Statistical Language Models

Week 9

Cookbook example revisited

- library(wordVectors)
- model = train_word2vec("cookbooks.txt","cookbook_vectors.bin",
- vectors=200,threads=4,
- cbow=1,window=12,
- iter=500,negative_samples=0)
- Iter is number of times it passes through the corpus. With many words smaller iters (fewer passes) is not so bad, for smaller corpus expect to need .
- Wod2Vec can take many hours to train (iters=5 takes ~1 minute on 4 cores)

Fish words

- #Find the fish words:
- model %>% closest_to("fish")
- #Expand out to search for more fish words:
- model %>% closest_to(model[[c("fish","salmon","trout","shad","flounder","carp","roe", "eels")]],50)
- #Can be used to find all recipes that relate to fish things, find ways of cooking fish,…

Make a plot in principal compenents space

- some_fish = closest_to(model,model[[c("fish","salmon","trout","shad","flounder","carp","roe","eels")]],150)

- fishy = model[[some_fish$word,average=F]]

- plot(fishy,method="pca")

Clustering word vectors to find "topics"

- Not the same as clustering documents.
- Text —> Word2Vec embeddings —> clusters
- Cluster words by similar usage not clustering into groups with similar word probabilities

Clustering word vectors to find topics

- centers = 150
- clustering = kmeans(model,centers=centers,iter.max = 40)
- sapply(sample(1:centers,10),function(n) {
-   names(clustering$cluster[clustering$cluster==n][1:10])
- })

Hierarchical clustering from seed words

- ingredients = c("peanut","onion","cumin","coconut")
- term_set = lapply(ingredients,
-         function(ingredient) {
-             nearest_words = model %>% closest_to(model[[ingredient]],20)
-             nearest_words$word
-         }) %>% unlist

- subset = model[[term_set,average=F]]
- subset %>%
-   cosineDist(subset) %>%
-   as.dist %>%
-   hclust %>%
-   plot

Relationship Planes

- tastes = model[[c("sweet","salty"),average=F]]
- # model[1:3000,] here restricts to the 3000 most common words in the set.
- sweet_and_saltiness = model[1:3000,] %>% cosineSimilarity(tastes)
- # Filter to the top 20 sweet or salty.
- sweet_and_saltiness = sweet_and_saltiness[
-   rank(-sweet_and_saltiness[,1])<20 |
-   rank(-sweet_and_saltiness[,2])<20,
-   ]
- plot(sweet_and_saltiness,type='n')
- text(sweet_and_saltiness,labels=rownames(sweet_and_saltiness))

Flavour planes

- tastes = model[[c("sweet","salty","savory","bitter","sour"),average=F]]

- # model[1:3000,] here restricts to the 3000 most common words in the set.
- common_similarities_tastes = model[1:3000,] %>% cosineSimilarity(tastes)

- #examine a few words, not the most common ones:
- common_similarities_tastes[20:30,]

Flavour planes

- #limit to the closest words to
- high_similarities_to_tastes = common_similarities_tastes[rank(-apply(common_similarities_tastes,1,max)) < 75,]
- high_similarities_to_tastes %>%
-   prcomp %>%
-   biplot(main="Fifty words in a\nprojection of flavor space")

Analytic tasks

- Use sentiment or topic number to examine the evolution of a book (or legal case transcript)
- Consider a time series of cosine similarities to look at drift in words, change in language.
-

Job Adverts Data Set
2339 Job ads in Data Science etc as categorized by lexicon

```
• BriefType = rep(NA,length(Title))
• BriefType[grep(Title,pattern="(data)|(statistic)|(learning)",ignore.case = TRUE)]                      = "OtherData"
• BriefType[grep(Title,pattern="(data)*.*(Research)|(post[[:punct:]]*\\s*doc)",ignore.case = TRUE)]     = "DataResearcher"
• BriefType[grep(Title,pattern="(math)|(crypt)|(geom)|(Operations+ research)",ignore.case = TRUE)]  = "Math"
• BriefType[grep(Title,pattern="(tutor)|(teach)",ignore.case = TRUE)]                                    = "Teacher"
• BriefType[grep(Title,pattern="analy",ignore.case = TRUE)]                                              = "Analyst"
• BriefType[grep(Title,pattern="engineer",ignore.case = TRUE)]                                           = "Engineer"
• BriefType[grep(Title,pattern="technician",ignore.case = TRUE)]                                         = "DataTechnician"
• BriefType[grep(Title,pattern="data scien",ignore.case = TRUE)]                                         = "DataScientist"
• BriefType[grep(Title,pattern="(machine)|(ML)|(deep\\slearning)|(\\sai\\s)|(\\sartificial\\sintelligence\\s)",ignore.case = TRUE)] = "ML"
• BriefType[grep(Title,pattern="(machine)|(ML)|(deep\\slearning)\\s*Research",ignore.case = TRUE)]  = "MLResearcher"
• BriefType[grep(Title,pattern="math.*Research",ignore.case = TRUE)]                                     = "MathResearch"
• BriefType[grep(Title,pattern="(manager)|(lead)",ignore.case = TRUE)]                                   = "ManagerLead"
• BriefType[sapply(Title,nchar)>500]                                                                     = NA  #I consider these to be broken
```

- library(wordVectors)
- library(magrittr)

```
head(UniqueJobsCleanWithIndicatorsNoFrench[,"DescFullOrig"])

model10_200 =
            train_word2vec("UniqueJobsCleanWithIndicatorsNoFrench.txt",
         "Jobs_CBOW_wind10_iter500_vec200.bin",vectors=200,threads=6,
                         cbow=1,window=10,  iter=500,negative_samples=0)
```

Closest to may not always be "most commonly used and closest to"

- model20_200 %>% closest_to("statistics")
- model20_200 %>% closest_to("r")
- model20_200 %>% closest_to("data")
- grep(pattern="kindergarten",UniqueJobsCleanWithIndicatorsNoFrench[,"DescFullOrig"],ignore.case=TRUE)

Closest to (vector addition)

- model20_200 %>% closest_to(~"statistics"+"science",25)
- model20_200 %>% closest_to(~"predict"+"visualize",25)

- model %>% closest_to(~"cheese"+"broccoli",25)
- model %>% closest_to(~"cinnamon"+"raisin",25)

Closest to (vector subtraction)

- model20_200 %>% closest_to(~"degree"-"phd",25)
- model20_200 %>% closest_to(~"statistics"-"communication",25)#orthog

- model %>% closest_to(~"cheese"-"broccoli",25)
- model %>% closest_to(~"cinnamon"-"raisin",25)

Antonyms and synonyms and bias

- demo_vectors %>% closest_to("good")
- demo_vectors %>% closest_to("bad")
- demo_vectors %>% closest_to(~"bad"-"good")

- demo_vectors %>% closest_to(~ "he" - "she")
- demo_vectors %>% closest_to(~ "she" - "he")

Related to vector addition in high dimensions

- "cinnamon" is to "raisins" as sweet is to?
- model %>% closest_to(~"cinnamon"-"raisins"+"sweet",15)
- Another way is to think of removing the "sweet" without "raisins" direction from cinnamon:
- model %>% closest_to(~"cinnamon"-"raisins"+"sweet",15)
- model20_200 %>% closest_to(~"machine"-"neural"+"predict",25)

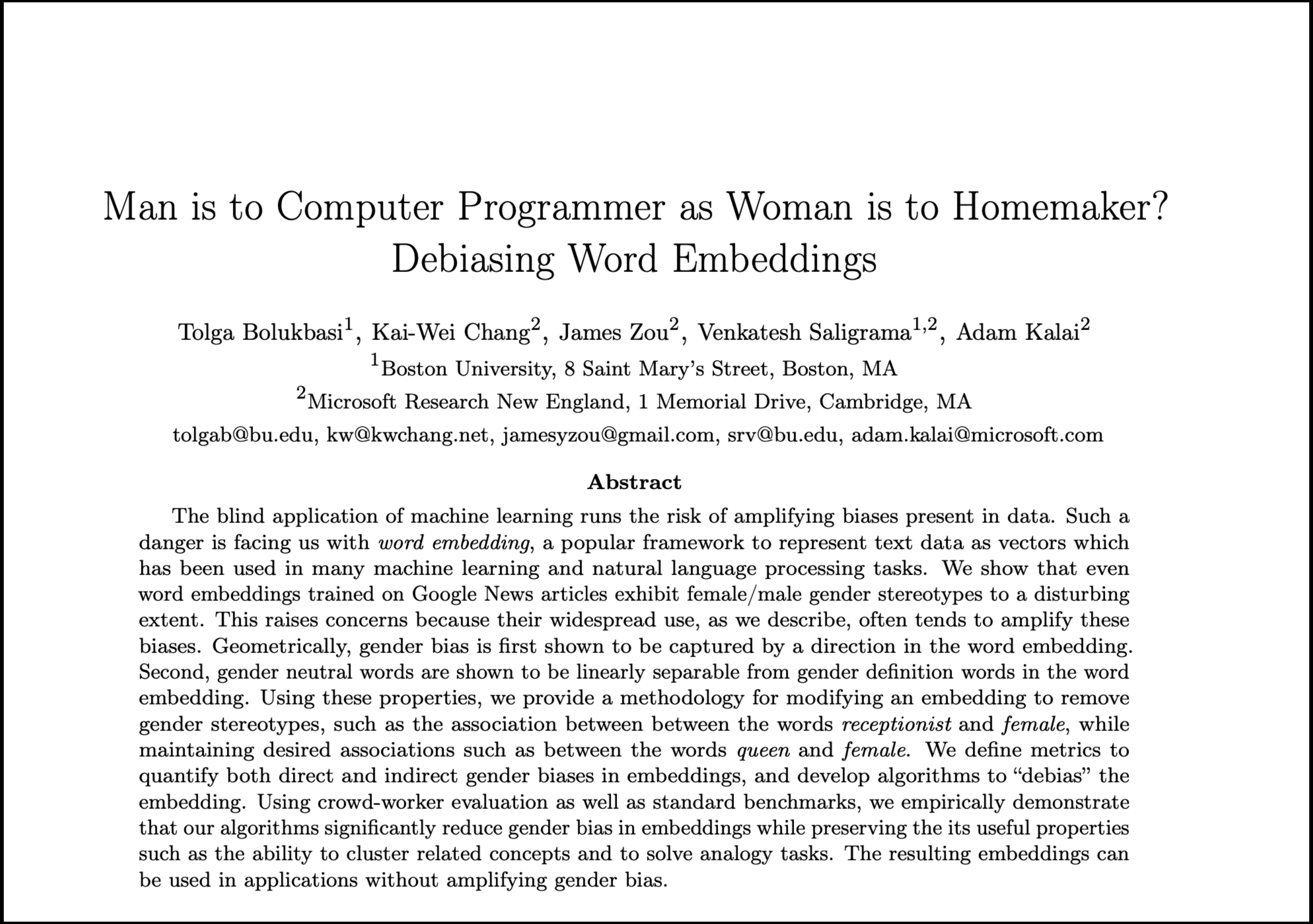
- model20_200[[c("statistics","computer","learning","machine","phd","masters","mathematics"), average=F]] %>%

-   plot(method="pca")

- model20_200[[c("statistics","computer","learning","neural","predict","deep","machine"), average=F]] %>%

-     plot(method="pca")

https://arxiv.org/abs/1607.06520



Man is to Computer Programmer as Woman is to Homemaker? Debiasing Word Embeddings

Tolga Bolukbasi[1], Kai-Wei Chang[2], James Zou[2], Venkatesh Saligrama[1,2], Adam Kalai[2]
[1]Boston University, 8 Saint Mary's Street, Boston, MA
[2]Microsoft Research New England, 1 Memorial Drive, Cambridge, MA
tolgab@bu.edu, kw@kwchang.net, jamesyzou@gmail.com, srv@bu.edu, adam.kalai@microsoft.com

Abstract

The blind application of machine learning runs the risk of amplifying biases present in data. Such a danger is facing us with *word embedding*, a popular framework to represent text data as vectors which has been used in many machine learning and natural language processing tasks. We show that even word embeddings trained on Google News articles exhibit female/male gender stereotypes to a disturbing extent. This raises concerns because their widespread use, as we describe, often tends to amplify these biases. Geometrically, gender bias is first shown to be captured by a direction in the word embedding. Second, gender neutral words are shown to be linearly separable from gender definition words in the word embedding. Using these properties, we provide a methodology for modifying an embedding to remove gender stereotypes, such as the association between between the words *receptionist* and *female*, while maintaining desired associations such as between the words *queen* and *female*. We define metrics to quantify both direct and indirect gender biases in embeddings, and develop algorithms to "debias" the embedding. Using crowd-worker evaluation as well as standard benchmarks, we empirically demonstrate that our algorithms significantly reduce gender bias in embeddings while preserving the its useful properties such as the ability to cluster related concepts and to solve analogy tasks. The resulting embeddings can be used in applications without amplifying gender bias.