Natural_Language_Processing_Project

Initialization

The initial step that loads the required libraries and downloads the data sets if not all read on file.

```
library(tidyverse)
library(tidytext)

#downloads the corpus files, profanity filter and English dictionary

url <- "https://d396qusza40orc.cloudfront.net/dsscapstone/dataset/Coursera-SwiftKey.zip"
url2 <- "https://www.freewebheaders.com/download/files/facebook-bad-words-list_comma-separated-text-fil
url3 <- "https://raw.githubusercontent.com/dwyl/english-words/master/words_alpha.txt"
if(dir.exists("~/R/Capestone/data/") == FALSE){
       dir.create("~/R/Capestone/data/")}

if(file.exists("~/R/Capestone/data/data.zip") == FALSE|
   file.exists("~/R/Capestone/data/prof.zip")==FALSE|
   file.exists("~/R/Capestone/data/diction.txt")==FALSE){
        download.file(url,destfile = "~/R/Capestone/data/data.zip")
        download.file(url2,destfile = "~/R/Capestone/data/prof.zip")
        download.file(url3,destfile = "~/R/Capestone/data/diction.txt")
        setwd("~/R/Capestone/data/")
        unzip("~/R/Capestone/data/prof.zip")
        unzip("~/R/Capestone/data/data.zip")
        setwd("~/R/Capestone")
}
```

Create corpus

At this stage the files are open and joined to create a corpus for the project. The Corpus is so large and requires some much ram that a sample of 20% is taken.

```
blog <- read_lines("~/R/Capestone/data/final/en_US/en_US.blogs.txt")
news <- read_lines("~/R/Capestone/data/final/en_US/en_US.news.txt")
twitter <- read_lines("~/R/Capestone/data/final/en_US/en_US.twitter.txt")
blog <- tibble(text = blog)
news <- tibble(text = news)
twitter <- tibble(text = twitter)

set.seed(90210)
corpus <- full_join(blog,twitter) %>% full_join(news) %>%
        slice_sample(prop = 0.10)
```

Corpus filtering

Here the corpus filter is created to remove profanity and any word that is not in the English dictionary.

```
prof <- read_lines("~/R/Capestone/data/facebook-bad-words-list_comma-separated-text-file_2021_01_18.txt
prof <- prof %>% str_split(", ") %>% flatten %>% unlist
prof <- tibble("word" = prof)

english <- read_lines("~/R/Capestone/data/diction.txt")
english <- tibble("word" = english[!english==""])
```

Vocabulary

A vocabulary of words is created from the unique words with the applied filters.

```
#clean up ram
rm(blog,news,twitter)
voc <- english %>% anti_join(prof)

unigram <- corpus %>% unnest_tokens(ngram, text, token = "ngrams", n = 1) %>%
        semi_join(voc, by = c("ngram"="word"))
#decreases the voc size
voc <- tibble(word = unigram$ngram)
```

Out of Vocabulary

To model out of vocabulary words we take a sampling of the least frequent unigrams and change them to the character "". If a word is tested that isn't in the vocabulary for the corpus, the quantity will be converted to "".

```
#OOV 1% of the least likely unigrams
unk <- unigram %>%
        count(ngram) %>%
        filter(n==1) %>%
        slice_sample(prop = 0.005)
unigram[unigram$ngram %in% unk$ngram,]$ngram <- "<unk>"
unigram <- count(unigram, ngram)
```

There are still many word that only appear once (14887) so it useful to remove these words as the add little to prediction value of the models but significantly to the time and memory requirement. This is mainly due to the filtering.

```
remove <- unigram$n==1
unigram <- unigram[!remove,]
voc <- tibble(word = unigram$ngram)
```

Bigrams

The bigrams are created and then split into individual words which can be filtered by the vocabulary list.

```
bigram <- corpus %>% unnest_tokens(ngram, text, token = "ngrams", n = 2) %>%
        separate(ngram, c("word1", "word2"))
bigram$word1[bigram$word1 %in% unk$ngram] <- "<unk>"
bigram$word2[bigram$word2 %in% unk$ngram] <- "<unk>"
bigram <- bigram %>% filter(word1 %in% voc$word, word2 %in% voc$word) %>%
        count(word1, word2)
```

Trigrams

Likewise, the trigrams are created in a similar manner

```
trigram <- corpus %>% unnest_tokens(ngram, text, token = "ngrams", n = 3) %>%
        separate(ngram, c("word1", "word2", "word3"))
trigram$word1[trigram$word1 %in% unk$ngram] <- "<unk>"
trigram$word2[trigram$word2 %in% unk$ngram] <- "<unk>"
trigram$word3[trigram$word3 %in% unk$ngram] <- "<unk>"
trigram <- trigram %>%
        filter(word1 %in% voc$word, word2 %in% voc$word, word3 %in% voc$word) %>%
        count(word1, word2, word3)
```

Testing

A tibble is made for model testing, the test set itself is from one of the quizzes

```
dist <- tibble(word = voc$word)
input <- tibble(text = c("When you breathe, I want to be the air for you. I'll be there for you, I'd li
                         "Guy at my table's wife got up to go to the bathroom and I asked about dessert
                         "I'd give anything to see arctic monkeys this",
                         "Talking to your mom has the same effect as a hug and helps reduce your",
                         "When you were in Holland you were like 1 inch away from me but you hadn't tim
                         "I'd just like all of these questions answered, a presentation of evidence, an
                         "I can't deal with unsymetrical things. I can't even hold an uneven number of
                         "Every inch of you is perfect from the bottom to the",
                         "I'm thankful my childhood was filled with imagination and bruises from playin
                         "I like how the same people are in almost all of Adam Sandler's")) %>%
        mutate(line = row_number()) %>%
        unnest_tokens(word, text) %>%
        filter(word %in% voc$word)

w <- filter(input, line==1) %>%
        select(word) %>%
        slice_tail(n=2) %>%
        as.vector()
```

Modeling

In this section, the different models are examined

Maximum Likliness Estimate (MLE)

```
temp <- trigram %>% filter(word1 %in% w[1,], word2 %in% w[2,])
MLEdf <- tibble(word=temp$word3, MLE=temp$n/sum(temp$n))
dist <- left_join(dist, MLEdf)
dist[is.na(dist$MLE),]$MLE <- 0
```

Add one smoothing

```
ADDdf <- tibble(word=temp$word3, ADD=temp$n)
dist <- left_join(dist, ADDdf)
dist[is.na(dist$ADD),]$ADD <- 0
dist$ADD <- dist$ADD + 1
dist$ADD <- dist$ADD / sum(dist$ADD)
```

Good Turing

```
Nr <- count(temp, n) %>% add_row(n = 0, nn = 0)  %>%
        arrange(n)
Nr %<>% mutate(c= 0) %>% mutate(GT = 0)
total <- sum(Nr$nn*Nr$n)
Nr$GT[Nr$n==0] <- Nr$nn[Nr$n==1] / total
Nr$nn[Nr$n==0] <- nrow(dist) - total
for (i in 2:nrow(Nr)) {
        Nr$c[i] <-  (Nr$n[i]+1)*Nr$nn[i+1]/Nr$nn[i]
        Nr$c[i][is.na(Nr$c[i])] <- 0
        Nr$GT[i] <-  Nr$c[i] / total
}
Nr$GT[Nr$n==max(Nr$n)] <- 1 - sum(Nr$GT)
Nr$GT <- Nr$GT/ Nr$nn
GT <- tibble(word=voc$word)
GT <- left_join(GT, select(temp, word3, n), by = c("word" = "word3"))
GT$n[is.na(GT$n)] <- 0
GT <- left_join(GT, select(Nr, n, GT))
dist <- left_join(dist, select(GT, word, GT))
```

Absolute Discounting

```
ADdf <- voc %>% left_join(select(temp, word3, n), by = c("word" = "word3"))
ADdf$n[is.na(ADdf$n)] <- 0
d <- 0.5
temp2 <- trigram %>% filter(word1 == w[1,]) %>%
        count(word3) %>%
        mutate(p3 = n / sum(n))

ADdf <- mutate(ADdf, p = n /sum(n), p1 = ((n-d) > 0) * (n-d) / sum(n),
               p2 = d/sum(n)* nrow(temp))
ADdf <- left_join(ADdf, select(temp2, word3, p3), by = c("word" = "word3" ))
ADdf$p3[is.na(ADdf$p3)] <- 0
ADdf <- mutate(ADdf, AD = p1+p2*p3)
```

Kneser Ney

Results

```
maxes <- dist %>% filter(!word == "<unk>") %>%
        summarise_at(-1, which.max) %>%
        flatten() %>%
        unlist
dist[maxes,]
```

```
## # A tibble: 3 x 4
##   word    MLE       ADD         GT
##   <chr> <dbl>     <dbl>      <dbl>
## 1 wore      0 0.0000203 0.00000730
## 2 wore      0 0.0000203 0.00000730
## 3 wore      0 0.0000203 0.00000730
```