Jared Yu

STA 141C

Homework 2

1. The first step of the task was to utilize the learned techniques from the discussion class to properly filter out each description row of a file. This involved using the 'tm' package to understand how best to split the description column into a useful character vector. There was apparently a small mistake, where tolower() was used after the removeWords() for stopwords(). This prevented any of the stop words from being removed because they are all stored in lower case. Also, after some quick online searching, it was found out that stopwords() had an additional input called 'SMART' which includes important stop words such as 'for'. Using roughly the discussion format, the first step was to remove punctuation from the description using gsub() with the regex [[:punct:]]. Afterwards the description was converted to lower case using tolower(). This makes it possible for the next step to function properly which is removeWords() with stopwords() 'en' and 'SMART.' These would take care of the simple ordinary words such as 'and' or 'for' which are not helpful keywords for categorizing what an expense is. Then the stemDocument() function was used to match together similar words by breaking them down to their stem portions. The last part was to both change double white spaces to single white spaces with stripWhitespace() and gsub() with a regex expression to remove leading and trailing whitespaces. The former part made it easier to identify keywords later on using strsplit() while looking for single spaces, and the latter makes it possible to properly count the length of a string of keywords. Leading and trailing whitespaces could account for larger than actual lengths.

   There was an attempt to try and understand if it was possible to increase the efficiency of running consecutive functions such as the ones that were demonstrated above. However, the approach of calling functions in a logical order already seems to be the optimal method. No alternative was discovered through a quick online research. None of the functions are called in any sort of for loop, so it is likely that they are already being processed as fast as possible.

2. After running the script, it became obvious that the process was far too slow. It was learned from the professor's office hours, that the goal was to create a character vector per row. Each row had already been split into strings with spaces between each 'phrase.' The next step was to assign the proper weighting based on the spending for that item. This was initially done using an lapply() along with some other commands, but the TA during office hours mentioned that Map() was a more suitable function. Therefore, a new function character_weights() was written to go with the Map() to split the string of phrases into a data frame with a column for keywords and a column for weighted prices. The TA had said that using lapply() on data which was not in list form was similar to using a for loop.

   The initial part of creating the weighted words was not overly complicated, but it was the debugging process while going through some test files that proved to be most difficult. During the initial running of the script, certain files would take noticeably longer than others. One of the files put into question the usefulness of the description being <NA> for certain items. The strategy to deal with these was to ignore items with such descriptions, as there was no clear

method to assign any weights when there is no explanation of what the money is for. Another incident is finding a file with only one item and an NA description. This would also cause an error as this case can make the function to break down as there is insufficient valid information going in to properly give an output.

It was realized that the process was going to be far too slow. To go through a single file, it could take quite a few minutes. So, to complete all 91 files it would possibly take a few hours. An initial step was to try and print the file that the script was working on at a time. This would make it possible to understand which files were problematic in the process. Using the file name, the file would then be read line by line through the pieces of the function. This process gave a rough gauge of how long it would take for the file to go through certain parts of the weighting. Doing this gave a surprising result, when going through the file individually, errors would show up that had not occurred before when the function was running in the background. One of the first errors showed that the initial removal of blank spaces was insufficient, leading to problems in matching row lengths. Another error was the existence of 0's and Inf's values. These were dealt with by removing 0's and making sure it was not possible to divide by length 0. The assumption here is that the few instances of when these bugs were happening was not significant enough to be worrying towards the overall goal of assigning weights.

Despite having code which seemed to run correctly, an issue still existed of code efficiency. The script would still take quite a long time to run each file, and the professor had said that it could take from 30 minutes to 1 hour. However, in my case it would have possibly taken still several hours to complete the task. Removing irrelevant cases such as 0's and Inf's for spending reduced the time it would take to read some rows; however it was still not enough to deal with larger files. A more vectorized approach was used to perform certain calculations that initially would take several seconds to complete. The initial Map() process was changed so that portions such as calculating the weights for each keyword was done beforehand in a vectorized manner. This made the character_weights() function which was created earlier unnecessary. Additionally, the stringsAsFactors was set to TRUE so that the data.frame() function would not automatically turn keywords into factors, a process that takes up unnecessary time. Another trick was using fixed=TRUE for strsplit(). Before do.call() was used with 'rbind' on a large list, instead Map() was used with 'cbind' on the keywords and their prices to create a long matrix. Afterwards, do.call() was used with 'rbind' this time on a far simpler piece of data structure which would be done far quicker. Initially the data.table package was tried out utilizing rbindlist(), which is an optimized version of do.call with rbind. However, this was not necessary in the end. These changes made it possible to take advantage of R's ability to write vectorized code. Doing so made a significant difference in the processing speed of each individual file.

After the experience in the difference of timing using a vectorized approach, it seems that the only way to further optimize the speed would be to try and do more vectorization within the programming. It seems that the initial understanding about the apply functions was not entirely correct. They still seem to utilize some style of for loop within their C-language function, and can therefore still be troublesome. Looking deeper into the calculations allows for insight into when

vectorization can be used to make speed faster. Also, subtle details such as stringsAsFactors can further be found to make it so that there is less wasted computation time.

3. Below are tables of the specific agencies and the top 25 weighted words. The format for the tables is that the first row is the keyword, and the second row is the weighted price for each keyword. The proportion that has been calculated is only the proportion out of the top 25, however. This pattern repeats for the third and fourth row, along with the fifth and sixth row, etcetera. Additionally, the keywords have been highlighted in yellow.

National Science Foundation (655)

research	oper	program	servic	center	manag	observatori	support
0.108	0.095	0.073	0.058	0.051	0.049	0.045	0.044
scienc	nation	fellowship	graduat	grfp	igf	integr	mainten
0.042	0.039	0.036	0.036	0.029	0.028	0.028	0.028
contract	mod	stand	054	collabor	system	comput	ocean
0.027	0.026	0.026	0.026	0.025	0.022	0.021	0.019
engin							
0.019							

Federal Bureau of Investigation (262)

igf	equip	servic	hardwar	ot	ammunit	softwar	support	0200	mainten
0.207	0.101	0.099	0.082	0.080	0.067	0.064	0.040	0.039	0.028
toner	laptop	cisco	train	tactic	suppli	ct	cl	tor	scanner
0.021	0.018	0.016	0.015	0.014	0.014	0.013	0.012	0.011	0.011
copier	server	comput	itap	printer					
0.011	0.010	0.010	0.010	0.010					

U.S. Customs and Border Protection (778)

igf	servic	requir	copier	comput	chief	parti	ot	softwar
0.131	0.101	0.082	0.052	0.050	0.050	0.050	0.049	0.046
mainten	support	radio	equip	investig	transmiss	overhaul	ess	offic
0.045	0.043	0.028	0.028	0.028	0.025	0.025	0.022	0.021
train	function	f5	furnitur	laptop	19	background		
0.020	0.019	0.018	0.017	0.017	0.016	0.016		

Forest Service (110)

circuit	servic	usf	helicopt	737	boe	bois	id	type
0.270	0.070	0.053	0.052	0.049	0.049	0.049	0.049	0.047
call	need	cwn	report	q4	fy05	support	technolog	offic
0.043	0.043	0.030	0.029	0.028	0.027	0.016	0.016	0.016
ii	access	retrofit	tnb	culvert	replac	laboratori		
0.015	0.011	0.011	0.011	0.008	0.006	0.005		

4. In the first agency, the National Science Foundation, it seems to have keywords that match the name of the agency. Keywords such as: 'research,' 'observatori,' 'scienc,' 'fellowship,' 'graduat,' and 'ocean' all seem quite typical of scientific research. In this agency then it seems that the keywords are consistent with the name and purpose of the agency. Also, 'research' has the greatest weight out of the top 25, which makes sense for a scientific agency. Other words which have greater weight that seem to be of greater worth such as: 'oper' (operation), 'center,' and 'manag' (management).

   In the second agency, the Federal Bureau of Investigation, it also seems to have keywords that match the name of the agency. Keywords such as: 'hardwar,' 'ammunit,' 'softwar,' 'mainten,' 'toner,' 'laptop,' 'cisco,' 'scanner,' 'copier,' 'server,' 'comput,' and 'printer.' It is interesting to note that much of the spending seems to be focused on items that are typical of any company which has people writing reports and communicating them with others. It is also possible that the spending is so high in these normal everyday office expenses because the FBI needs to have regularly updated equipment which can be costly for an organization so large. It is also interesting that 'igf' has the greatest weight, double that of the next keyword. IGF is an inherently governmental function, and is an unspecified expense, but it is possible that the FBI does many things that aren't specifically mentioned due to them being 'top secret' or 'classified.' However, this is merely speculation.

   The third agency, the U.S. Customs and Border Protection, also has certain keywords which may be typical of such an organization: 'copier,' 'comput,' 'softwar,' 'mainten,' 'radio,' 'investig,' 'office,' and 'laptop.' Once again, like with the FBI, much of the purchases seem focused on regular office supplies. It's possible that the agency went through similar technology updates like the FBI, where it had to purchase newer hardware and software to keep up with changing equipment.

   In the fourth agency, the Forest Service, has possibly fewer keywords that seem to indicate that it belongs to anything related to the Forest Service. Keywords such as 'helicopt,' '737,' and 'boe' indicate that the agency has purchase helicopters and Boeing 737's, highly expensive items which could easily make it to the top of the list. It is also not impossible that these items were purchased for fighting forest fires. Helicopters and airplanes have been used to dump water on forest fires, and this could possibly be the reason. There are far less 'office equipment' items like in the previous two agencies. Also, it is not easy to identify keywords which may indicate what the agency is doing. However, a Forest Service agency is also not any typical, easy to interpret case, where their daily operations can be understood simply. It is also interesting to note that 'circuit' is so heavily weighted in comparison to other keywords. After doing some research into the file, it seems that the costs come mostly from a few AT&T Circuits which are considerably more expensive than other items. The total cost of the circuits is over a million dollars, while the median cost of an expense was merely $17,000.

   To make the understanding clearer, it would possibly be useful to investigate agencies themselves and have a better understanding of the costs of their usual operations. When

examining what sort of items, they use which are costly or frequently used, it may be better to see if the weighted character vectors are useful in explaining what type of agency is being examined. Without knowledge of the patterns of an agency, it is hard to decipher what certain keywords can be indicative of.

It was noticed that a common description is IGF::OT::IGF. There seems to be variations, but it stands for inherently governmental functions. The OT can change, but describes a type of function, in this case it merely stands for other functions. Many of the purchases are not quite accounted for clearly and are described in a brief manner. This seems to be a common practice, however.

Also, an initial belief was that it would be useful to remove the numbers, however looking at instances when they show up in the top 25 show that it is not necessary. Although there are instances when it is not clear, it potentially does not outweigh instances of when using numbers can be indicative of an item. An example would be the '737' as a Boeing airplane. Many of the top keywords are also several letters that are difficult to understand their meaning. It is possible that certain combinations of letters are used frequently in certain files, and they have a meaning which is not explicit in the description. Also, a concern is that the stemming process removes too much of a word, making it hard to understand the meaning afterwards.

5. The parallel package allows for parallel processing in RStudio. Normally, when using a regular lapply(), this would be similar to running a process using a single core of a computer. The laptop used for this assignment has 8 cores, and parallel processing can take advantage of such a situation. There is an overhead process, where a 'manager' splits up a task into different parts and sends them to the 'workers.' These workers, or clusters, are the cores that are assigned to do a task that does not depend on the order from which it is done. This assignment is such a case, where files are processed one-by-one where order is not an issue. It can be said that situations can be 'embarrassingly parallel,' this assignment likely fitting into such a circumstance. The overhead process completes when the 'workers' send the data back to the 'manager.' The parallel process does not actually speed up any of the individual process, it merely distributes it such that the process is split between groups of cores. The parLapply() and clusterApplyLB() (instead of parLapplyLB, according to the professor in Piazza @76) were used to perform the parallel processing. The clusterApplyLB differs in that it has a load balancing factor. This works by balancing the tasks which are sent to the cores so that they are always busy if possible. With different sized files, it would be possible for certain workers to be done, while others are still busy processing large files. The load balancing would offset this by minimizing the wait that workers will do. In both cases of parallel processing, the results were completed faster than with the serial lapply. Below is a table of the times for the three different apply functions:

$Function$	$Elapsed\ Time\ (seconds)$
$lapply$	477.19
$parLapply$	195.92
$clusterApplyLB$	186.67

It is also evident that the load balancing version was even faster than the parLapply which doesn't use the load balancing factor. It seems to make sense then to use the load balancing version due to the varying size of the files that are being asked to be processed. There is also the possibility that if there were more than 91 files within the range, the difference would increase between parLapply and clusterApplyLB. The reason is that there is overhead time, and the penalty of it may decrease if the LB had more files to show its usefulness.

References:

https://stackoverflow.com/questions/2261079/how-to-trim-leading-and-trailing-whitespace

https://stackoverflow.com/questions/8920145/count-the-number-of-all-words-in-a-string

https://stackoverflow.com/questions/13043928/selecting-rows-where-a-column-has-a-string-like-hsa-partial-string-match

https://stackoverflow.com/questions/22678203/fast-way-to-split-string-and-convert-to-long-format-in-data-table

https://stackoverflow.com/questions/7031127/data-frames-and-is-nan

https://www.rdocumentation.org/packages/tm/versions/0.7-6/topics/stopwords

https://computing.llnl.gov/tutorials/parallel_comp/#DesignLoadBalance

https://stat.ethz.ch/R-manual/R-devel/library/parallel/html/clusterApply.html

https://piazza.com/class/jqmje0ujwrm2wx?cid=76

https://www.rdocumentation.org/packages/parallel/versions/3.5.2/topics/clusterApply

https://www.fpds.gov/fpdsng_cms/index.php/en/newsroom/108-nherently-governmental-functions.html

http://gforge.se/2015/02/how-to-go-parallel-in-r-basics-tips/

'Advanced R', Hadley Wickham

'The Art of R Progamming', Norman Matloff