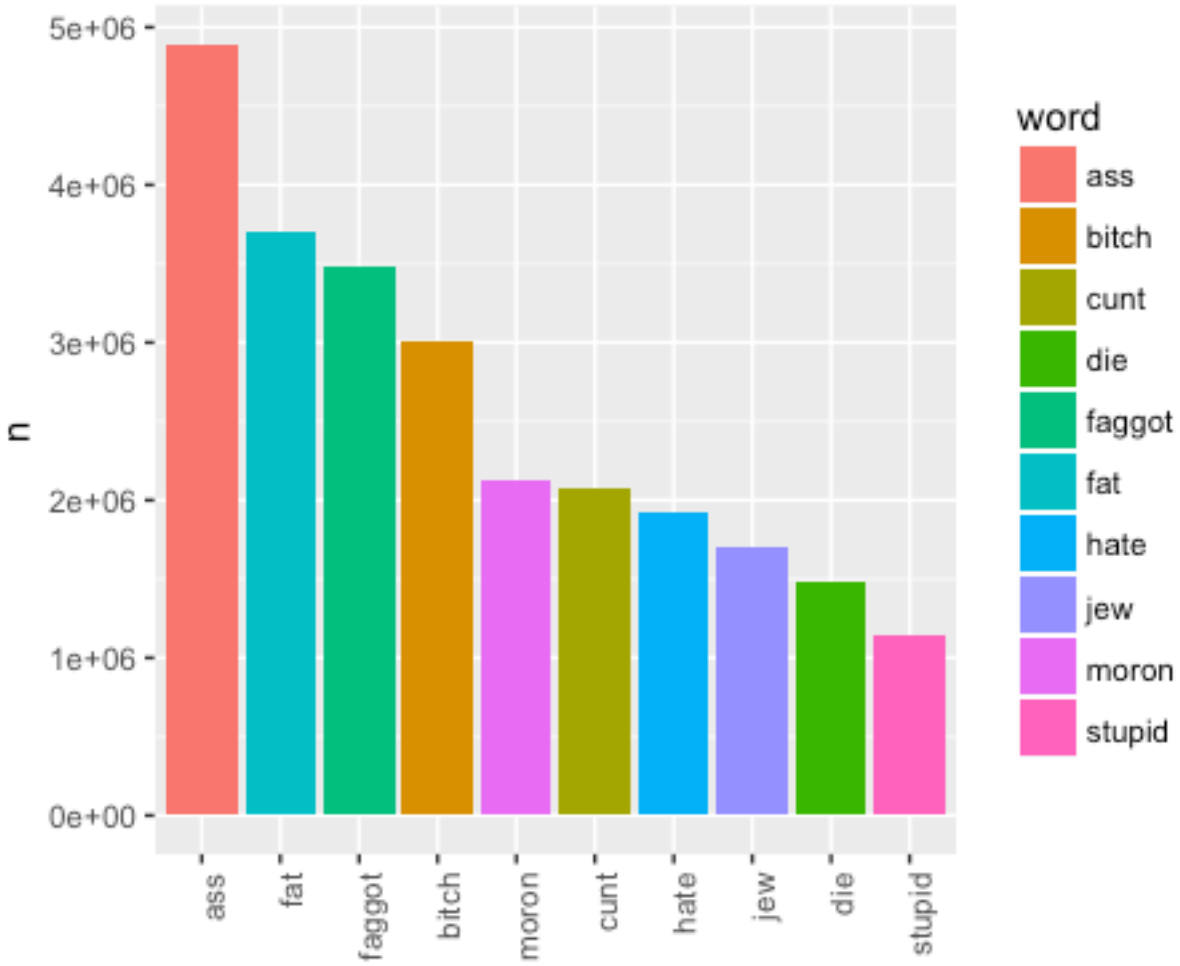
Explore threat comments:

```
## Explore threat
threat.set <- comments.hex[, "threat"] == 1
threat.hex <- comments.hex[threat.set, ]
threat.words.hex <- clean_token(threat.hex$clean_text, stop_words_chr, rare_words_chr)
h2o.head(threat.words.hex)
nrow(threat.words.hex) #9219
threat.words <- as.data.table(threat.words.hex)
threat.words <- threat.words %>% group_by(C1) %>% mutate(wc=n())
top.threat_words <- threat.words %>% tally(wc) %>% top_n(10) %>% ungroup()
head(top.threat_words, 10)
names(top.threat_words)[1] <- "word"
names(top.threat_words)[2] <- "n"


top.threat_words %>%
  ggplot(aes(x=reorder(word,-n), y=n, fill=word)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90, hjust=1), axis.title.x = element_blank())#
```



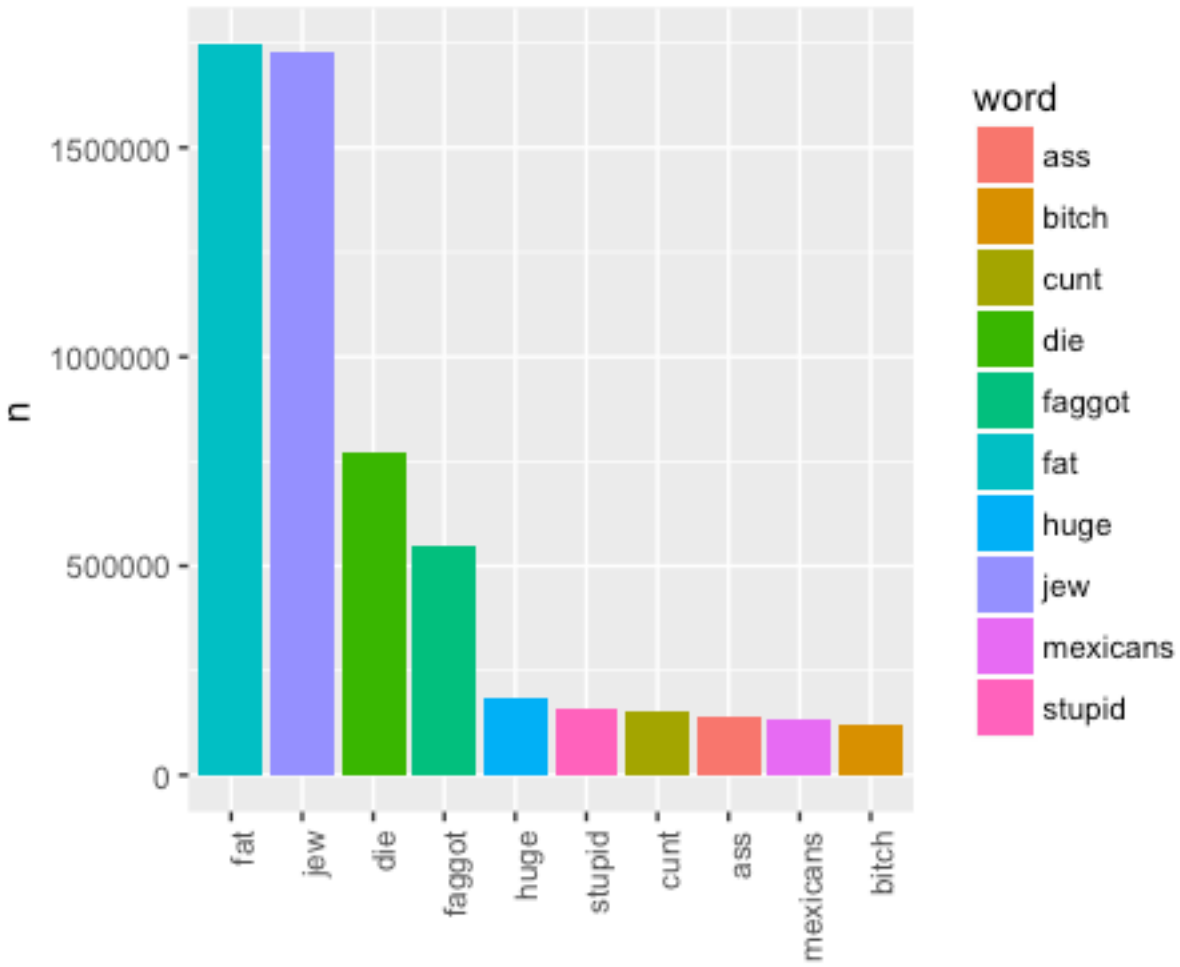
Explore insult comments:

```
## Explore insult
insult.set <- comments.hex[, "insult"] == 1
insult.hex <- comments.hex[insult.set, ]
insult.words.hex <- clean_token(insult.hex$clean_text, stop_words_chr, rare_words_chr)
h2o.head(insult.words.hex)
nrow(insult.words.hex) #9219
insult.words <- as.data.table(insult.words.hex)
insult.words <- insult.words %>% group_by(C1) %>% mutate(wc=n())
top.insult_words <- insult.words %>% tally(wc) %>% top_n(10) %>% ungroup()
head(top.insult_words, 10)
names(top.insult_words)[1] <- "word"
names(top.insult_words)[2] <- "n"


top.insult_words %>%
  ggplot(aes(x=reorder(word,-n), y=n, fill=word)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90, hjust=1), axis.title.x = element_blank())#
```



And explore identity hate comments:

```
## Explore identity_hate
identity_hate.set <- comments.hex[, "identity_hate"] == 1
identity_hate.hex <- comments.hex[identity_hate.set, ]
identity_hate.words.hex <- clean_token(identity_hate.hex$clean_text, stop_words_chr,
rare_words_chr)
h2o.head(identity_hate.words.hex)
nrow(identity_hate.words.hex) #27081
identity_hate.words <- as.data.table(identity_hate.words.hex)
identity_hate.words <- identity_hate.words %>% group_by(C1) %>% mutate(wc=n())
top.identity_hate_words <- identity_hate.words %>% tally(wc) %>% top_n(10) %>% ungroup()
head(top.identity_hate_words, 10)
names(top.identity_hate_words)[1] <- "word"
names(top.identity_hate_words)[2] <- "n"


top.identity_hate_words %>%
  ggplot(aes(x=reorder(word,-n), y=n, fill=word)) +
  geom_bar(stat="identity") +
  theme(axis.text.x=element_text(angle=90, hjust=1), axis.title.x = element_blank())#
```



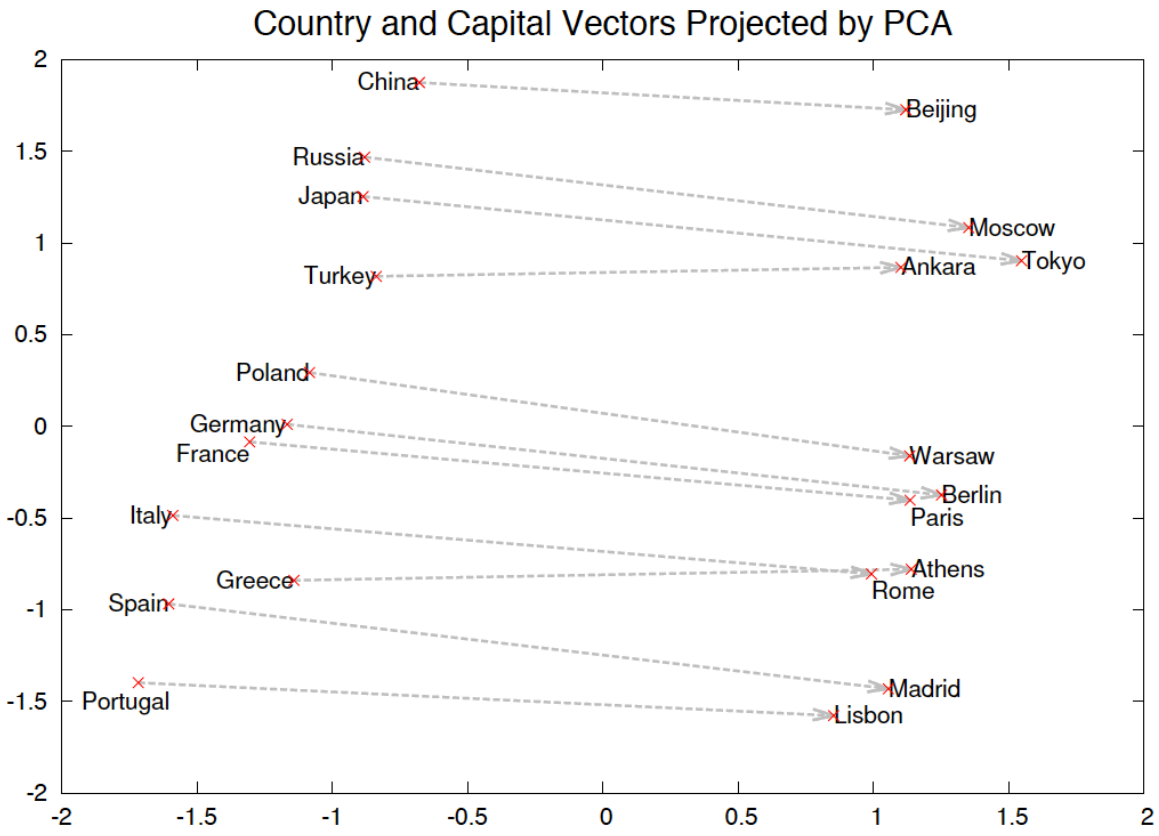
Word2Vec model

To better classify text, we need effective representation of words, sentences and text. Word2vec is an approach to represent the meaning of word.
The key idea is based on:
- Represent each word with a low-dimensional vector;
- Word similarity = vector similarity
- Predict surrounding words of every word

Word2Vec is faster and can easily incorporate a new sentence or add a word to the vocabulary (because it learns from input texts). It was developed by Mikolov, Sutskever, Chen, Corrado and Dean in 2013 at Google Research.

To represent the meaning of word, word2vec use two basic neural network models (shallow learning, no deep learning): Continuous Bag of Word (CBOW) that use a window of word to

predict the middle word and Skip-gram (SG) that use a word to predict the surrounding ones in window.
Word2vec demonstrates that, for vectorial representations of words, shallow learning can give great results. It scales very well, allowing models to be trained using more data (runs on a single machine and we can train a model with few resource at home).



Now we can use H2O.ai (http://www.h2o.ai) library to train our word2vec models and find some chains of synonyms (take some minutes, depend on your machine).

```
##############################################################################
####
# Build word2vec model
vectors <- 10
w2v.model <- h2o.word2vec(words.hex
                          , model_id = "w2v_model"
                          , vec_size = vectors
                          , min_word_freq = 10
                          , window_size = 5
                          , init_learning_rate = 0.025
                          , sent_sample_rate = 0
                          , epochs = 18)
```

Check - find synonyms for the word 'water',mexicans','hardcore','metallica','hot','idiot'
print(h2o.findSynonyms(w2v.model, "water", count = 5))
print(h2o.findSynonyms(w2v.model, "hardcore", count = 5))
print(h2o.findSynonyms(w2v.model, "metallica", count = 5))
print(h2o.findSynonyms(w2v.model, "hot", count = 5))
print(h2o.findSynonyms(w2v.model, "idiot", count = 5))
print(h2o.findSynonyms(w2v.model, "mexicans", count = 5))

The word2vec model can be used to train a XGBoost or other model to predict the toxicity grade for each comment.

```
> ##### Check - find synonyms for the word 'water',mexicans','hardcore','metallica','hot','idiot'
> print(h2o.findSynonyms(w2v.model, "water", count = 5))
   synonym     score
1 drilling 0.9778500
2 steadily 0.9723787
3     coal 0.9713380
4      oil 0.9712507
5   divers 0.9706305
> print(h2o.findSynonyms(w2v.model, "hardcore", count = 5))
   synonym     score
1    metal 0.9753072
2     soad 0.9713529
3     punk 0.9713167
4     prog 0.9690148
5 subgenre 0.9690027
> print(h2o.findSynonyms(w2v.model, "metallica", count = 5))
  synonym     score
1     rap 0.9805354
2    funk 0.9800954
3    punk 0.9689288
4 misfits 0.9687625
5   dusty 0.9652824
> print(h2o.findSynonyms(w2v.model, "hot", count = 5))
  synonym     score
1   shoes 0.9871039
2   shirt 0.9757431
3   candy 0.9718538
4    gals 0.9679759
5   tires 0.9669617
```