University of Dublin

TRINITY COLLEGE

Using data manipulation to see the impact on Word Sense Disambiguation using an EM algorithm

David Hughes
B.A. (Mod.) Computer Science, Linguistics and a Language
Final Year Project April 2019
Supervisor: Dr. Martin Emms

School of Computer Science and Statistics

O'Reilly Institute, Trinity College, Dublin 2, Ireland

Declaration

I hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: _____ Date: _____

Abstract

In this project, data from a Google 5-gram corpus of the target word mouse will be manipulated in ways such as the filtering of stop words and removal of data to see the impact these procedure have on finding the sense of mouse. The sense of the target word will be determined using an expectation maximisation algorithm which will assign probabilities to each sense given a year. The results of the EM algorithm will be plotted and an inspector program will be used to see the words that occur most frequently with each sense. By doing this the sense of the word can be determined and the results of the filtered process will be compared to the unfiltered starting corpus.

Acknowledgements

I would like to sincerely thank my supervisor, Dr. Martin Emms, for his guidance, support and knowledge in helping me to complete this project. His help has proved invaluable throughout this project.

I would also like to thank my family for their constant support and encouragement. I could not have done this without them.

Contents

List of Figures

List of Tables

1 Introduction

The linguist John Rupert Firth famously said "You shall know a word by the company it keeps" (1). What he meant by this was that by looking at the words around a certain target word you will be able to gain an understanding of the meaning of that word. Not all words, however, add to the meaning of a word or help our understanding of a words meaning. The words that have the largest effect on the meaning of a word are so-called content words. Words which do not have much of an effect on the meaning of a word are so-called stop words like "the" or "a". By removing these stop words, this projects seeks to see if it is still possible to get a clear idea of when a new sense has emerged.

As well as this, over time languages change and with that so too do the senses of words. New words, that previously did not appear alongside a word hundreds of years ago, may do so now. In the case of the word mouse, which will be the focus of this project, such a change in sense may come about because of technological advances.

In this project, the sense of the word mouse over a period of over 100 years will be analyzed using an Expectation Maximization algorithm and inspector program to see when new senses may have emerged and what words occur most often with each sense. Several variations to the data set used in this project, such as stop word filtering, will be made to see if it is possible to get a clearer idea of the what words most often occur with each sense and what impact these changes might have to each sense found. In sections 1.1 and 1.2, the aims and motivation for the project will be discussed respectively. Then in section 1.3, an overview of the project will be given.

1.1 Aims of project

The first goal of this project is to see how an unsupervised machine learning model, the Expectation Maximization (EM) algorithm, can be used to find the meaning change of a target word.

Another goal of this project was to see how manipulating the data would impact the results of the EM algorithm. This will include the development of programs to filter stop words from

the data and also remove data to see what impact it may have on results.

1.2 Motivation

The motivation for this project stems from the fact that many words in natural language have more than one meaning. For example, the word *mouse* could mean either the animal or the device used to move a cursor on a computer screen. When these new senses emerge and in what context they happen in is a fascinating part of natural language and one that can be analyzed using unsupervised methods such as EM. Seeing whether removing stop words and data will have an impact on the results of this EM procedure could also be interesting.

1.3 Overview of Project

This project consists of a discussion about the related works and some background information to give context to the project.

After this, the design and implementation of the methods used in this project will be discussed and explained.

Then the results obtained will be analyzed and the project will conclude with a summary of these results and some potential for future work.

2 Background

In order to help understand the most important parts of the project this chapter will discuss some background information. Some information on related work will be given in section 2.1 to set the basis for this project. Section 2.2 discusses the Diachronic Model which the Expectation Maximization procedure is based on. Section 2.3 talks about the EM algorithm itself. In section 2.4 information on the data used and in section 2.5 stop words will be discussed. Finally, some difficulties faced while completing this project will be described in section 2.6.

2.1 Related Work

Some related work in the area of word sense disambiguation using the EM algorithm was done by Arun Kumar Jayapal (2). This paper gives some excellent detail about the EM algorithm itself and its use in tackling the word sense disambiguation problem with several data sets using several different target words. One section of the paper also deals specifically with the target word mouse which can be used as a comparison for this project.

Other work in this area also includes another paper by Arun Kumar Jayapal and Dr. Martin Emms (3) which discusses a diachronic model to detect new sense emergence. This paper uses time-stamped raw text to see how words acquire new meanings and proposes a model to do this along with using a Gibbs sampling method for parameter estimation.

Similar to this, Arun Kumar Jayapal and Dr. Martin Emms have collaborated on another paper (4) which tries to detect change and emergence for multiword expressions. This paper looks at how multiword expressions and their n-grams change over time and also proposes both a model to detect this change using context word probabilities and expectation maximization technique, similar to this project, for parameter estimation of that model.

2.2 Diachronic Model

A diachronic model is used to model the meaning change of a language. The particular probabilistic model used for this project was originally developed by Dr. Martin Emms at Trinity College Dublin and is described in the paper "Dynamic Generative model for Diachronic Sense Emergence Detection" (3) as follows where Y is the year, S is the sense and w is the word string:

$$P(Y, S, w; \pi_{1:N}, \theta_{1:K}, \tau_{1:N}) = P(Y; \tau_{1:N})P(S|Y; \pi_{1:N}) \prod_i P(w_i|S; \theta_{1:K}) \quad (1)$$

With a more simplified version as follows:

$$P(Y, S, w) = P(Y)P(S|Y) \prod_i (P(w_i|S) \quad (2)$$

An expectation maximization algorithm is then used to work out these parameters.

2.3 Expectation Maximization algorithm

The Expectation Maximization (EM) algorithm is an iterative estimation technique used to find the maximum likelihood estimate (MLE) of the parameters of a model with parameter distribution θ from incomplete data. MLE is used to estimate the model parameters for which the observed data will have the maximum likelihood (2). In the case of this project, the model will be the diachronic model described in section 2.1 and the incomplete data will be the target word which has no labeled sense. The program used for this project, was developed by Dr. Martin Emms at Trinity College Dublin and is described in the paper "An unsupervised EM method to infer time variation in sense probabilities"(5) as follows:

The EM algorithm seeks to find the MLE by iteratively applying the following two steps:

(E-step) - For each training instance d^i in a data set, D = (d^1,d^2,...d^n), consider all its possible completions with setting S, (S = k, x^d), computing for each its conditional probability P(S = k|x^d;θ^n) under the current estimates n. Let γ^d(k) represent this conditional probability.

(M-step) - Treating γ^d(k) as if they were genuine counts, apply Maximum Likelihood Estimation to the virtual corpus of completions to derive new estimates for $\theta^n + 1$.

The EM procedure ensures that the data gets likelier over iterations.

2.4 Data Used - Google 5-grams

The data used for this project was a sub-corpus of the Google 5-gram data set with the target word *mouse*. The Google 5-gram data set was created by digitizing books and noting the frequencies of strings using a yearly count of 5-grams in these digitized books. The mouse data set contains occurrences of the word mouse along with four other words, or tokens, per line. Each line also contains the year that the 5-gram occurred and how many times it occurred that year. (6)

2.5 Stop words

Stop words, sometimes also known as function words, are described as the most frequent words used in language. These words have very little to do with the meaning of a word and thus filtering them out should give a clearer image of what a word means and when new meaning may have emerged. For this project, the stop word list used was the same one used by NLTK software in Python (7). This list includes words like "the" and "a" and about 120 more of the most frequent words used in the English language. There may be more words but this list was used for convenience and consistency.

2.6 Difficulties Faced

Some difficulties faced when completing this project included some long run time problems when filtering stop words and also some errors when removing data for certain experiments which will be discussed later in the project. The long run time problem caused by the filtering of stop words happened because after each line was filtered of stop words, a new file would be opened and this line would be put into the new file. This new file would be opened each time a line was filtered so for files which had over 20,000 lines the file would have to be opened 20,000 times. This was fixed by creating an empty string and adding each newly filtered line to that string. Once all the lines in a file had been filtered the string with all the filtered lines was added to a new file, thus only having to open the file once.

Another problem encountered was when removing lines from files, a white space character would be added to the end of the remaining lines. This caused the EM algorithm and inspector program to think that the year was a part of the n-gram. This resulted in really erratic graphs and the most frequent words seen were filled with years instead of words. Removing this empty space character fixed this problem.

3 Design and Explanation of Programs Used

The following chapter will describe the design of the programs used to filter stop words and remove data from the mouse data set. This chapter will also describe the EM algorithm, inspector program and R program used in this project. Some of the parameter settings for the EM and inspector program will be provided for clarity.

3.1 Filtering stop words program

The filtering program used in this project was developed in the programming language Python. The source code for this program is included in the Appendix section at the end of this report. Simply put, the program takes a list of stop words and filters them from a given file. This stop word list is the same one used by NLTK software in Python (7). Stop words are the most frequent words that occur in natural language and have very little to do with the meaning of a word.

More precisely, the program can do two things:

Firstly, it can filter all the stop words from all the files in a predefined directory. The input location of this directory is hard coded as "input_location" and so too is the location of the result of this filtering process, which is called "output_location" in the code. The program then calls the function "get_stop_words_list(stop_words_path) which takes the predefined location of stop words list and converts the file into a list. This is then stored in the variable "stopwords". The program then moves on to open the path of each file. It then creates an empty string called "filestring" where the filtered contents of the file will go. It then opens each file and prints that the file has been opened and counts the number of lines in the file. This was done as a check to make sure things were working properly. It then loops through each line in the file and calls the function "remove_stop_words(line,stopwords)". This function takes a line and the list of stop words and removes any stop words in the line. Once this is finished this is added to "filestring" along with the new line character "\n" so that the

result will have one sentence per line and not just one big long line. The program then outputs the number of lines and also creates a new file to put the results into called "new_file_path". The program then opens this new path and puts the finished results of "filestring" into it.

The second thing this program does is to take a predefined list of files and only filter the stop words from the files listed in that file. This program opens this list of files and then loops through the names of each of the files, if the file ends in a new line character, remove the new line character. This was done as the final file in the list would often have a new line character at the end which would mess with how the name of the new output file would obtained later in this part of the program. It then creates an empty string known as "filestring" to put the results of the filtering process. The filtering process then begins by opening each file in the list, again printing the name and number of lines as a sanity check, and filtering the stop words using the "remove_stop_words(line, stopwords)" function previously mentioned. The new file path is determined by getting the final four characters of the string like so: new_file_name = file_name[-4:]. This is added on to the output path. Finally, the "filestring" with the results of this process is written to the output file.

This is executed using a command line prompt by typing something like the following, where FilterStopWords.py is the name of the program followed by either a "1" to filter an entire directory or "2" to filter a list of files:

```
$ py FilterStopWords.py 1
```

The idea behind doing this was to try and get rid of words that typically have little to do with the meaning of a word and see what the result would be when the filtered data set was passed through the EM algorithm.

This approach was used as it opens each file to be filtered once, puts the newly filtered line of the file into a string and once all lines have been filtered, opens up a new output file and writes "filestring" to this output file. Previous attempts at doing this opened the output file every time a line was filtered. This way of doing things meant that the output file would be opened for every line in the file causing the program to run for several hours. Doing it as described using the "filestring" to hold all the results at once allows for the program to only have to open output files once and thus making it much faster taking only a few minutes.

3.2 Removing lines program

This program was used to remove data from the mouse data set so that only every 100th n-gram would remain. So for example, if a file had 1000 lines this program would create a

new file with only 10 lines. The program was developed in Python and the source code is available in the Appendix section at the end of the report. This program works similarly to how the filtering program works and was used to see what effect it would have on the EM algorithm in finding the sense of a target word.

The program first goes through a list of file and filters the words from each file in the list. It, firstly, checks the name of each file and makes sure that there are no files with a new line character at the end. It then creates an empty string called "filestring" to put every 100th line into. It then gets every 100th line by doing "for line in f2.read().split('"')[::100]:". This line is then added to "filestring" minus the final character as this was found to be an empty space. Not removing this empty space character resulted in messing with the results of the EM algorithm. The program then gets the file name, where the output will go, by getting the final four characters of the current file and adding this to a predefined output location. The result of "filestring" is then written to this new output location.

The next part of this program filters the result of the previous step of any stop words just like in the section 3.1.

As described above, the program removes an empty space character at the end of each line when getting every 100th line. Leaving this empty space character caused the EM program to think that the year was a part of the N-gram. This resulted in really erratic graphs and also inspector results that had years as the most frequently occurring words. Doing it as described above stops this and deletes any unwanted artifacts.

3.3 Expectation Maximization algorithm

The Expectation Maximization algorithm is used to apply probabilities to each sense found.

The code for the EM algorithm was developed by Dr. Martin Emms at Trinity College Dublin. The full list of parameters can be seen in the documentation (8) but the main parameters used for the experiments in the Results chapter are provided below with a short explanation of their use.

-targ T - specify the target word.
-files FILENAME - specify a list of files which will make up the corpus.
-whether_csv_suffix yes/no – specify if the file extension is .csv or not.
-corpus_type N – specify what type of corpus is being used. In this project the Google ngram corpus was used which is specified as '2' here.
-left N – specify the number of words expected to the left of the target word.
-right N – specify the number of words expected to the right of the target word.

-whether_pad yes/no – specify if there is to be extra padding characters if there are not enough words to left/right of target word.
-group_size N – specify if years are to be grouped together.
-expts FILENAME – specify the output file.
-senseprobs_string N/N – specify the initial probability of each sense where N is less than or equal to 1 and both numbers add up to 1..
-word_rand yes/no – specify whether to generate random word probabilities for each sense.
-set_seed yes/no – specify whether separate runs of program all produce the same results.
-rand_word_mix yes/no – specify whether or not to initialize value supplied to rand_word_mix_level -rand_word_mix_level N – specify the level of randomness to add to non-random word probability initialization.
-words_prior_type unif/corpus – specify whether to set all word probabilities to be the same value or set word probabilities to be the corpus probability of the word.
-unsup yes/no – specify whether to run the unsupervised parameter learning process.
-num_senses N – specify the number of senses to find. If >2 then senseprobs_string should have more than 2 probabilities.
-max_it N – specify the number of iterations.
-time_stamp yes/no – generate a time stamp and add it to the end of the output files.

3.4 Plotting program

The plotting algorithm was developed in the programming language R. R is typically used to analyze data and in the case of this project it was used to plot graphs to see the results of the EM algorithm.

3.5 Inspector Program

The inspector program was also developed by Dr. Martin Emms and takes whatever parameters were used in the EM algorithm along with the following parameters explained below (8).

This program outputs the most frequent words that appear with each sense. Depending on the parameters used this can be based on the probability of a word given a sense, $P(w|S)$, or the probability of a word given a sense divided by the probability of the word appearing in the corpus, $P(w|S)/P_{CORP}(w)$.

-sym SYM TABLE FILE – specify the generated symbol table to be used which has been generated by the EM program.
-wordprobs WORD PROBS FILE – specify the word probability file to be used which has been generated by the EM program..
-senseprobs SENSE PROBS FILE – specify the sense probability file to be used which has been generated by the EM program..
-w_g_ss yes/no – specify whether to show all significant words for all sense.
-w_g_s yes/no – specify whether to show significant words for a sense.
-first N – specify a sense number when comparing two senses.
-second N – specify a sense number when comparing two senses.
-max_show N – specify the number of words to show.
-ranking to_corpus/freq – specify in what way the words should be ranked. By choosing 'to_corpus', the program will rank the words according to their corpus probabilities. By choosing 'freq', the program will rank the words in order of their frequency.

4 Results and Evaluation

In this chapter, the results of the EM algorithm and inspector programs will be discussed. The data for each of the four tests was taken from the mouse data set between the years 1900 and 2008. The EM algorithm is used to get the probabilities of a word given a sense for each year. The EM algorithm was set to 100 iterations for each test. These probabilities are then plotted by a plotting program in R to help visualize the data. Then the results of the inspector program will be displayed in a table. The results of this program where tested using parameters for the outright most frequent words $P(w|S)$, in other words when the parameters of the inspector program where set to "w_g_s yes –first 0 –second 1 ranking freq", and also when these most frequent words where divided by how often they occurred in the corpus $P(w|S)/P_{CORP}(w)$, in other words when the parameters of the inspector program where set to "w_g_ss yes -ranking to_corpus". These programs help to visualize the results and get a clearer idea of what and when a sense occurs in a corpus.

Some terminology for the following chapter is that the senses will be ordered in index order. For example, when talking about two senses it will be discussed as "Sense 0" and "Sense 1". This was done out of convenience as this is the way the inspector program works. As well as that, the colours in the graph correspond to a sense in the inspector results so red = sense 0, blue = sense 1 and black = sense 2.

It should be noted in the following tables that a sense named 'Sense 1' may not necessarily be the same as 'Sense 1' in another table.

4.1 Results of Experiment 1

The first test carried out for this project tried to figure out what senses of the target word mouse the EM algorithm could find given data from the years 1900 to 2008.This data was then filtered of stop words to compare results and see the impact it would have on finding senses. The plot in Figure 4.1 shows the results of the EM algorithm. In the graph on the left it can be seen that up until around 1980 there was only one sense of the word mouse (blue line) but then a new sense of the word mouse appears (red line). This emergence happens at the same time regardless of filtering but, in the case of the filtered data in the graph on the right, the newly emerged sense appears to be slightly more prevalent in the mid 1990s and early 2000s and eventually becomes equal to the old sense of mouse by the end of the data.

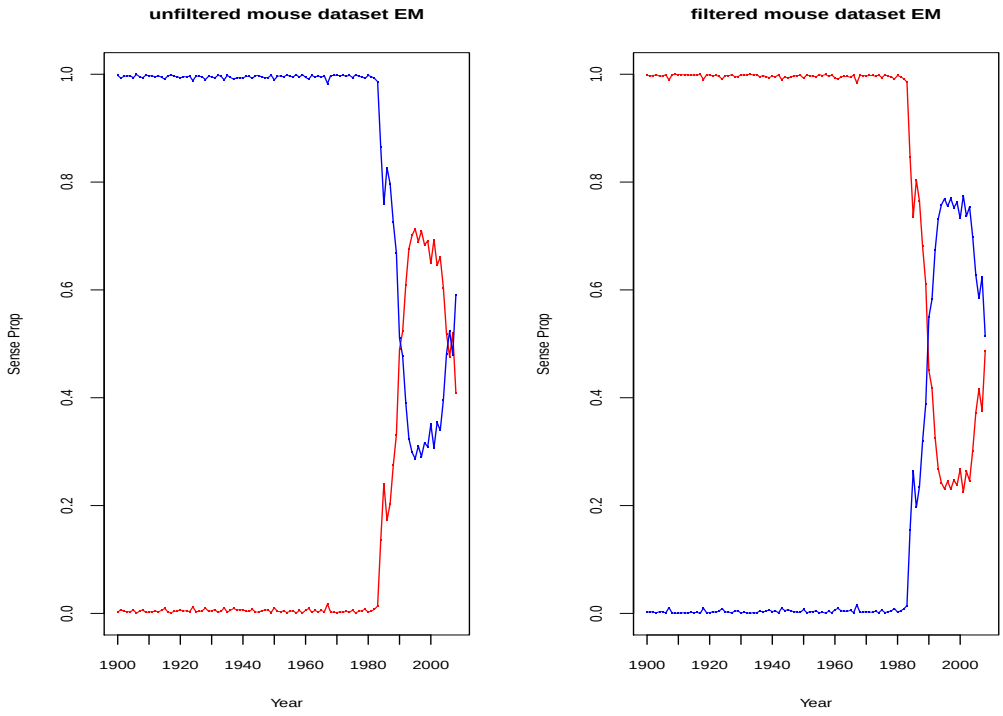

Figure 4.1: Left: Results of EM test of unfiltered mouse data set. Right: Results of filtered mouse data set

In order to gain an understanding of the potential meaning of these two senses an inspector program was used to look at the top 30 words that occurred most often with each sense. The result for the unfiltered test can be seen in Table 4.1. Looking at this table there are many stop words that do not contribute much to the understanding of the meaning of the word. However, there are still several words that do give some context. Sense 0 corresponds to the red line in the graph on the left hand side in Figure 4.1 and has words like 'button', 'pointer' and 'click' as some of its top words. Sense 1, which corresponds to the blue line in

the graph on the left hand side in Figure 4.1, has words like 'cat', 'rat', and 'cells' as its top words. From this it can be determined that Sense 0 has something to do with the mouse used with a computer and that Sense 1 has something to do with the animal mouse. Sense 1 also has several scientific words like 'cell' and 'embryo' which corresponds to the testing that would often happen to mice in laboratories.

Table 4.1: The top 30 words that occurred most often given a sense in the unfiltered data set.

Sense 0	**Sense 1**
the	the
button	a
,	.
pointer	,
.	of
to	_END_
and	and
left	in
click	-
END	as
a	cat
you	with
right	rat
START	or
over	cells
release	(
your	"
down	keyboard
with	to
is	_START_
move	human
of	)
drag	model
or	is
on	anti
use	game
hold	like
when	from
then	for
Release	embryo

The results of the filtered test can be seen in Table 4.2. By looking at this table, one can see that all the stop words have gone from the top 30 most frequent words. What remains are content words which give a better understanding of what the target word might mean. Some of the most frequent words in Sense 1 are 'cat', 'rat', 'cells' and also 'little' and 'field'. This gives the impression that this sense is about the animal mouse. Sense 2 has occurs with

words like 'button', 'pointer', 'click' and 'drag'. It would stand to reason that this sense has to do with the computer mouse.

Table 4.2: The top 30 words that occurred most often given a sense in the filtered data set.

Sense 0	**Sense 1**
.	button
,	,
END	.
-	pointer
cat	left
rat	_END_
cells	click
"	_START_
(	right
human	release
)	move
START	keyboard
anti	drag
game	use
like	hold
embryo	using
mammary	-
house	Release
embryos	clicking
brain	cursor
cell	position
little	Move
rabbit	clicks
field	press
:	model
/	changes
tumor	user
development	Click
virus	moving
's	)

4.2 Results of Experiment 2

The second experiment carried out on the mouse data set was to check for a third sense in the data. As seen in experiment 1, Sense 0 in the unfiltered test and Sense 1 in the filtered test had words corresponding to a more scientific meaning so perhaps looking for a third sense would reveal some interesting results.

Again the same data from the years 1900 to 2008 was used. The plot in Figure 4.2 shows the results of the EM test when the EM parameter "num_senses" is set to 3. The results now show three senses for the target word mouse. Table 4.3 and Table 4.4 show the most frequent words which occur for each sense in the unfiltered and filtered data set, respectively, when the inspector program parameters where set to "w_g_s yes –first 0 –second 1 –ranking freq" for one run and "w_g_s yes –first 1 –second 2 -ranking freq" for a second run. This is the only way to look at three senses for the setting "ranking freq".

The new sense that the EM algorithm has found corresponds to the black line in the unfiltered data set on the left of Figure 4.2 and blue line in the filtered data set on the right of Figure 4.2. The black line corresponds to Sense 2 in Table 4.3 and the blue line corresponds to Sense be 1 in Table 4.4. By looking at the results of the inspector program in these tables, there seems to a lot of words which relate to the sense of mouse found in a lab. Words like "cells", "embryo" and "brain" are quite prominent for these senses. By looking at the graph for the unfiltered data in Figure 4.2, this sense begins to pick up around 1940. Perhaps around this time, lab rats and mice were being used very frequently and therefore appeared in a lot of literature. In the filtered graph in Figure 4.2, however, this "lab" sense of the word is consistent across the last 100 years or so. Filtering stop words from the data set causes this sense to be much more prominent.

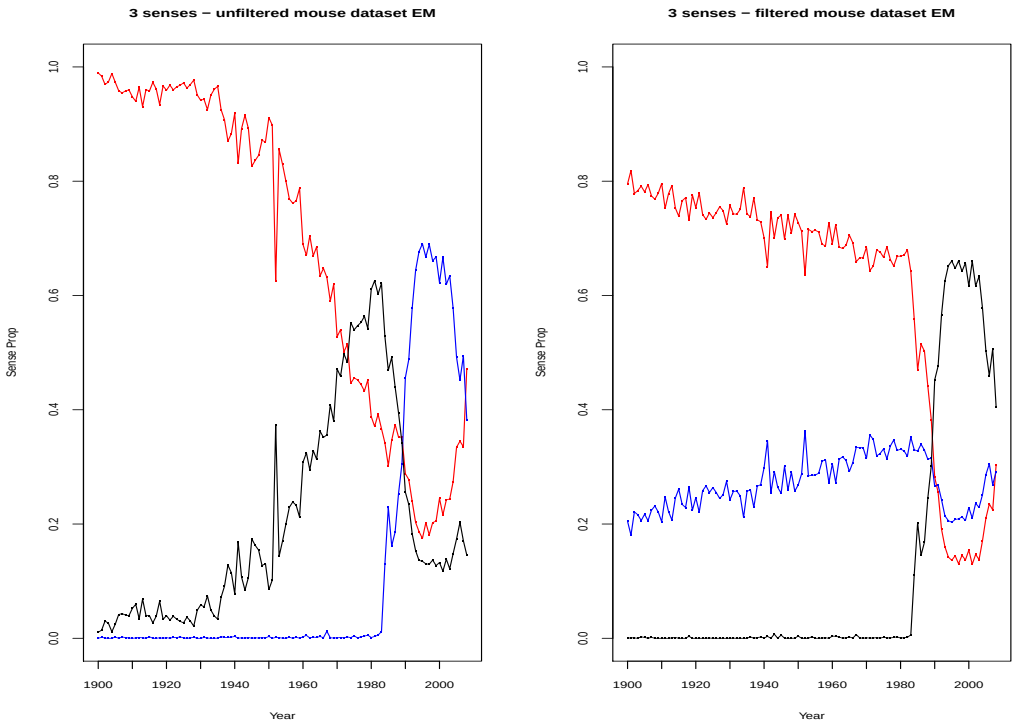

Figure 4.2: Left: Results of EM test on unfiltered mouse data set looking for three senses. Right: Results of filtered on mouse data set looking for three senses

Table 4.3: The top 30 words that occurred most often for three senses in the unfiltered data set using parameters "w_g_s yes" and "ranking freq".

Sense 0	**Sense 1**	**Sense 2**
a	the	the
,	button	.
the	pointer	in
and	.	of
.	,	_END_
END	to	-
of	left	cells
-	and	and
in	_END_	(
as	click	,
cat	right	a
or	you	anti
with	over	embryo
rat	release	)
keyboard	_START_	mammary
"	on	embryos
START	your	human
to	down	cell
is	a	from
game	move	brain
click	with	house
model	is	by
like	of	tumor
that	drag	/
The	use	development
for	or	virus
not	hold	:
was	when	gene
little	then	with
such	Release	bone

Table 4.4: The top 30 words that occurred most often for three senses in the filtered data set using parameters "w_g_s yes" and "ranking freq".

Sense 0	Sense 1	Sense 2
,	.	button
-	_END_	,
.	button	pointer
cat	"	left
rat	model	click
(	,	_START_
cells	left	right
END	embryo	.
anti	brain	release
human	keyboard	move
game	click	drag
START	)	use
)	using	keyboard
"	development	hold
mammary	's	Release
like	embryos	-
cell	cells	cursor
rabbit	;	sing
tumor	clicks	clicking
/	transgenic	Move
house	right	position
virus	?	press
field	-	changes
trap	_START_	user
–	pointer	Click
bone	adult	moves
Mus	gene	select
white	like	moving
IgG	rat	(
little	skin	holding

Another use of the inspector program is to check for all senses and rank their most frequent words based on how frequent they are divided by their corpus probability ,$P(w|S)/P_{CORP}(w)$, as mentioned in the introduction to this chapter. Table 4.5 and Table 4.6 show the most frequent words which occur for each sense in the unfiltered and filtered data set, respectively, when the inspector program parameters where set to "w_g_ss yes –ranking to_corpus". This way of ranking will cause words which appear very often, in other words stop words like "the", to appear less often in the top 30 words. This is because these words are given a high probability in the EM algorithm and are then divided by the number of times they occur in the corpus which will also be very high thus leading to a result which

is close to one. Content words like “button” or “cat” will appear more often in the most frequent words for this way of ranking as they appear much less in the corpus.

Table 4.5: The top 30 words that occurred most often for three senses in the unfiltered data set using parameters "w_g_ss" and "ranking to_corpus".

Sense 0	Sense 1	Sense 2
cat	button	cells
a	pointer	anti
rat	left	embryo
"	right	in
as	release	mammary
keyboard	over	embryos
game	move	(
,	down	brain
like	your	cell
model	drag	tumor
little	you	/
such	to	virus
or	hold	development
and	on	of
for	click	house
In	then	gene
was	when	bone
not	use	Mus
trap	Release	IgG
that	cursor	marrow
than	clicking	from
-	the	human
field	Move	embryonic
but	position	:
's	press	_END
quiet	is	skin
clicks	$_s TART$	)
church	Click	adult
A	changes	.
’	while	-

Table 4.6: The top 30 words that occurred most often for three senses in the filtered data set using parameters "w_g_ss" and "ranking to_corpus".

Sense 0	Sense 1	Sense 2
-	_END	pointer
cat	.	button
rat	model	left
cells	brain	release
anti	"	click
game	development	right
(	embryo	move
mammary	transgenic	drag
human	;	hold
,	?	use
rabbit	's	_START
tumor	embryos	Release
cell	adult	cursor
virus	strains	Move
/	gene	clicking
like	church	keyboard
bone	genome	position
field	skin	press
trap	early	changes
Mus	clicks	Click
IgG	device	user
marrow	developing	select
house	liver	moves
–	preimplantation	using
white	macrophages	holding
guinea	models	,
)	'	moving
footed	man	Position
pig	peritoneal	dragging
goat	little	place

4.3 Results of Experiment 3

The third experiment consisted of removing data so that only every 100 lines from each year were considered. The stop words where then filtered from the data set after this and compared to the unfiltered version. This test was carried out to see how much data is actually needed to see the sense of a target word change. Figure 4.3 below shows the results of the EM test for this experiment. Looking at these graphs it seems that filtering every 100 Ngrams from the data has a larger, negative effect on the data that still has stop words. The EM algorithm struggles to figure out what the sense of the target word is and only around the 1970s does it begin to be more consistent. Comparing this to the experiment 1, Figure 4.1, where the second sense only emerged in the 1980s, using only every 100th Ngram from the full data set negatively affect the results.

On the other hand, the data set with the stop words filtered seems to have a much more similar look to that of its counterpart in Figure 4.1. The results in Figure 4.3 are still somewhat erratic at the beginning but at around 1980, the same as in the first experiment, it is able to clearly define a second sense.

On the whole, filtering too many lines from a data set can have a negative effect on trying to find the sense of a word, particularly when the data set has many words which do not contribute much to the sense of a target word.

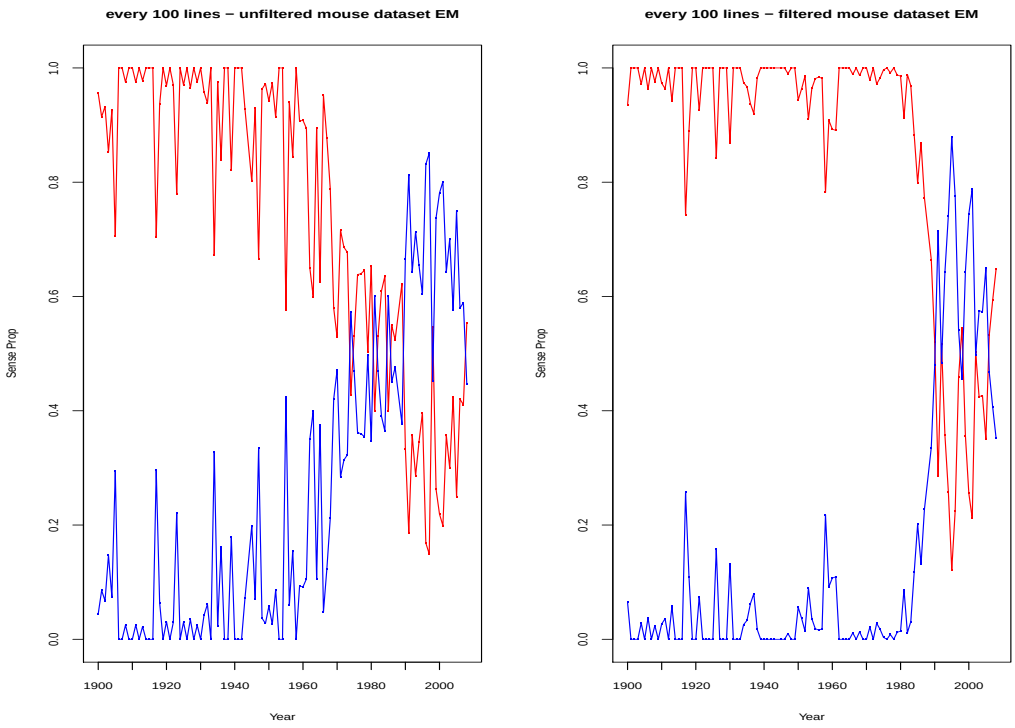

Figure 4.3: Left: Results of EM test on unfiltered mouse data set, where only every 100th Ngram was considered, looking for two senses. Right: Results of filtered on mouse data set, where only every 100th Ngram was considered, looking for two senses

Below are the results of the top 30 most frequent words that occur with each data set for this experiment with the inspector parameters "w_g_s yes" and "ranking freq", provided in Table 4.7 (data set with stop words) and Table 4.8 (data set with stop words filtered). In Table 4.7 the most common words are mostly stop words to begin with but eventually some content words are found which give a sense of the word. For Table 4.8 it is clearer earlier in the most frequent words what the sense might be as a result of the filtering. These tables again show that one sense belongs to the animal sense of the target word mouse and the other sense belongs to the computer mouse.

Table 4.7: The top 30 words that occurred most often for two senses in the unfiltered data set with every 100th Ngram using parameters "w_g_s" and "ranking freq".

Sense 0	**Sense 1**
a	the
,	button
the	.
.	and
and	_END_
END	,
-	to
of	in
START	of
in	right
The	pointer
pointer	with
as	release
with	left
cat	you
points	over
keyboard	a
"	move
click	your
rat	on
or	drag
game	or
is	_START_
to	Release
(	is
human	click
anti	use
like	embryo
that	then
model	by

Table 4.8: The top 30 words that occurred most often for two senses in the filtered data set with every 100th Ngram using parameters "w_g_s" and "ranking freq".

Sense 0	**Sense 1**
.	button
END	,
,	pointer
-	right
cat	_START_
"	release
embryo	.
rat	left
(	move
game	click
cells	Release
human	drag
click	points
keyboard	use
anti	keyboard
clicks	using
house	Click
like	cursor
)	clicking
model	)
START	released
mammary	hold
cell	(
/	press
embryos	-
little	Drag
field	moving
's	one
brain	let
tumor	moves

The words found by the inspector program using the parameters "w_g_ss yes" and "ranking to_corpus" can be seen in Table 4.9 (data set with stop words) and Table 4.10 (data set with stop words filtered).

Table 4.9: The top 30 words that occurred most often for two senses in the unfiltered data set with every 100th Ngram using parameters "w_g_ss" and "ranking to_corpus".

Sense 0	Sense 1
The	button
cat	right
points	release
-	left
rat	move
game	drag
"	Release
as	your
a	over
anti	then
like	Click
/	use
model	cursor
into	to
little	released
such	down
field	hold
rabbit	clicking
small	mammary
man	embryo
be	press
that	few
'	on
keyboard	Drag
2	moving
has	tumor
was	the
not	you
for	while
trap	go

Table 4.10: The top 30 words that occurred most often for two senses in the filtered data set with every 100th Ngram using parameters "w_g_ss" and "ranking to_corpus".

Sense 0	**Sense 1**
END	button
cat	pointer
embryo	right
rat	release
game	left
"	move
cells	Release
human	points
-	Click
anti	clicking
house	cursor
mammary	_START_
/	hold
model	released
field	use
tumor	press
little	drag
.	Drag
like	moving
man	using
'	let
brain	moves
bone	go
marrow	either
Mus	releasing
embryos	Move
?	twice
2	Place
development	select
trap	position

4.4 Results of Experiment 4

The final experiment tested on the data for this project grouped the previous experiments results, experiment 3, into decades. In other words, data was removed so that only every 100 lines of the full mouse data set remained. Stop words were then filtered as a comparison and then the years were grouped together by decade for each data set. Therefore, in Figure 4.4 below, the year 1900 consists of the years 1900 to 1909, the year 1910 consists of the years 1910 to 1919 and so on. This grouping process was done by changing the parameter in the EM program "group_size" from the default "1" to "10".

It is interesting to see here that, similarly to experiment 3, the effect of removing lines from the stop words filtered data set is less disruptive when compared to removing lines from the data set that still has stop words. Grouping the data by decade also seems to improve the results of the filtered data set and a graph relatively similar to that in Figure 4.1 is achieved where a new sense emerges around 1980.

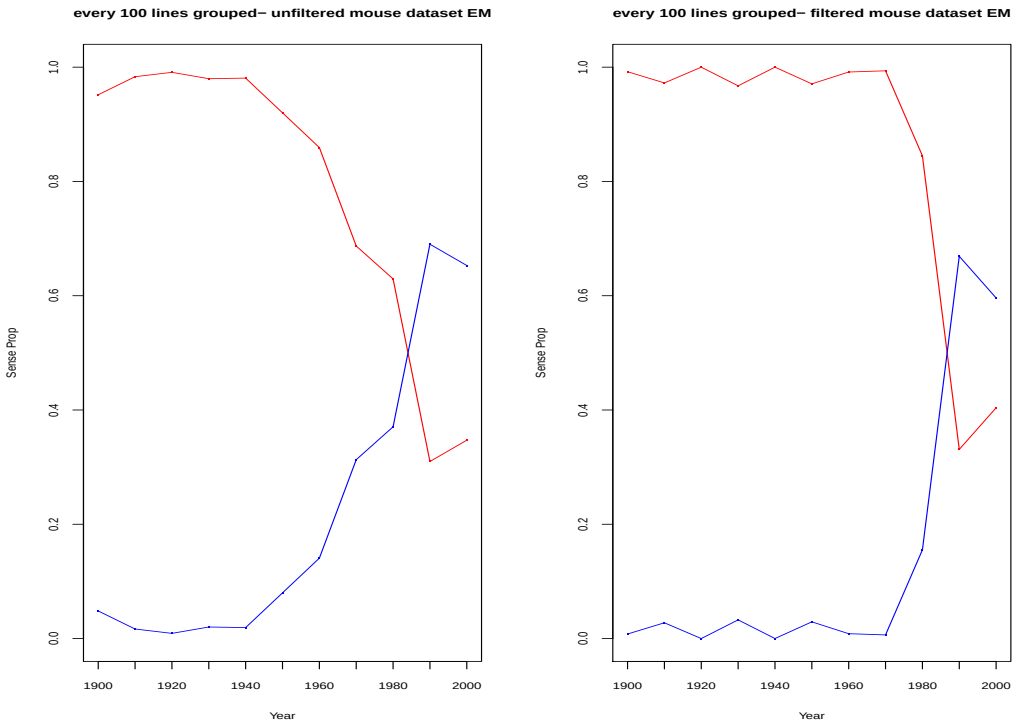

Figure 4.4: Left: Results of EM test on unfiltered mouse data set, where only every 100th Ngram was considered, looking for two senses and years were grouped together by decade. Right: Results of filtered on mouse data set, where only every 100th Ngram was considered, looking for two senses and years were grouped together by decade.

The results of the inspector program when the parameters where set to "w_g_s yes" and "ranking freq" can be seen in Table 4.11 and Table 4.12. Again, the results of this run of the inspector program are quite similar to that of the same run in experiment 3 (Table 4.9 and

Table 4.10) with the top word results being almost identical.

It would stand to reason that grouping the results by decade has little effect on what words might appear with a given sense of a target word.

Table 4.11: The top 30 words that occurred most often for two senses in the unfiltered data set with every 100th Ngram grouped by decade using parameters "w_g_s yes" and "ranking freq".

Sense 0	Sense 1
a	the
,	button
the	.
.	and
and	,
END	to
of	_END_
-	right
in	of
START	in
The	pointer
pointer	with
as	release
with	left
cat	you
points	over
"	your
keyboard	a
rat	move
or	on
(	_START_
click	drag
game	or
human	is
to	click
is	Release
anti	use
like	embryo
house	then
that	down

Table 4.12: The top 30 words that occurred most often for two senses in the filtered data set with every 100th Ngram grouped by decade using parameters "w_g_s yes" and "ranking freq".

Sense 0	**Sense 1**
.	button
END	,
,	pointer
-	right
cat	_START_
"	release
embryo	.
rat	left
(	move
game	click
cells	Release
human	drag
click	points
keyboard	use
anti	keyboard
clicks	using
house	Click
)	clicking
like	cursor
mammary	released
START	hold
cell	)
/	press
embryos	(
little	-
brain	Drag
field	moving
tumor	one
model	let
:	moves

The results of the inspector program when the parameter where set to "w_g_ss yes" and "ranking to_corpus" can be seen in Table 4.13 and Table 4.14. The results are quite similar to that of experiment 3 for both the unfiltered stop words data set and filtered data set with words like "The", "cat" and "points" at the top of the unfiltered data set and "button", "right" and "release" at the top of the filtered data set.

Table 4.13: The top 30 words that occurred most often for two senses in the unfiltered data set with every 100th Ngram grouped by decade using parameters "w_g_ss" and "ranking to_corpus".

Sense 0	Sense 1
The	button
cat	right
points	release
rat	left
-	move
game	your
"	over
as	drag
a	Release
anti	then
like	Click
house	use
human	cursor
model	to
/	down
little	released
such	you
field	hold
rabbit	clicking
small	mammary
man	on
be	press
'	when
2	few
Mus	Drag
for	moving
into	embryo
has	tumor
was	while
trap	the

Table 4.14: The top 30 words that occurred most often for two senses in the filtered data set with every 100th Ngram grouped by decade using parameters "w_g_ss yes" and "ranking to_corpus".

Sense 0	Sense 1
END	button
cat	pointer
embryo	right
rat	release
"	left
game	move
cells	Release
human	points
-	Click
anti	clicking
house	cursor
mammary	_START_
/	hold
brain	use
field	released
tumor	press
like	using
little	drag
.	Drag
rabbit	moving
man	let
bone	moves
2	go
marrow	either
Mus	releasing
embryos	Move
?	twice
development	Place
trap	user
white	select

5 Conclusion

In this chapter, a summary of the project and results will be discussed and also some approaches to future work.

5.1 Summary of key methods, results and findings

In summary, this project tested an EM algorithm on a Google 5-grams data set with the target word mouse. This included tests on the full mouse data set, the mouse data set with stop words filtered out, and also the mouse data set with only every 100th n-gram to find two senses and, in one case, three senses of the word mouse.

The results of this project show that filtering stop words can have an effect on the prevalence of a meaning as seen in Figure 4.1 but does not have an effect on when this new sense emerges. Filtering stop words from the data set also showed that a third sense appeared more consistently than in the unfiltered data set (Figure 4.2). When removing large amounts of data, i.e. leaving every 100th n-gram for each year, filtering the stop words helped the EM algorithm identify the emergence of a new sense more accurately than in the unfiltered case (Figure 4.3) and grouping these results by decades proved this even further (Figure 4.4).

5.2 Future Work

For future work on this project there are a few things that could be done differently and some things that could be added.

For the filtering of stop words part of the project things like punctuation and artifacts of the data like "_END_" and "_START_" could be added to the stop words list so that they do not appear in the top 30 most frequent words list. This could help to see what the actual most frequent words are rather that just what appeared in the data set.

Another thing that could have changed with the filtering programs was to ask in the command line for the path of the list of files to be filtered instead of hard coding it into the program. This could allow for different lists of files to be used more easily.

The project also only looks at one target word whereas different target words could also have been tested. There are lots of words which have experienced meaning change and seeing the effect of stop word filtering on other words could be very interesting.

Other parameter estimation algorithms such as Gibbs sampling could also be used in future work to see the differences they might have compared to EM.

Bibliography

[1] J. R. FIRTH. A synopsis of linguistic theory, 1930-1955. *Studies in Linguistic Analysis*, 1957. URL https://ci.nii.ac.jp/naid/10020680394/en/.

[2] Arun Jayapal. Finding diachronic sense changes by unsupervised methods. diploma thesis, School of Computer Science Statistics, Trinity College, University of Dublin.

[3] Martin Emms and Arun kumar Jayapal. Dynamic generative model for diachronic sense emergence detection. In *Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers*, pages 1362–1373, Osaka, Japan, December 2016. The COLING 2016 Organizing Committee. URL http://aclweb.org/anthology/C16-1129.

[4] Martin Emms and Arun Jayapal. Detecting change and emergence for multiword expressions. In *Proceedings of the 10th Workshop on Multiword Expressions (MWE)*, pages 89–93, Gothenburg, Sweden, April 2014. Association for Computational Linguistics. doi: 10.3115/v1/W14-0815. URL https://www.aclweb.org/anthology/W14-0815.

[5] Martin Emms and Arun Jayapal. An unsupervised em method to infer time variation in sense probabilities. In Dipti Misra Sharma, Rajeev Sangal, and Elizabeth Sherly, editors, *ICON 2015 : 12th International Conference on Natural Language Processing*, pages 266–271, 2015.

[6] Jean-Baptiste Michel, Yuan Kui Shen, Aviva Presser Aiden, Adrian Veres, Matthew K. Gray, , Joseph P. Pickett, Dale Hoiberg, Dan Clancy, Peter Norvig, Jon Orwant, Steven Pinker, Martin A. Nowak, and Erez Lieberman Aiden. Quantitative analysis of culture using millions of digitized books. *Science*, 331(6014):176–182, 2011. ISSN 0036-8075. doi: 10.1126/science.1199644. URL http://science.sciencemag.org/content/331/6014/176.

[7] sebleier. Nltk's list of english stop words, November 2008. URL https://gist.github.com/sebleier/554280.

[8] Dr. Martin Emms. Documentation for em and inspector programs, April 2019. URL https://www.scss.tcd.ie/Martin.Emms/SenseDynamics/SenseDynamicsCode/.

A1 Appendix

A1.1 Python source code for filtering stop words from files

```
import os
import sys

def remove_stop_words(string, stopwords_list):
    string_to_list = string.split()
    x = (' '.join(i for i in string_to_list if i.lower() not in (x.lower() for x in
        stopwords_list)))
    return x

def get_stop_words_list(stopwords_path):
    with open(stopwords_path, 'r') as f:
        stopwords = f.read().split()
    return stopwords

def main():
    input_location = 'C:/Users/User/Desktop/mouse'
    stop_words_path = 'C:/Users/User/Desktop/NLTK-stop-word-list.txt'
    output_location2 = 'C:/Users/User/Desktop/test2/'
    output_location = 'C:/Users/User/Desktop/filter_mouse/'
    stopwords = get_stop_words_list(stop_words_path)

    input = sys.argv[1]
    if input == '1':
        print('Input = ' + input + ', opening all files in directory and filtering
            stopwords.')
        for root, dirs, files in os.walk(input_location):
```

```
        for name in files :
            file_path = os.path.join(root,name) # joins the new path of the file
                to the current file in order to access the file

             filestring = '' # file string which will take all the lines in the
                file and add them to itself
            with open(file_path, 'r') as f: # open the file
                print('just opened ' + name)
                print('\n')
                i = 0
                for line in f: # read file line by line
                    i += 1
                    x = remove_stop_words(line, stopwords) # remove stop words
                        from line
                     filestring += x # add newly filtered line to the file string
                     filestring += '\n' # Create new line
                print(' lines = ')
                print(i)
                print('\n')
            new_file_path = os.path.join(output_location,
                                         name) # creates a new file of the file
                                             that is currenlty being filtered of
                                             stopwords
            with open(new_file_path, 'a') as output_file: # opens output file
                output_file.write( filestring )
elif input == '2':
    print('Input = ' + input + ', filtering stopwords from files given in list
        file .')
     list_file = 'C:\\Users\\User\\Desktop\\list_of_files2.txt'
    with open( list_file , 'r') as f:
        for file_name in f:
            if file_name.endswith('\n'):
                file_name = file_name[:-1]
             filestring = '' # file string which will take all the lines in the
                file and add them to itself
            with open(file_name, 'r') as f2: # open the file
                print('just opened ' + file_name)
                print('\n')
                i = 0
```

```
                for line in f2:  # read file line by line
                    i += 1
                    x = remove_stop_words(line, stopwords) # remove stop words
                        from line
                     filestring  += x # add newly filtered line to the file string
                     filestring  += '\n' # Create new line
                print(' lines = ')
                print(i)
                print('\n')
            new_file_name = file_name[-4:]
            new_file_path = output_location2 + new_file_name
            with open(new_file_path, 'w') as output_file:  # opens output file
                #print(output_file)
                output_file.write( filestring )

if __name__ == "__main__":
    main()
```

A1.2 Python source code for taking every 100th line from file and filtering results

```
import os

def remove_stop_words(string, stopwords_list):
    string_to_list = string.split()
    x = (' '.join(i for i in string_to_list if i.lower() not in (x.lower() for x in
        stopwords_list)))
    return x

def get_stop_words_list(stopwords_path):
    with open(stopwords_path, 'r') as f:
        stopwords = f.read().split()
    return stopwords

def main():

    output_location = 'C:/Users/User/Desktop/100_lines/'
     list_file = 'C:\\Users\\User\\Desktop\\list_of_files2.txt'
```

```
stop_words_path = 'C:/Users/User/Desktop/NLTK-stop-word-list.txt'
with open( list_file , 'r') as f:
    for file_name in f:
        if file_name.endswith('\n'):
            file_name = file_name[:-1]

         filestring = '' # file string which will take all the lines in the file
            and add them to itself
        with open(file_name, 'r') as f2: # open the file
            print('just opened ' + file_name)
            print('\n')
            i = 0
            for line in f2.read(). split ("\n") [::100]: # read file line by line
                i += 1
                 filestring += line # add newly filtered line to the file string
                 filestring = filestring [:-1]
                 filestring += '\n' # Create new line
            print(' lines = ')
            print(i)
            print('\n')
        new_file_name = file_name[-4:]
        new_file_path = output_location + new_file_name
        with open(new_file_path, 'w') as output_file: # opens output file
            output_file.write( filestring )

input_location = 'C:/Users/User/Desktop/100_lines/'
output_location = 'C:/Users/User/Desktop/100_lines_filtered/'
stopwords = get_stop_words_list(stop_words_path)
for root, dirs, files in os.walk(input_location):
    for name in files :
        file_path = os.path.join(root,
                                 name) # joins the new path of the file to the
                                     current file in order to access the file

         filestring = '' # file string which will take all the lines in the file
            and add them to itself
        with open(file_path, 'r') as f: # open the file
            print('just opened ' + name)
            print('\n')
```

```
            i = 0
            for line in f:  # read file line by line
                i += 1
                x = remove_stop_words(line, stopwords) # remove stop words from
                    line
                 filestring += x # add newly filtered line to the file string
                 filestring += '\n' # Create new line
            print(' lines = ')
            print(i)
            print('\n')
        new_file_path = os.path.join(output_location,name) # creates a new file
            of the file that is currenlty being filtered of stopwords
        with open(new_file_path, 'a') as output_file: # opens output file
            output_file.write( filestring )

if __name__ == "__main__":
    main()
```