Improved embeddings

Thomas Lumley

9/26/2018

Pre-trained embeddings

GloVe embeddings are based on a model based on conditional probabilities of co-occurrence. If X_i is the count of occurrences of word i, and X_{ij} is the count of occurrences of word j in a text, then it tries to have

$$(w_i - w_j)^T w_k = \log \frac{X_{ik}/X_k}{X_{jk}/X_k}$$

Allows for analogies as vectors, eg

$$w(\text{king}) - w(\text{queen}) \approx w(\text{man}) - w(\text{woman})$$

a

The model is fitted by minimising

$$Q = \sum_i X_{ij}{}^{\alpha}(w_i^T w_j + b_i + b_j - \log X_{ij})$$

where b_i are intercept terms relating to the frequency of each word.

Pre-trained GloVe

We will use the 42B GloVe embeddings from Stanford, trained on 42 billion words from the 'Common Crawl' web corpus.

There are nearly 2 million distinct words and 300 dimensions of embedding

The dataset is slightly too large to use in R with 8MB memory

dbplyr

Read the data into MonetDB

```
library(DBI)
library(MonetDBLite)
dbcon<-dbConnect(MonetDBLite(),"./DB")
monet.read.csv(dbcon, "glove.42B.300d.txt", "glove",header=FALSE,
 best.effort=TRUE, sep=" ",quote="",
 col.names=c("word",paste0("e",1:300)))
```

Training data

Human-rated "positive" and "negative" words from https://www.cs.uic.edu/~liub/FBS/sentiment-analysis.html#lexicon

```
pos_words<-scan("positive-words.txt", blank.lines.skip=TRUE,
comment.char=";", what="")
neg_words<-scan("negative-words.txt", blank.lines.skip=TRUE,
comment.char=";", what="")

library(dplyr)
ms<-MonetDBLite::src_monetdblite("./DB")
train_words<-tibble(word=c(pos_words,neg_words),
  positive=rep(1:0,c(length(pos_words),length(neg_words))))
train_words<-copy_to(ms, train_words,
   name="train_words",temporary=FALSE)
```

Extract embeddings

Look up words with `inner_join`, read into R, convert to a matrix

```
glove<-tbl(ms,"glove")
train_words<-tbl(ms,"train_words")
train <- inner_join(train_words, glove) %>% collect()
train_y<-train$positive
train_x<-train %>% select(-word,-positive) %>% as.matrix()
rownames(train_x)<-train$word
```

Fit a model

We'll fit a logistic model with an L2 penalty: essentially a network with no hidden layers and a single neuron with sigmoid activation function.

```
library(glmnet)
test<-sample(nrow(train_x),nrow(train_x)/5)
fit<-cv.glmnet(train_x[-test,],train_y[-test],family="binomial" )
predicted<-predict(fit$glmnet.fit, train_x[test,],s=fit$lambda.min)
```

Out of sample predictions: pretty good!

```
> predicted[sample(nrow(predicted),20),]
     heartily      throbbed    enchanting       screwed inconsistence
     6.550734     -2.219986      6.928539     -6.624885     -3.799926
    castigate      scratchy        poorly disintegrated        defame
    -3.404154     -8.875896     -5.705270     -5.424058     -5.034735
      smudged insufficiency      ferocity    inexorably  irredeemable
    -6.615081     -5.092952     -2.888919     -2.187144     -5.236300
      unknown   disapointed      venerate      strictly        jagged
    -4.940612     -3.690092      1.403324      0.646811     -5.756622
```

Predictions

```
make.sentiment <- function(db,model){
    glove<-tbl(ms,"glove")
    function(text){
      word_tbl<-copy_to(db,
      tibble(
        word=tolower(strsplit(text,"[[:blank:],.!?;:'\"]")[[1]])),
            name="temp_words",overwrite=TRUE)
        word_x<-inner_join(word_tbl, glove) %>%
             collect() %>% select(-word) %>% as.matrix()
        if(nrow(word_x)==0) return(0)
        sentiments<-predict(model$glmnet.fit, word_x,
            s=model$lambda.min)
        mean(sentiments)
        }
    }

sentiment<-make.sentiment(ms,fit)
```

Try it out

```
> sentiment("They have a cave troll")
[1] -2.000714
> sentiment("You shall not pass")
[1] -0.1889209
> sentiment("The others cast themselves down upon the fragrant grass,
  but Frodo stood awhile still lost in wonder.")
[1] -0.1405773
> sentiment("If more of us valued food and cheer and song above
  hoarded gold, it would be a merrier world.")
[1] 2.495909
```

Unfortunate

```
> sentiment("Let's go out for Italian food.")
[1] 1.387002
> sentiment("Let's go out for Chinese food.")
[1] 1.04452
> sentiment("Let's go out for Mexican food.")
[1] 0.6954334
```

More personal

```
> sentiment("My name is Thomas and I am a data scientist")
Joining, by = "word"
[1] 1.284291
> source("studentnames.R")
> students$namescore<-sapply(students$given,
+   function(n) sentiment(paste("My name is",n,"and I am a data scientist.")))

> students<-students[order(students$namescore),]
```

```
> head(students[,-1])
      given namescore
27   Hrishi 0.7586472
20     Riki 0.8197963
32  Marrick 0.8301052
2     Omkar 0.8407849
4   Raditya 0.8588888
49      Gus 0.8882476
> tail(students[,-1])
      given namescore
61     Erin  1.361055
60   Lauren  1.366832
29   Joshua  1.368145
13   Angela  1.448930
59  Deborah  1.451585
38    Grace  1.765753
```

```
> sentiment("My name is Grace and I am a bank robber")
[1] 0.9233358
> sentiment("My name is Deborah and I am a bank robber")
[1] 0.6091682
> sentiment("My name is Grace and I am a data scientist")
[1] 1.765753
> sentiment("My name is Deborah and I am a data scientist")
[1] 1.451585
> sentiment("My name is Hrishi and I am a data scientist")
[1] 0.7586472
> sentiment("My name is Marrick and I am a data scientist")
[1] 0.8301052
> sentiment("My name is Hrishi and I love  kittens")
[1] 0.9681818
> sentiment("My name is Marrick and I love  kittens")
[1] 1.057504
> sentiment("My name is Hrishi and I love happy kittens")
[1] 1.696798
```

Input bias

The training words for our model were hand-selected, but the training data for the word embeddings was just available text.

There's similar bias with the `word2vec` embeddings trained on Google News

Machine learning will happily extract bias from the data it's given.

One approach to reducing bias is to use the semantic structure of the embeddings: any particular type of bias will be low-dimensional and can be estimated and subtracted off to give new embeddings

Basically: set up matched pairs of words that shouldn't be different, and see how different they are.

Ethics

Debiasing in this way doesn't seem to reduce predictive accuracy for sentiment, but it takes a lot of work.

Modern predictive models will pick up **any** signal in the input data, so **you** need to make sure you don't have signals you don't want picked up.