Prediction model(s) for next word prediction: Model Building, Model Selection and Predictions

[Author: Prashant Ratnaparkhi]

Overview

The goal is to build a predictive text model to predict next word in a sequence of words. This report focuses on the model building and model selection process. The raw data, used for model building, consists of three files containing twitter messages, news articles and blogs written in colloquial English text. These files constituted the corpus for the project. Natural Language Processing (NLP) course videos, course notes and other references listed in Section 6, were reviewed and different approaches for model building and prediction were considered. This report shows how the data is transformed at each step. This report has seven sections. First section summarizes the raw data, the second section summarizes the results of cleaning & preprocessing the data.The third section describes how the models are built, evaluated and selected. The fourth section demonstrates the results of prediction obtained using the selected model. Notes on the implementation are included in the fifth section. The sixth section lists the references. The last part is Section 7 - Source Code and it contains the complete source code used to produce this document.

1. Brief summary of raw data

The number of lines, words and word types (unique words) are tabulated below. This includes numbers, non-ascii text, symbols, swear words, mis-spelt words - essentially anything & everything included in the file. This table is included to give a good sense of the size of the raw data. [Source code is in Section 7.1.]

```
##            fileName numLines numWords numWordTypes
## 1 en_US.twitter.txt  2360148 31105049       336039
## 2    en_US.news.txt    77259  2755778        84974
## 3   en_US.blogs.txt   899288 38601569       309279
```

2. Cleaning, pre-processing & corpus creation

For the final selected language model (based on Good-Turing discounting and Katz back off scheme), a sample of 60% of raw data is selected to create training data set. Cleaning & preprocessing involved removal of: (i) non-ascii characters, (ii) swear words, (iii) numbers, and (iv) punctuation marks. After pre-processing, and cleaning, the Corpus object was created using using 'tm' package. [Note: Source code is in function 'createNgmsForTraining' and 'preprocAndCreateCorpus' of Section 7.2.]

3. Language Models:

3.1 Approaches considered and evaluated

1) After viewing the lectures on language models (references 1 & 2), it was decided to use tri-gram langauge model, for the first implementation. Most of the reference materials, indicated that tri-gram language models are reasonably accurate and it is very hard to get higher accuracy using higher level (say four-gram or five-gram language models).

2) Uni-gram, bi-gram and tri-gram tables were built using 'TermDocumentMatrix' from 'tm' package. There are: 470195 uni-grams, 5858843 bi-grams and 15674831 tri-grams in the respective tables. These three tables have the n-gram (u, uv or uvw) and respective 'Count' in each row, and that forms the

basis for model building. Sample data from the three tables is shown below. [Note: Source code is in function 'createDifferentNgrams' of Section 7.2].

```
##               u Count
##  1:   brastraps     1
##  2:    braswell     3
##  3:   braswells     1
##  4:        brat   130
##  5:       brata     1
##  6:   brataboom     1
##  7:    bratbino     1
##  8:    bratcher     3
##  9: bratchikova     1
## 10:      brated     1
```

```
##                    uv Count
##  1:        about cmin     1
##  2:        about cmon     1
##  3:         about cms     2
##  4:         about cmu     1
##  5:         about cnn     2
##  6: about cnndialogues     1
##  7:    about coachella     2
##  8:      about coaches     3
##  9:     about coaching     5
## 10:       about coachs     1
```

```
##                   uvw Count
##  1:     a couple folks     1
##  2:   a couple follows     1
##  3:       a couple for    11
##  4:  a couple formerly     1
##  5:   a couple formica     1
##  6:      a couple four     1
##  7:      a couple free     7
##  8: a couple freelance     1
##  9:   a couple friends    16
## 10:      a couple from    15
```

3.2 Models built and evaluated

1) Markovchain objects and Transition Matrix: The very first approach involved using 'markovchain' package and 'transitionMatrix'. However, mathematical operations on any reasonable size transitionMatrix (4000 X 4000) did not complete on the Windows laptop with 8GB RAM and 4 CPUs. So this approach was not pursued further. [Note: Source code for this approach is not included this document.]

2) Linearly Interpolated tri-gram model: This model is based on Maximum Likelihood Estimates (conditional probabilities) and linear interpolation. More details of this approach can be found in Prof. Michael Collins' course notes (reference 5: Section 1.4.1) This model is defined as given below:

(i) The trigram, bigram, and unigram maximum-likelihood estimates (conditional probabilities) are defined as: Qmle(w|uv) = count(uvw) / count(uv); Qmle(w|v) = count(vw) / count(v); Qmle(w) = count(w) / (count-of-all-unigrams-in-training-corpus).

(ii) In linear interpolation, all three estimates are used to define tri-gram estimate as follows: Q(w|uv) = lambda1 X Qmle(w|uv) + lambda2 X Qmle(w|v) + lambda3 X Qmle(w), where lambda1 >= 0, lambda2 >= 0 and lambda3>= 0 and, lambda1 + lambda2 + lambda3 = 0.

(iii) This model was implemented. However, lambda parameter calculations posed many challenges, and could not be completed due to various reasons. With simple values of lambda parameters, such as, lambda1 = lambda2 = lambda3 = 0.3333, produced mediocre results on test data. Hence a different approach was considered. [Note: Source code for this model is not included in this document.]

3) Good-Turing discounting and Katz back-off model. The theory behind this model is discussed in detail in Chapter 6 - N Grams, of book, Speech and Language Processing by Jurafsky and Martin (reference 6). The required formulae are also listed along-side the code in Section 7.2. In a nutshell, the underlying principles and steps to build this model model are:

(i) 'Count' is defined as the number of times an n-gram occurs in the data. Frequency of occurence of 'Count' is called frequency of freqency (labelled as 'freqCount' in the displayed sample from the tables.) First, Count and frquency of frequency ('freqCount') are calculated. Good Turing discounting is one method which adjusts the Counts of low frquency n-grams, based on the next high frequency Count using the formula formula 6.29 given in Chapter 6 - N-grams (reference 6). Using this formula GTCounts (or counts discounted using Good-Turing method) are calculated.

(ii) So for our training data set, based on Good Turing smoothing, the tri-gram model is created. The raw data is now represented with three n-gram tables. There are 165439 uni-grams, 1561438 bi-grams and 2244766 tri-grams. Note that the number of uni, bi and tri grams is less when compared to the numbers shown in Section 3.1. This is because, all n-grams which occur only once are excluded. Such n-grams are treated in the same manner as unseen n-grams. Sample of rows of the uni, bi and tri gram tables are displayed below to indicate sample values of 'Count' (number of occurences of the n-gram), 'freqCount' (frequency of the Count of the n-gram) and 'GTCount' (Good-Turing discounted Count of the n-gram). [Note: source code for creating this model is included in this document - function - calculateGTCounts of Section 7.2]

```
##                u Count freqCount GTCount
##  1:        flinn     5      8862   4.255
##  2:    flinstone     6      6393   6.000
##  3:        flint    81       121  81.000
##  4:    flintlock     2     50044   1.258
##  5:   flintridge     2     50044   1.258
##  6:       flints     2     50044   1.258
##  7:   flintshire     2     50044   1.258
##  8:   flintstone    17      1195  17.000
##  9:  flintstones    16      1298  16.000
## 10:       flinty     5      8862   4.255
```

```
##                   uv Count freqCount GTCount
##  1:      all traffic     6     63203   6.000
##  2:     all training     3    256375   2.150
##  3:  all transferred     3    256375   2.150
##  4:        all trash     4    141332   3.118
##  5:     all traveler     3    256375   2.150
##  6:    all traveling     4    141332   3.118
##  7:     all treasure     2    627829   1.171
##  8:        all treat     2    627829   1.171
##  9:        all trees     2    627829   1.171
## 10:     all trending     5     89760   4.171
##                     uvw Count freqCount GTCount
##  1:     a single song     4    193977   2.970
##  2:     a single soul    10     24764  10.000
##  3: a single standard     2   1120838   1.008
##  4:     a single step    18      7098  18.000
##  5:    a single story     3    385541   1.988
##  6:   a single strand     3    385541   1.988
##  7:   a single street     3    385541   1.988
##  8:   a single stroke     4    193977   2.970
##  9:        a single t     2   1120838   1.008
## 10:  a single teacher     2   1120838   1.008
```

3.3 Selected model.

1) Evaluation of models using cross validation and perplexity analysis could not be completed in given time.
2) The final model is seleted based on a few tests using sample test data. Based on such simple tests, and based on accuracy of results on the given quizzes, language model using Good-Turing discounting and Katz back off scheme is selected.

4. Predicting Next Words

1) Prediction of next word is done using the data created in the previous Section and applying Katz back off scheme.
2) No matter how large is the training data set, there will always be a large number of unseen (Count = 0) n-grams. For example in our training data set there are 2244766 seen tri-grams, and a total number of 165439 seen unigrams. So number of unseen tri-grams will be [(165439 X 165439 X 165439) - 2244766] a very large number. Discounting is used to assign probabilty mass to unseen n-grams. GTCounts are used to calculate probabilities. These probability values are called discounted probabilities. [Note: Source code is in functions: calculateUgmDiscProb, calculateBgmDiscProb, calculateTgmDiscProb of Section 7.2]
3) Katz back-off for prediction: Back-off means, if there is no example of a particular n-gram, then we use (n-1)-gram. If no example of a (n-1)-gram, then we use (n-2)-gram, and so on. The tri-gram version of the back-off models is represented by the formula 6-30 in chapter 6 - N-grams (reference 6). Here alpha1 and alpha2 are the paramters which ensure that the probabilities of lower level n-grams (for example probabilities of bi-grams and uni-grams) all add up to 1. [Note: Source code is in functions - selectUsingUgm, selectUsingBgm, selectUsingTgm, calculateUgmAlpha, and calculateBgmAlpha or Section 7.2]
4) To demonstrate the use of the model, a prediction function is written. This function accepts the trained data model, sequence of words and word choices as inputs. This function uses underlying model representation to predict the next word in the sequence of words. An example of usage of this prediction function is given below. [Note: Source code is in function predictNextWord of Section 7.2]

```r
sentence1 = "Talking to your mom has the same effect as a hug and helps reduce your"
choices1 = "happiness stress sleepiness hunger"
nextWord1 = predictNextWord(modKatz.l, sentence1, choices1)
nextWord1
```

```
## [1] "stress"
```

5. Notes on the implementation

1) The call to termDocumentMatrix gives lots of isssues if 'dictionary' argument is used. For example, it seems to produce incorrect results in case of generation of uni grams when dictionary argument is used. More details on this are in the source code.

2) It could have been better to calculate and store discounted probabilities of all n-grams and also the alpha paramters for each uni, bi grams, in respective tables. However, such calculations did not finish on Windows 8GB/4 CPU laptop in 24 hours. Hence, calculation of discounted probabilities and alpha parameters is done during prediction time. With this approach prediction takes longer.

3) It takes approx. 4.5 hours to generate this pdf document from the Rmd file on Windows 2003 64bit server with 4 CPUs and 8GB RAM. Most of the time is spent in building the uni, bi and tri gram tables with 'freqCount' and then calculating 'GTCounts' for each n-gram.

6. References

1) Coursera/Columbia University NLP course: https://class.coursera.org/nlangp-001
2) Coursera/Stanford University NLP course: https://class.coursera.org/nlp/lecture/preview
3) http://cran.r-project.org/web/packages/markovchain/vignettes/an_introduction_to_markovchain_package.pdf
4) http://www.aw-bc.com/greenwell/markov.pdf
5) http://www.cs.columbia.edu/~mcollins/lm-spring2013.pdf
6) Book: Speech and Language Processing by Daniel Jurafsky and James H. Martin
7) Book: Text Analysis with R for students of Literature by Matthew L. Jockers
8) http://en.wikipedia.org/wiki/Natural_language_processing
9) http://cran.r-project.org/web/packages/tm/vignettes/tm.pdf
10) Task View: http://cran.r-project.org/web/views/NaturalLanguageProcessing.html
11) Text Mining Infrastructuer in R: http://www.jstatsoft.org/v25/i05/

7. Source Code

```r
### Section 7.1 - Code for preparation of brief summary of raw data.
library(tm)
library(SnowballC)
library(filehash)
library(MASS)
Sys.setenv(JAVA_HOME='C:\\Program Files\\Java\\jre7')
library(RWeka)
library(stringi)
library(Rgraphviz)
library(tau)
library(ggplot2)
library(plyr)
library(data.table)
library(reader)
twtr.lines.v = scan("D:/prashant/coursera-dsc/capstone/final/en_us/en_US.twitter.txt",
                    what = "character", sep = "\n")
news.lines.v = scan("D:/prashant/coursera-dsc/capstone/final/en_us/en_US.news.txt",
                    what = "character", sep = "\n")
blogs.lines.v = scan("D:/prashant/coursera-dsc/capstone/final/en_us/en_US.blogs.txt",
                     what = "character", sep = "\n")
## Create table with file name, no. of lines, no of words/tokens, and word ## types.
lnTwts = length(twtr.lines.v)
lnNews = length(news.lines.v)
lnBlgs = length(blogs.lines.v)
twtr.v = tolower(paste(twtr.lines.v, collapse = " "))
news.v = tolower(paste(news.lines.v, collapse = " "))
blogs.v = tolower(paste(blogs.lines.v, collapse = " "))
twtrWords.v = unlist(strsplit(twtr.v, "\\W"))
newsWords.v = unlist(strsplit(news.v, "\\W"))
blogsWords.v = unlist(strsplit(blogs.v, "\\W"))
twtrWords.v = twtrWords.v[which(twtrWords.v != "")]
newsWords.v = twtrWords.v[which(newsWords.v != "")]
blogsWords.v = twtrWords.v[which(blogsWords.v != "")]
twtrWordTypes.v = unique(twtrWords.v)
newsWordTypes.v = unique(newsWords.v)
blogsWordTypes.v = unique(blogsWords.v)
numTwtrWords = length(twtrWords.v)
numNewsWords = length(newsWords.v)
numBlgsWords = length(blogsWords.v)
numTwtrWordTypes = length(twtrWordTypes.v)
numNewsWordTypes = length(newsWordTypes.v)
numBlgsWordTypes = length(blogsWordTypes.v)
twtrSummary = c("en_US.twitter.txt", lnTwts, numTwtrWords, numTwtrWordTypes)
newsSummary = c("en_US.news.txt", lnNews, numNewsWords, numNewsWordTypes)
blogsSummary = c("en_US.blogs.txt", lnBlgs, numBlgsWords, numBlgsWordTypes)

fileName = c("en_US.twitter.txt", "en_US.news.txt", "en_US.blogs.txt")
numLines = c(lnTwts, lnNews, lnBlgs)
numWords = c(numTwtrWords, numNewsWords, numBlgsWords)
numWordTypes = c(numTwtrWordTypes, numNewsWordTypes, numBlgsWordTypes)
rawDataSummary.df = data.frame(fileName, numLines, numWords, numWordTypes)

### Section 7.2 - Code to build predition model based on Good Turing (GT) discounted counts and
### Katz-back off algorithm.

## Preprocess and use 60% data for training
createNgmsForTraining <- function(data.dir)
{
        # Tweeter file has approx. 2.4M lines, News file has 80K lines and blogs file has 900K lines.
        # TBD - Improve the code to handle 60%.

        twSkip = nwSkip = blSkip = 0
        ffUgm.dt = ffBgm.dt = ffTgm.dt = data.table(mle.cnt = c(0), freq.mle.cnt = c(0), GT.cnt = c(0))
        ugm.dt = data.table(u = c('happy'), Count = c(0))
        bgm.dt = data.table(uv = c('happy new'), Count = c(0))
        tgm.dt = data.table(uvw = c('happy new year'), Count = c(0))

        setkey(ugm.dt, u)
        setkey(bgm.dt, uv)
        setkey(tgm.dt, uvw)

        ## Read 2% at time.
```

```
twtr1p = 2400000 / 50
news1p = 80000 / 50
blgs1p = 900000 / 50

for (i in 1:30)
{
        setwd(data.dir)
        twtrs = n.readLines("en_US.twitter.txt", twtr1p, comment="", skip=twSkip, header=FALSE)
        news11 = n.readLines("en_US.news.txt", news1p, comment="", skip=nwSkip, header=FALSE)
        blogs = n.readLines("en_US.blogs.txt", blgs1p, comment="", skip=blSkip, header=FALSE)

        ## Preprocessing before creating training data set.
        ## Get rid of lines with non-ascii characters.
        twtrs1 = twtrs[-grep ("NonAsciChar", iconv(twtrs, "latin1", sub="NonAsciChar"))]
        news1 = news11[-grep ("NonAsciChar", iconv(news11, "latin1", sub="NonAsciChar"))]
        blogs1 = blogs[-grep ("NonAsciChar", iconv(blogs, "latin1", sub="NonAsciChar"))]

        ## Write these lines to create files in the train directory
        twtrCon1 = file("./Train/en_US.twitter.train.txt", "w")
        writeLines(twtrs1, twtrCon1)
        newsCon1 = file("./Train/en_US.news.train.txt", "w")
        writeLines(news1, newsCon1)
        blogsCon1 = file("./Train/en_US.blogs.train.txt", "w")
        writeLines(blogs1, blogsCon1)
        close(twtrCon1)
        close(newsCon1)
        close(blogsCon1)

        train.dir = "D:/prashant/coursera-dsc/capstone/final/en_us/Train"
        corp1.corpus = preprocAndCreateCorpus(train.dir)
        ngrams.l = createDifferentNgrams(corp1.corpus)


        uni.gm1.df = as.data.frame(as.matrix(ngrams.l[[1]]))
        bi.gm1.df = as.data.frame(as.matrix(ngrams.l[[2]]))
        tri.gm1.df = as.data.frame(as.matrix(ngrams.l[[3]]))
        ## Add a column called Count to each of the above data frames and sort on Count.
        uni.gm1.df$Count = rowSums(uni.gm1.df[,1:3])
        uni.gm1.df = uni.gm1.df[order(-uni.gm1.df$Count), ]
        bi.gm1.df$Count = rowSums(bi.gm1.df[,1:3])
        bi.gm1.df = bi.gm1.df[order(-bi.gm1.df$Count), ]
        tri.gm1.df$Count = rowSums(tri.gm1.df[,1:3])
        tri.gm1.df = tri.gm1.df[order(-tri.gm1.df$Count), ]


        ## Remove columns with document names and add a column with word(s) or n-grams
        uni.gm1.df$u = rownames(uni.gm1.df)
        uni.gm2.df = uni.gm1.df[,4:5]
        uni.gm2.df = uni.gm2.df[c(2,1)]

        bi.gm1.df$uv = rownames(bi.gm1.df)
        bi.gm2.df = bi.gm1.df[,4:5]
        bi.gm2.df = bi.gm2.df[c(2,1)]
```

```
                tri.gm1.df$uvw = rownames(tri.gm1.df)
                tri.gm2.df = tri.gm1.df[,4:5]
                tri.gm2.df = tri.gm2.df[c(2,1)]

                twSkip = twSkip + twtr1p
                nwSkip = nwSkip + news1p
                blSkip = blSkip + blgs1p


                ugm.dt = rbind(ugm.dt, uni.gm2.df)
                ugm.dt = ugm.dt[, lapply(.SD, sum), by = u]
                bgm.dt = rbind(bgm.dt, bi.gm2.df)
                bgm.dt = bgm.dt[, lapply(.SD, sum), by = uv]
                tgm.dt = rbind(tgm.dt, tri.gm2.df)
                tgm.dt = tgm.dt[, lapply(.SD, sum), by = uvw]
                setkey(ugm.dt, u)
                setkey(bgm.dt, uv)
                setkey(tgm.dt, uvw)
        }
        return(c(list(ugm.dt, bgm.dt, tgm.dt)))
}

# Function to build a Corpus object from the training data set given the directory
preprocAndCreateCorpus <- function(inp.dir)
{
        corp1=Corpus(DirSource(inp.dir, encoding = "UTF-8"),
                     readerControl=list(reader=readPlain, language="en"))
        ## Create a swear word list.
        ## List obtained from http://www.bannedwordlist.com/lists/swearWords.txt
        con1 = file("D:/prashant/coursera-dsc/capstone/swearWords.txt", 'r')
        swList = readLines(con1)
        close(con1)

        corp1Tr = tm_map(corp1, stripWhitespace)
        corp1Tr = tm_map(corp1Tr, removePunctuation) ## Ref Page 192 JM book
        corp1Tr = tm_map(corp1Tr, removeNumbers)
        corp1Tr = tm_map(corp1Tr, content_transformer(tolower))
        ##corp1Tr = tm_map(corp1Tr, removeWords, stopwords("english"))

        ## Remove swear words.
        corp1Tr = tm_map(corp1Tr, removeWords, swList)
        return(corp1Tr)
}

## Function to create uni, bi, and tri grams from the specified corpus.
createDifferentNgrams <- function(inp.corpus)
{
        ## Use words only from a dictionary. This should reduce the size.
        ##dcon1 = file("D:/prashant/coursera-dsc/capstone/EN-dictionary/EnglishWordlist.txt", "r")
        ##wrdList = readLines(dcon1)
        ##close(dcon1)

        ### Attempts to use dictionary argument to filter out mis-spelt words were
        ### not successful. Hence no dictionary is used.

        #tdm1 = TermDocumentMatrix(inp.corpus,
        #                   control = list(control=list(wordLengths=c(1,Inf)),
        #                   bounds = list(global = c(1,3))))

        ### The call to TDM gives lots of isssues if dictionary argument is used.
        ### In case of generation of uni grams (with dictionary argument).
        ### It does not generate tokens with 1 or 2 characters.

        tdm1 = TermDocumentMatrix(inp.corpus, control=list(wordLengths=c(1,Inf)))

        BgmTkns <- function(x) NGramTokenizer(x, Weka_control(min = 2, max = 2))
        tdmBgms1 <- TermDocumentMatrix(inp.corpus, control = list(tokenize = BgmTkns))

        ## Using dictionary on bi-grams does not work. So bi,tri grams
        ## will have mis-spelt words.
        ##tdmBgms11 <- TermDocumentMatrix(inp.corpus, control = list(tokenize = BgmTkns,
        ##                              dictionary = wrdList, bounds = list(global = c(1,3))))

        ##tdmBgms1a <- TermDocumentMatrix(inp.corpus, control = list(tokenize = BgmTkns,
        ##                bounds = list(global = c(2,3))))

        TgmTkns <- function(x) NGramTokenizer(x, Weka_control(min = 3, max =  3))
        tdmTgms1 <- TermDocumentMatrix(inp.corpus, control = list(tokenize = TgmTkns))
        ##tdmTgms1a <- TermDocumentMatrix(inp.corpus, control = list(tokenize = TgmTkns, bounds = list(

        return(c(list(tdm1, tdmBgms1, tdmTgms1)))
}

## Function to calculate Good-Turing counts for given n-gram using Count.
calculateGTCounts <- function(inp.dt, k, ffcnt1)
{
        inp.dt = inp.dt[order(inp.dt$Count), ]
        inp.dt$freqCount = 0
        inp.dt$GTCount = 0

        ## Make list of all unique mle counts, and sort it.
        mleCounts = sort(unique(inp.dt$Count))
        for (i in 1:length(mleCounts))
        {
                inp.dt[inp.dt$Count == mleCounts[i], ]$freqCount =
                        dim(inp.dt[inp.dt$Count == mleCounts[i], ])[1]
        }
        ## For first k values of mle Count, adjust the mle Count and the
        ## the adjusted value is set in GTCount column.
        ## For values of mle Count more than k, GT count is set to mle Count.

        for (i in 1:k-1)
        {
                # Calculate the GT Counts

                numr1 = mleCounts[i+1] * (inp.dt[inp.dt$Count == mleCounts[i+1], ]$freqCount[1] /
                                                      inp.dt[inp.dt$Count == mleCounts[i], ]$freqCount[1])

                numr2 = mleCounts[i] * ((k+1) *
                                (inp.dt[inp.dt$Count == mleCounts[k+1], ]$freqCount[1] / ffcnt1))

                denmr = 1 - ((k+1) * inp.dt[inp.dt$Count == mleCounts[k+1], ]$freqCount[1] / ffcnt1)
                inp.dt[inp.dt$Count == mleCounts[i], ]$GTCount = (numr1 - numr2) / denmr

        } ## For the rest set GT Count same as mle Count.
        for (i in k:length(mleCounts))
        {
                inp.dt[inp.dt$Count == mleCounts[i], ]$GTCount = mleCounts[i]
        }
        return(inp.dt)
}

predictNextWord <- function(m.df.l, stmt, opts)
{

        uMod.dt = m.df.l[[1]]
        bMod.dt = m.df.l[[2]]
        tMod.dt = m.df.l[[3]]

        setkey(uMod.dt, u)
        setkey(bMod.dt, uv)
        setkey(tMod.dt, uvw)

        stmt.tkns = tokenize(stmt)
        opts.tkns = tokenize(opts)
        tkns1 = stmt.tkns[stmt.tkns != " "]
        len1 = length(tkns1)
        u = tkns1[len1 - 1]
        v = tkns1[len1]

        w1 = opts.tkns[1]
        w2 = opts.tkns[3]
        w3 = opts.tkns[5]
        w4 = opts.tkns[7]

        tgm1 = paste(u, v, w1, sep=" ")
        bgm1 = paste(v, w1, sep=" ")
        ugm1 = w1
        tgm2 = paste(u, v, w2, sep=" ")
        bgm2 = paste(v, w2, sep=" ")
        ugm2 = w2
        tgm3 = paste(u, v, w3, sep=" ")
        bgm3 = paste(v, w3, sep=" ")
        ugm3 = w3
        tgm4 = paste(u, v, w4, sep=" ")
        bgm4 = paste(v, w4, sep=" ")
        ugm4 = w4
        predWord = 'NOTSELECTED'
```

```r
        predWord = selectUsingTgm(tMod.dt, bMod.dt, tgm1, tgm2, tgm3, tgm4, w1, w2, w3, w4)
        if (predWord == 'NOTSELECTED'){
                predWord = selectUsingBgm(bMod.dt, uMod.dt, tMod.dt, bgm1, bgm2, bgm3,
                                          bgm4, w1, w2, w3, w4)
        }
        if (predWord == 'NOTSELECTED'){
                predWord = selectUsingUgm(uMod.dt, bMod.dt, ugm1, ugm2, ugm3, ugm4)
        }

        return(predWord)
}

### Calculate the p.hat probability
## Use the formula based on Katz back-off,
## P-hat(w|uv) = discProb(w|uv), if discProb(w|uv) > 0
## P-hat(w|uv) = discProb(w|v) * Alpha, if discProb(w|uv) = 0 and
##                                      if discProb(w|v) > 0
## P-hat(w|uv) = discProb(w) * Alpha,    otherwise,
## to calculate the P-hat values for each tri-gram.

## Select the choice based on the tgm, if any of the four tgms is found in our model.
selectUsingTgm <- function(tg.dt, bg.dt, tg1, tg2, tg3, tg4, word1, word2, word3, word4)
{
        predictedWord = 'NOTSELECTED'
        p.hat.uvw1 = p.hat.uvw2 = p.hat.uvw3 = p.hat.uvw4 = 0
        if (length(tg.dt[tg.dt$uvw == tg1, ]$GTCount) == 1) {
                p.hat.uvw1 = calculateTgmDiscProb(tg.dt, bg.dt, tg1)
                predictedWord = word1 ## Start with w1, then change based on p.hat value
        }
        if (length(tg.dt[tg.dt$uvw == tg2, ]$GTCount) == 1) {
                p.hat.uvw2 = calculateTgmDiscProb(tg.dt, bg.dt, tg2)
                if (p.hat.uvw2 > p.hat.uvw1) { predictedWord = word2}
        }
        if (length(tg.dt[tg.dt$uvw == tg3, ]$GTCount) == 1) {
                p.hat.uvw3 = calculateTgmDiscProb(tg.dt, bg.dt, tg3)
                if (p.hat.uvw3 > max(p.hat.uvw1, p.hat.uvw2)) { predictedWord = word3}
        }
        if (length(tg.dt[tg.dt$uvw == tg4, ]$GTCount) == 1) {
                p.hat.uvw4 = calculateTgmDiscProb(tg.dt, bg.dt, tg4)
                if (p.hat.uvw4 > max(p.hat.uvw1, p.hat.uvw2, p.hat.uvw3)) { predictedWord = word4}
        }
        return(predictedWord)
}

## Select the choice based on the bgm, if any of the four bgms is found in our model.
selectUsingBgm <- function(bg.dt, ug.dt, tg.dt, bg1, bg2, bg3, bg4, word1, word2, word3, word4)
{
        predictedWord = 'NOTSELECTED'
        p.hat.uvw1 = p.hat.uvw2 = p.hat.uvw3 = p.hat.uvw4 = 0
        if (length(bg.dt[bg.dt$uv == bg1, ]$GTCount) == 1) {
                p.hat.uvw1 = calculateBgmDiscProb(bg.dt, ug.dt, bg1) *
                        calculateBgmAlpha(ug.dt, bg.dt, tg.dt, bg1)
                predictedWord = word1 ## Start with w1, then change based on p.hat value
        }
        if (length(bg.dt[bg.dt$uv == bg2, ]$GTCount) == 1) {
                p.hat.uvw2 = calculateBgmDiscProb(bg.dt, ug.dt, bg2) *
                        calculateBgmAlpha(ug.dt, bg.dt, tg.dt, bg2)
                if (p.hat.uvw2 > p.hat.uvw1) { predictedWord = word2}
        }
        if (length(bg.dt[bg.dt$uv == bg3, ]$GTCount) == 1) {
                p.hat.uvw3 = calculateBgmDiscProb(bg.dt, ug.dt, bg3) *
                        calculateBgmAlpha(ug.dt, bg.dt, tg.dt, bg3)
                if (p.hat.uvw3 > max(p.hat.uvw1, p.hat.uvw2)) { predictedWord = word3}
        }
        if (length(bg.dt[bg.dt$uv == bg4, ]$GTCount) == 1) {
                p.hat.uvw4 = calculateBgmDiscProb(bg.dt, ug.dt, bg4) *
                        calculateBgmAlpha(ug.dt, bg.dt, tg.dt, bg4)
                if (p.hat.uvw4 > max(p.hat.uvw1, p.hat.uvw2, p.hat.uvw3))
                        { predictedWord = word4}
        }
        return(predictedWord)
}

## Select the choice based on the ugm.
selectUsingUgm <- function(ug.dt, bg.dt, ug1, ug2, ug3, ug4)
{
        predictedWord = 'NOTSELECTED'
        p.hat.uvw1 = p.hat.uvw2 = p.hat.uvw3 = p.hat.uvw4 = 0
        if (length(ug.dt[ug.dt$u == ug1, ]$GTCount) == 1) {
                p.hat.uvw1 = calculateUgmDiscProb(ug.dt, ug1) *
                        calculateUgmAlpha(ug.dt, bg.dt, ug1)
                predictedWord = ug1 ## Start with w1, then change based on p.hat value
        }
        if (length(ug.dt[ug.dt$u == ug2, ]$GTCount) == 1) {
                p.hat.uvw2 = calculateUgmDiscProb(ug.dt, ug2) *
                        calculateUgmAlpha(ug.dt, bg.dt, ug2)
                if (p.hat.uvw2 > p.hat.uvw1) { predictedWord = ug2}
        }
        if (length(ug.dt[ug.dt$u == ug3, ]$GTCount) == 1) {
                p.hat.uvw3 = calculateUgmDiscProb(ug.dt, ug3) *
                        calculateUgmAlpha(ug.dt, bg.dt, ug3)
                if (p.hat.uvw3 > max(p.hat.uvw1, p.hat.uvw2))
                        { predictedWord = ug3}
        }
        if (length(ug.dt[ug.dt$u == ug4, ]$GTCount) == 1) {
                p.hat.uvw4 = calculateUgmDiscProb(ug.dt, ug4) *
                        calculateUgmAlpha(ug.dt, bg.dt, ug4)
                if (p.hat.uvw4 > max(p.hat.uvw1, p.hat.uvw2, p.hat.uvw3))
                        { predictedWord = ug4}
        }

        ## If still the word is not selected, then set it to first word.
        if (predictedWord == 'NOTSELECTED') { predictedWord = ug1 }
        return(predictedWord)
}
```

```r
## For the specified tgm calculate the discounted probability.
calculateTgmDiscProb <- function(tri.dt, bi.dt, givenTgm)
{
        ## For the tri gram, get the previous two words i.e. bi-gram
        ## Use the bi-gram to get total number of its occurances from bi-gram dt.
        ## discProb = giveTgm.GT.cnt/num-respctive-bgm
        givenTgm.GT.cnt = tri.dt[givenTgm]$GTCount

        i.tgm = strsplit(givenTgm, split = " ")[[1]]
        i.bgm = paste(i.tgm[1], i.tgm[2])
        # Get bgm count
        count.i.bgm = bi.dt[i.bgm]$Count
        discProb = givenTgm.GT.cnt/count.i.bgm

        return(discProb)
}

## For the specified bgm calculate the discounted probability.
calculateBgmDiscProb <- function(bi.dt, uni.dt, givenBgm)
{
        ## For the bi gram, get the previous word i.e. uni-gram
        ## Use the uni-gram to get total number of its occurances from uni-gram dt.
        ## discProb = giveBgm.GT.cnt/num-respctive-ugm
        givenBgm.GT.cnt = bi.dt[givenBgm]$GTCount

        i.bgm = strsplit(givenBgm, split = " ")[[1]]
        i.ugm = i.bgm[1]
        # Get ugm count
        count.i.ugm = uni.dt[i.ugm]$Count
        discProb = givenBgm.GT.cnt/count.i.ugm

        return(discProb)
}

## For the specified ugm calculate the discounted probability.
calculateUgmDiscProb <- function(uni.dt, givenUgm)
{
        ## Ugm discounted probability = ugm.GT.cnt/num-ugm
        nUgms = sum(uni.dt$Count)
        # For the mle.cnt of the ugm, get the GT.cnt.
        givenUgm.GT.cnt = uni.dt[givenUgm]$GTCount
        discProb = givenUgm.GT.cnt/nUgms
        return(discProb)
}

### alphaBg = (1 - sum of discounted probabilities of all seen trigrams based on this bigram)
### divided by
### (1-sum of discounted probabilties of all seen bi-grams which are based on the uni-gram
### for this bigram.)
### Calculate alpha parameter for the specified bgm
calculateBgmAlpha <- function(uni.dt, bi.dt, tri.dt, givenBgm)
{
        alpha = 1
        blank = ' '
        tgmsDiscProb = 0 ## Sum of probabilities of all tgms based on this bgm.
        ## Get a list of tri-grams based on the givenBgm
        givenBgmBlank = paste(givenBgm, blank, sep="")
        tgms.l = tri.dt[grepl(paste('^', givenBgmBlank, sep=""), tri.dt$uvw) == TRUE, ]$uvw

        if (length(tgms.l != 0)) {
                for (i in 1:length(tgms.l))
                {
                        tgmsDiscProb = tgmsDiscProb + calculateTgmDiscProb(tri.dt, bi.dt, tgms.l[i])
                }
        }


        bgmsDiscProb = 0 ## Sum of probabilities of all bgms with same uni gram as givenBgm
        givenBgmUgm = strsplit(givenBgm, split= " ")[[1]][1]
        givenBgmUgmBlank = paste(givenBgmUgm, blank, sep="")
        bgms.l = bi.dt[grepl(paste('^', givenBgmUgmBlank, sep=""), bi.dt$uv) == TRUE, ]$uv


        if (length(bgms.l != 0)) {
                for (i in 1:length(bgms.l))
                {
                        bgmsDiscProb = bgmsDiscProb + calculateBgmDiscProb(bi.dt, uni.dt, bgms.l[i])
                }
        }

        alpha = (1 - tgmsDiscProb)/(1 - bgmsDiscProb)

        return(alpha)
}

### alphaUg = (1 - sum of discounted probabilities of all seen bi-grams based on this unigram)
### divided by
### (1-sum of discounted probabilty of this unigram.)
### Calculate alpha parameter for the specified ugm
calculateUgmAlpha <- function(uni.dt, bi.dt, givenUgm)
{
        alpha = 1
        blank = ' '
        bgmsDiscProb = 0 ## Sum of probabilities of all bgms based on this ugm.
        ## Get a list of bi-grams based on the givenUgm
        givenUgmBlank = paste(givenUgm, blank, sep="")
        bgms.l = bi.dt[grepl(paste('^', givenUgmBlank, sep=""), bi.dt$uv) == TRUE, ]$uv
        if (length(bgms.l != 0)) {
                for (i in 1:length(bgms.l))
                {
                        bgmsDiscProb = bgmsDiscProb + calculateBgmDiscProb(bi.dt, uni.dt, bgms.l[i])
                }
        }
        ugmDiscProb = calculateUgmDiscProb(uni.dt, givenUgm)
        alpha = (1 - bgmsDiscProb)/(1 - ugmDiscProb)
        return(alpha)
```

```r
}

train.dir = "D:/prashant/coursera-dsc/capstone/final/en_us/Train"
inpData.dir = "D:/prashant/coursera-dsc/capstone/final/en_us/"
ngmTables.l = createNgmsForTraining(inpData.dir)
ugm20.dt = ngmTables.l[[1]]
bgm20.dt = ngmTables.l[[2]]
tgm20.dt = ngmTables.l[[3]]

### Remove the entries with Count (frequency 1) uni, bi and tri grams.
# Counts with frequency 1 are treated as unseen in the GT discounting model.
# So mle.cnt of 1 is not seen.
## However note down the frequency of mle count 1.
uffCount1 = dim(ugm20.dt[ugm20.dt$Count == 1, ])[1]
bffCount1 = dim(bgm20.dt[bgm20.dt$Count == 1, ])[1]
tffCount1 = dim(tgm20.dt[tgm20.dt$Count == 1, ])[1]

ugm.dt = ugm20.dt[ugm20.dt$Count > 1, ]
bgm.dt = bgm20.dt[bgm20.dt$Count > 1, ]
tgm.dt = tgm20.dt[tgm20.dt$Count > 1, ]

kFactor=5 ## After this value of mleCount, GTCount = mleCount.
ugm1a.dt = calculateGTCounts(ugm.dt, kFactor, uffCount1)
bgm1a.dt = calculateGTCounts(bgm.dt, kFactor, bffCount1)
tgm1a.dt = calculateGTCounts(tgm.dt, kFactor, tffCount1)

setkey(tgm1a.dt, uvw)
setkey(bgm1a.dt, uv)
setkey(ugm1a.dt, u)
modKatz.l = list(ugm1a.dt, bgm1a.dt, tgm1a.dt)
```