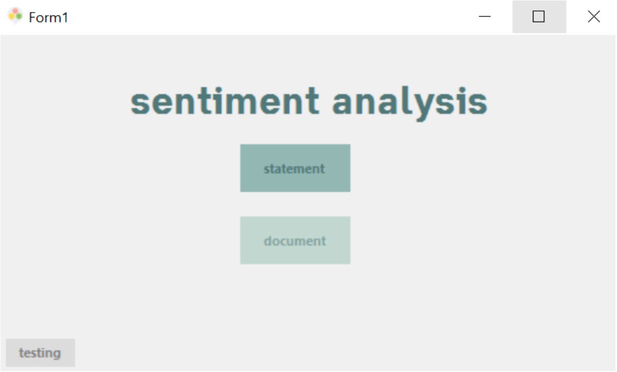
Software Design and Development Project

Anna Dong

Contents

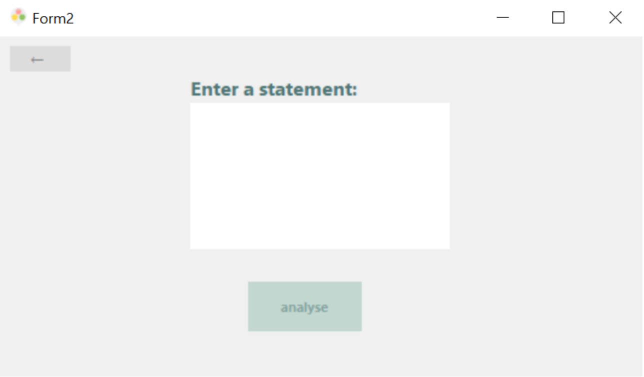
Instruction Manual

Purpose: Sentiment analysis, which is to take in a text input and classify the input as having a positive or negative tone.
Note: the program can be closed anytime by clicking the “x” button on the top of any window, and to return to the previous page, click the back button on the top right.
For any unfamiliar terms, check the glossary at the end of the manual.

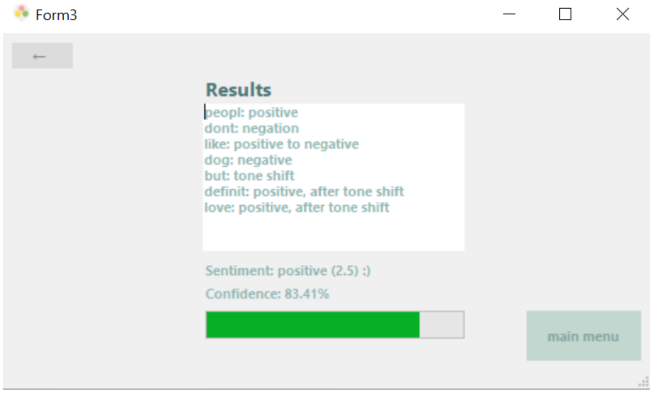


Menu:
The three buttons on the main menu all correspond to the three modes this program can run by. For each button:

- “statement”: see Statement mode
- “document”: see Document mode
- “testing”: see Testing page



Statement mode:
To enter in a statement, select the text box in the middle of the screen, and type it in. There is no word limit, but it’s advisable to input lengthy documents into document mode for faster processing. When the statement is ready to be analysed, click the “analyse” button.

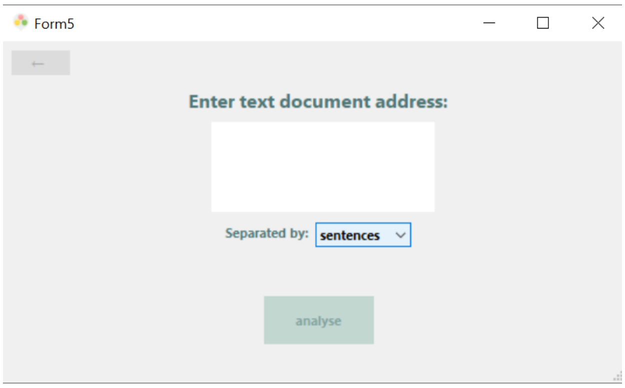


```
peopl: positive
dont: negation
like: positive to negative
dog: negative
but: tone shift
definit: positive, after tone shift
love: positive, after tone shift

Sentiment: positive (2.5) :)
Confidence: 83.41%
```

The resulting sentiment for each word is shown in the text box in the middle. The word may look different to the original, as it has been processed and reverted to its root form. Special words can affect the way sentiment is calculated:

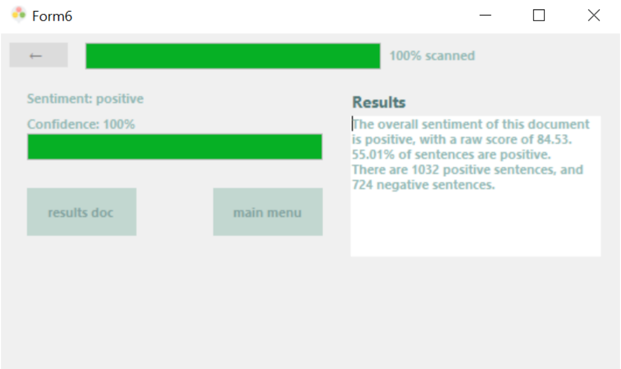
- Negations
- Intensifiers
- Tone shifts

The overall sentiment of the statement, along with its raw score and confidence of the program is displayed on the bottom of the screen. To return to the main menu, click the button on the bottom right corner.



Separated by: sentences

Document mode:
To enter in a document (document must be .txt), select the text box in the middle of the screen, and paste its full address in. The system will display an error message if the document address is invalid. The program will either split the document and analyse it line by line, or sentence by sentence, which can be changed via the drop down menu. When the document is ready to be analysed, click the "analyse" button.



100% scanned

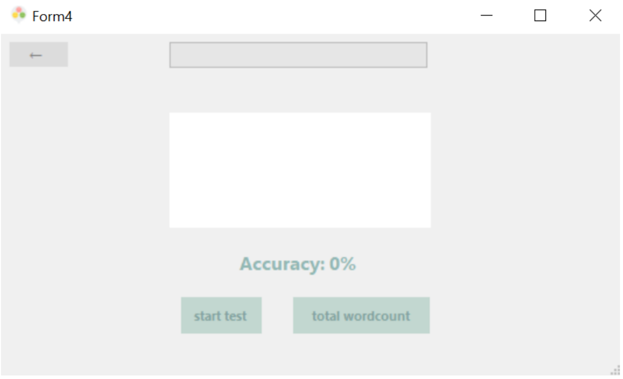
Sentiment: positive

Confidence: 100%

Results

The overall sentiment of this document is positive, with a raw score of 84.53. 55.01% of sentences are positive. There are 1032 positive sentences, and 724 negative sentences.

As the document is being processed, the progress bar will update. Once the program finishes, results will be shown (overall sentiment, raw score, number of positive/negative sentences, program confidence). To access a word-by-word analysis (like the results section in statement mode), click the "results doc" button, which opens a text document created in C://Temp. To return to the main menu, click the "main menu" button.



Accuracy: 0%

Testing page (mostly for developers):
start test: scans through a pre-selected document of tweets, and compares its calculated results with the actual results. Outputs the number of: correctly identified pos/neg tweets, and the actually positive but identified negative/actually negative but identified positive tweets. This is mainly to display the accuracy percentage of the program for this particular dataset.

total wordcount: scans through tweet database used in calculations, and counts the number of words in the entire database. Used to keep track of size of database for certain variables within the sentiment calculation.

Glossary:
Sentiment: overall tone - either positive, negative or neutral
Raw score: the calculated sentiment represented as a number. The further away from 0, the stronger the sentiment
Negation: words that flip the meaning of the word after it (eg. not good). Words after a negation have their sentiments changed from positive to negative, and vice versa

Intensifiers: words that intensify the sentiment of the word after it (eg. really good)
Tone shifts: any words after “but” will have their sentiment intensified, until the next stopping punctuation (.,?!)

Installation:
(Program currently only designed to work on computers with Windows OS)

1. Find folder (“sentiment analysis application”) and unzip if necessary
2. Inside the folder will be an application and a .dll file. Do not remove or separate these files from the folder.
3. Run the application and wait for Windows Defender to scan the application (up to 10 seconds).

Logbook

2/2:

Experimented with stemming and lemmatisation, imported a public package for lemmatisation to test (LemmaGen/LemmaSharp), using NuGet. Started a simple input processing program that:

- turns an input sentence into lowercase
- puts it into an array
- strips the punctuation
- turns each word into its root form via lemminisation
- prints each word out

Planning to remove noise next: stop words, hyperlinks, twitter handles, etc. Not sure whether or not I should get rid of emoticons like ":)", ":D" and ":P" - they add to the sentiment of the statement but how to ignore those while getting rid of other punctuation??

15/2

Worked on removing noise:

- Anything with "@" or "#" in it was removed (getting rid of handles and hashtags)
- Stopwords removed
- Emojis removed

Not bothered to keep emoticons. Next time, aiming to start reading the database files and finding a way to process and store the inputs so that I can start writing the actual sentiment analysis. When inputting in tweets with "" in them, will create an error, unless there is a "/" before each quotation mark, which also means my input processing program will ignore the quotation marks. Maybe before putting into the program, replace all " to '

21/2

Planned what to do next, what calculations I needed to make from the database of tweets, and how to calculate probabilities of pos/neg based on the database:

- For each word in the input sentence, find the total number of times the word occurred in the negative tweets, then divide it by the total words in negative reviews (to get probability of x given y)
- Multiply the probabilities of all the words in the sentence, then multiply by the probability of a statement being negative
- Repeat above for the positive sentiment, then compare the negative and positive probabilities
- The biggest probability is the sentiment of the statement

This means that I need to make the following from the tweet database:

- The total number of times every word occurred in negative/positive reviews (write file with word counts for each stemmed word)
- Total number of words in negative/positive reviews (2 variables with word count)
- The probability of any document expressing a negative/positive sentiment (50%, since equal number of positive and negative reviews)

- Use above to calculate probability of every word (considering negative or positive) and put onto a file (one for positive, one for negative)

23/2
Downloaded tweet database, has 1.6 million unprocessed tweets, with sentiment ratings for each tweet (0 = neg, 4 = pos). No “” in tweets, so no need to remove.

Created a subprogram that opens up the database, reads each tweet, then creates a dictionary with all the unique words in the database and how many instances each unique word appeared within the tweets. However, since the database is so big, even without stemming the data every line, it takes around 5 mins to create the dictionary. With stemming, the rate of every line run through the program is around 1 line/sec, which would take around 18 days to run. Will need to either cut down the database size, or find a way to make the code run more efficiently.

1/3
Still trying to solve the processing time problem. Found out it's not the actual file reading that's slowing the program down, but the lemmatisation process. WIthout the lemmatisation subprogram running, it should take 1.1 hours to finish the csv file (compared to 18 days). Considering replacing lemmatisation with stemming, will sacrifice accuracy for efficiency.

Checked data again, forgot about links in tweets to images, videos, websites, etc. Filtered out during processing by removing all words that contain “http://”.

Implemented a stemming program (Porter2Stemmer) into the code to replace lemmatisation, making processing much faster - takes 22 seconds to sort and process 0.5% of the database. Next session, will finish program so that it writes the number of unique words onto a new txt file for later reference (only need to run this part of the program once).

2/3
Subprogram can now write word count onto a new file. Did a short test of the subprogram, found that some links, numbers, and also some punctuation (“?” and “+”) was missed during the filtering process. Added them to the filter, and changed the link filter from “http://” to just “http” to accommodate for a wider range of links.

8/3
Ran program for about 20 mins, came back to it saying it was 500% through the program. Found out it was reading the line too fast, moving on to the next line before finishing the one it was on. Fixed by moving a statement out of a loop it shouldn’t have been in, now runs much faster (takes around 10 mins to run through the program), and much more efficiently (no more unnecessary repeating of lines). Fixed some more bugs, trying to successfully run program again. Found some links that STILL aren’t being filtered out, and also found that when words are connected by punctuation (eg. “apples…oranges”) without a space, the program will treat the two words as one, and will output “applesporanges”). Will need to start splitting words by punctuation, not just by spaces.

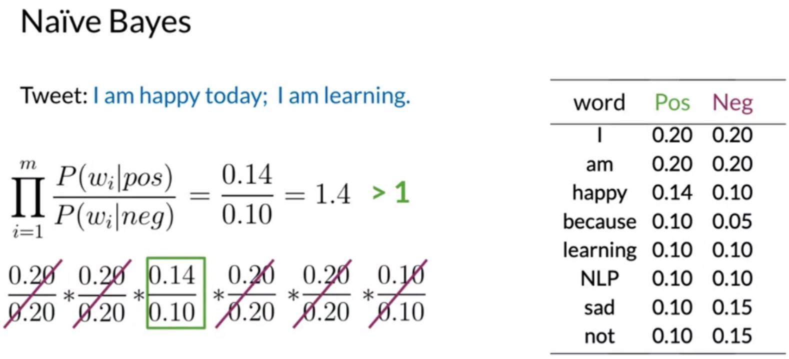
10/3

Changed link filter again to both "http" and ".com". In word processing, split by spaces and punctuation, rather than just spaces. In word list, found lots of words with repeating characters eg. "awwww". Created a subprogram that removes over 2 repeating consecutive letters ("awwwww" should be changed to "aww"). Removed emoji check, since that only removed emojis separated by 2 spaces, not emojis connected to a word without any spaces. Replaced with check at the start of inputProcess, that removes any character that isn't a letter or punctuation. Fixed a handful of bugs. Ran the code to process the database.

14/3

Created subprogram to add the word counts from the word count file (positive and negative separately), giving me a total unique word count of negative: 5193410 and positive: 4826117. Made two subprograms: one to calculate the individual probability of a word being negative/positive, and one to calculate the total negative and positive probability of a whole sentence. Tested with two extreme examples, program works! However, by multiplying the probabilities of words in a sentence to form the final probability, the final probability is really small, and definitely doesn't add to 100%.

May try to switch over to the Naive-Bayes Classifier, which is used with sentiment analysis projects similar to mine.



Naïve Bayes

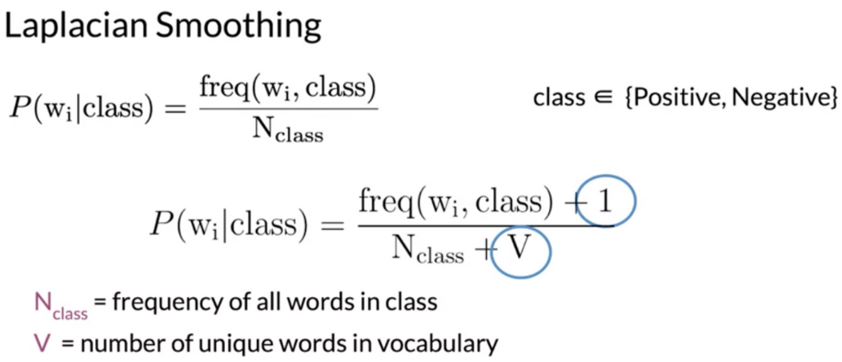
Tweet: I am happy today; I am learning.

$$\prod_{i=1}^{m} \frac{P(w_i|pos)}{P(w_i|neg)} = \frac{0.14}{0.10} = 1.4 \quad > 1$$

$$\cancel{\frac{0.20}{0.20}} * \cancel{\frac{0.20}{0.20}} * \boxed{\frac{0.14}{0.10}} * \cancel{\frac{0.20}{0.20}} * \cancel{\frac{0.20}{0.20}} * \cancel{\frac{0.10}{0.10}}$$

word	Pos	Neg
I	0.20	0.20
am	0.20	0.20
happy	0.14	0.10
because	0.10	0.05
learning	0.10	0.10
NLP	0.10	0.10
sad	0.10	0.15
not	0.10	0.15

May also try out Lapacian smoothing, which solves the problem of having a probability of 0 for one of my words (no occurrences in tweet database). Without smoothing, multiplying the probabilities together, with one probability as 0 will result in the total probability to be 0 (which is bad).



Laplacian Smoothing

$$P(\mathrm{w_i}|\mathrm{class}) = \frac{\mathrm{freq(w_i, class)}}{\mathrm{N_{class}}}$$

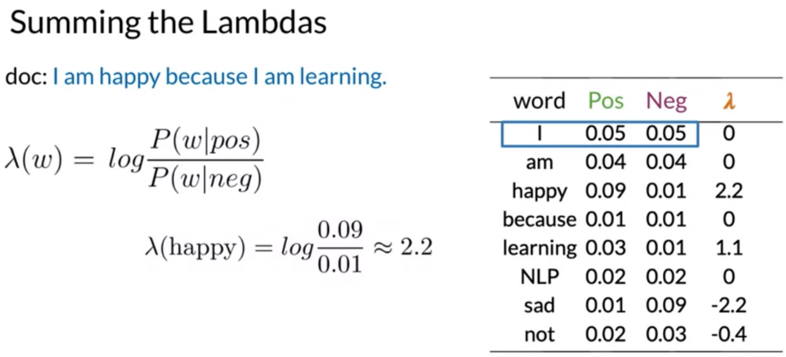
class ∈ {Positive, Negative}

$$P(\mathrm{w_i}|\mathrm{class}) = \frac{\mathrm{freq(w_i, class)} + 1}{\mathrm{N_{class}} + \mathrm{V}}$$

$\mathrm{N_{class}}$ = frequency of all words in class

V = number of unique words in vocabulary

OR if I used log likelihoods, this would negate the need for Laplacian smoothing, and make working with small number much easier:



Summing the Lambdas

doc: I am happy because I am learning.

$$\lambda(w) = log\frac{P(w|pos)}{P(w|neg)}$$

$$\lambda(\text{happy}) = log\frac{0.09}{0.01} \approx 2.2$$

word	Pos	Neg	λ
I	0.05	0.05	0
am	0.04	0.04	0
happy	0.09	0.01	2.2
because	0.01	0.01	0
learning	0.03	0.01	1.1
NLP	0.02	0.02	0
sad	0.01	0.09	-2.2
not	0.02	0.03	-0.4

To calculate the final probability of a sentence, you would just need to sum all the log likelihoods (instead of multiplying).

15/3

Ended up only using log probabilities. Ran the program on some test sentences, found that my ratios were working (correctly classifying the sentiment of each word), but my overall sentence sentiment was much too pessimistic. Out of the 3 really positive sentences I tested, only one was categorised into positive sentiment.

Also, if the word doesn't show up in the word list, either the pos or neg probability will equal 0 or infinity, messing up the final probability.

Fixed the overall sentence sentiment! Had to modify the original Naive Bayes method, ended up doing:

P(sentence) = $\frac{\Sigma log(P(word|positive))}{\Sigma log(P(word|negative))}$

With this method, the sentence length won't affect the probability like an average would. This way, positive sentences are < 0, and negative sentences are > 0.
Still need to fix the 0 and infinity problem - should I just ignore the words that aren't in my list?

To solve this, implemented an if statement - if the word is not on my wordlist, it would return a 50% chance that the word is either positive or negative. This way, for each unknown word in a sentence, the overall rating will move more towards neutral.

Cleaned up the program, added some formatting, and made a loop so that users can type in multiple sentences.

17/3

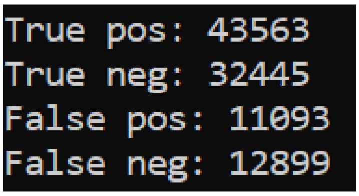
Thinking of inputting a new database of tweets (to avoid any previous bias) to see how well my program categorises them. Also, need to further dampen the impact of sentence-neutral words (such as ‘dog’), thinking about finding the most sentiment heavy word (the word with the biggest ratio between positive and negative probability) and giving that word a bigger weighting in the overall sentiment of the sentence?

Found new dataset, will test on program, then add to word list. Also realised that filtering by “.com” isn't very useful when I’ve already removed all instances of “.”.

19/3
Scanned through the new document, made a new file of word counts to add to final probabilities. Should make the program more accurate and increase the probability of the sentiment being correct. This tweet database is much smaller (100,000 tweets), pre categorised, randomly organised. Will test this database on the program first, then implement the database onto the sentiment probabilities.

Just realised that when I split each line from the database by ‘,’ the program would also split the tweet by any commas inside the tweet as well. May need to redo some of my program. Fixed the problem by splitting the string by two quotation marks and a comma “,” ran the program again. New word count: neg: 6158232, pos: 5840863. Compared to old word count: neg: 5171568, pos: 4780631, words scanned have been greatly increased.

Ran new tweet database into program, categorised each one, then checked against actual sentiment. The program had a 76% accuracy rate, which I’m very happy with. Will add new tweet database into program to help calculate sentiment next.



```
True pos: 43563
True neg: 32445
False pos: 11093
False neg: 12899
```

24/3
Re-processed the new database, and sorted its unique words into a new wordcount file, planning to combine this into my old word count file to make the probabilities more accurate (with a larger and different dataset).

Combined with my old dataset, some probabilities are now improved (eg. like is positive). However, not sure how to deal with the probabilities from the new dataset:

- I could add the word counts from the new dataset onto the old, the divide by the total word count to get overall probability
- I could calculate word probabilities separately for the two databases, then get the final by averaging them both
- Or, I could calculate them separately, then compare the two probabilities to see which one was the most extreme (further away from 0) and only use that one (to get a stronger reading?? This way may overinflate the probabilities of otherwise neutral words)

Ended up going with option 1. Tested the program again on some random samples:

- "Not" is throwing the program off. Lots of ways to fix this - could reverse the whole sentiment of the sentence depending on how many nots there are? Or just reverse the whole sentiment of the word after "not"? Are there more words like "not" (that negate a sentence) that I should account for?
- List of negations:
    - doesn't, isn't, wasn't, shouldn't, wouldn't, couldn't, won't, can't, don't, no, not, nothing, neither, nowhere, never
    - After processing, those should shorten to: doesnt, isnt, wasnt, shouldnt, wouldnt, couldnt, wont, cant, dont, no, not, noth, neither, nowher, never

Implemented negations - a negation should now reverse the sentiment of the word after it. Next, I want to figure out words that increase the sentiment of the word after it: "really", "extremely", etc. (intensifiers)

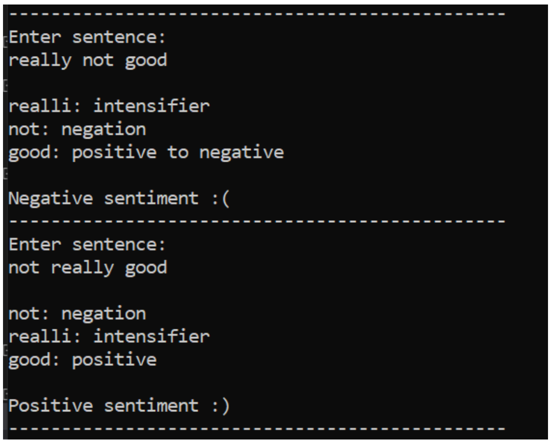
List of intensifiers:

- Extremely, really, super, very

After processing:

- Extrem, realli, super, veri

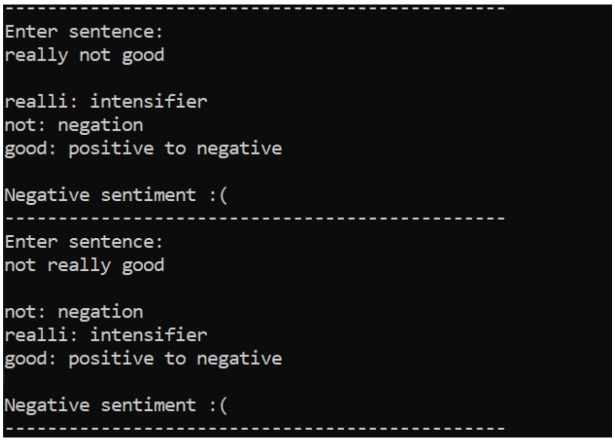
Intensifiers now double the sentiment of the word after it. These negations and intensifiers only take into account the word before it, so when I stack both of them: "really not good" or "not really good" they return different values.



```
-----------------------------------------------
Enter sentence:
really not good

realli: intensifier
not: negation
good: positive to negative

Negative sentiment :(
-----------------------------------------------
Enter sentence:
not really good

not: negation
realli: intensifier
good: positive

Positive sentiment :)
-----------------------------------------------
```

In both of these examples, both the negation and the intensifier should act on the third word, instead of each other. Could do a lazy fix by taking into account the last two words instead of one, but there's probably a smarter way to solve.

Fixed! Intensifiers and negations can now stack, by making two variables for negation status and intensifier status. Every consecutive negation, the negation status switches (making it work for double negations "not not") and every consecutive intensifier, the intensity status increases by 1 (so that for 3 "really" in a row, the next non-negation/non-intensifying word will have a 3x increased sentiment). Will also work for a mix of the two.



```
-----------------------------------------------
Enter sentence:
really not good

realli: intensifier
not: negation
good: positive to negative

Negative sentiment :(
-----------------------------------------------
Enter sentence:
not really good

not: negation
realli: intensifier
good: positive to negative

Negative sentiment :(
-----------------------------------------------
```

25/4

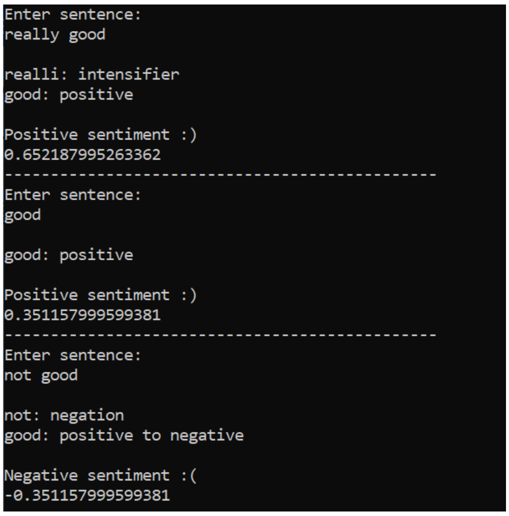
Came back to program, decided to change the way it calculated probabilities to make it easier for me to implement a confidence meter (eg. 90% confident statement is positive). Changed from:

P(sentence) = $\frac{\Sigma log(P(word|positive))}{\Sigma log(P(word|negative))}$

To:

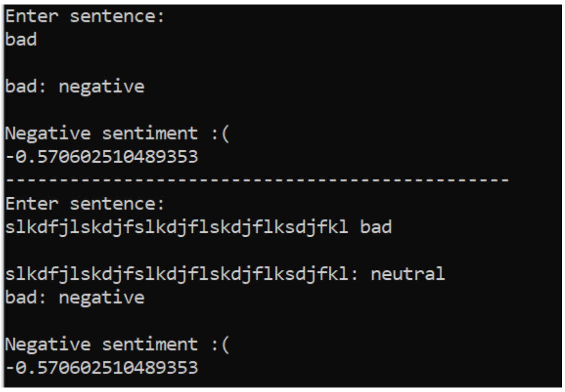
P(sentence) = $\Sigma \frac{log(P(word|positive))}{log(P(word|negative))}$

This way, all my probabilities are centered around 0 instead of 1: positive sentences have a probability > 0, and vice versa. Seems to work ok:



```
Enter sentence:
really good

realli: intensifier
good: positive

Positive sentiment :)
0.652187995263362
---------------------------------------------
Enter sentence:
good

good: positive

Positive sentiment :)
0.351157999599381
---------------------------------------------
Enter sentence:
not good

not: negation
good: positive to negative

Negative sentiment :(
-0.351157999599381
```

However, neutral words do not affect the probability of the sentence anymore:



```
Enter sentence:
bad

bad: negative

Negative sentiment :(
-0.570602510489353
-----------------------------------------------
Enter sentence:
slkdfjlskdjfslkdjflskdjflksdjfkl bad

slkdfjlskdjfslkdjflskdjflksdjfkl: neutral
bad: negative

Negative sentiment :(
-0.570602510489353
```

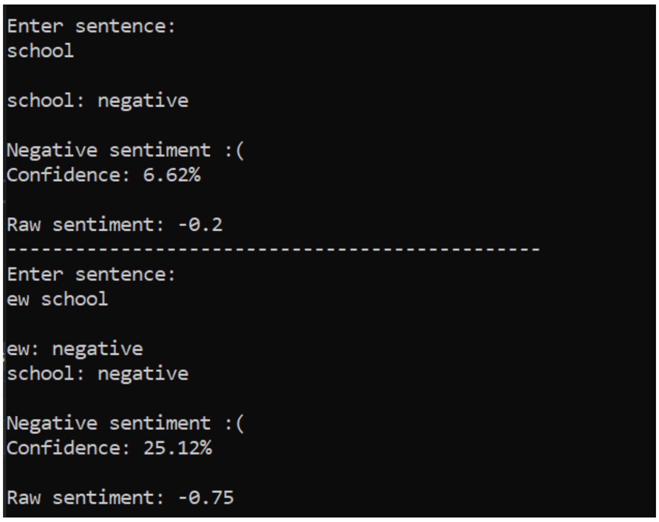
To calculate the confidence of the program, and reflect that in a percentage:

- Confidence could be the distance the sentiment is from 0 (probably the easier option)
- Or it could be the difference between the probability that the sentiment is positive, and the probability that the sentiment is negative (the larger the gap, the higher the confidence?).

After testing extremely positive and extremely negative sentences, I'm pretty sure that the sentiment does not go above or below 3 and -3. This way, I can take those as my range (the closer the sentiment is to the boundary, the more confident the program is/more likely the program is correct?).

Eg. if a negative sentence has a sentiment of -2.1, the confidence percentage is (-2.1)/(-3) = 70%

Implemented confidence meter:



```
Enter sentence:
school

school: negative

Negative sentiment :(
Confidence: 6.62%

Raw sentiment: -0.2
-----------------------------------------------
Enter sentence:
ew school

ew: negative
school: negative

Negative sentiment :(
Confidence: 25.12%

Raw sentiment: -0.75
```

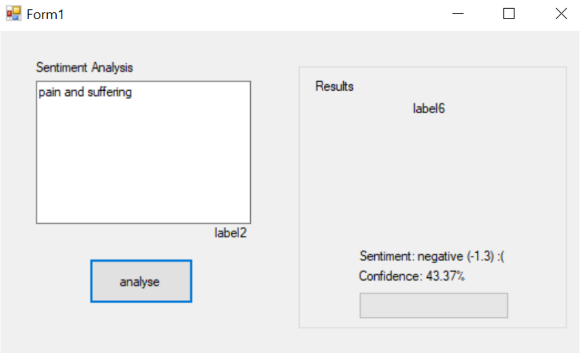
However, very obviously negative or positive sentences are getting low confidence percentages. Maybe should move the range from 3 to 2? It's unlikely that a user would input a sentence that would score above or below 2 and -2, unless it's intentional. Currently keeping it at 3 just to be safe.

Rounded the percentage and raw sentiment to 2dp.

Next, maybe should move program over to forms, instead of console so I can have a UI?

28/4

Started moving project over to windows form, and designed a basic UI:

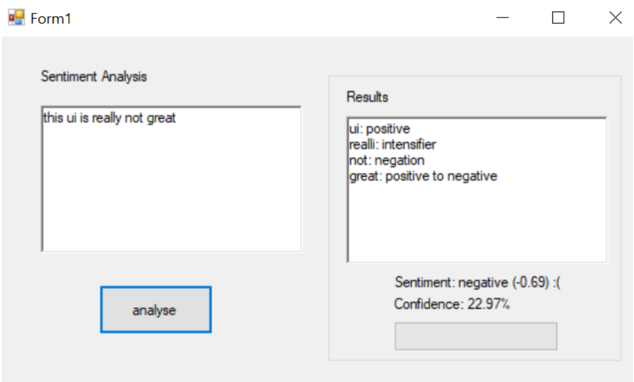


pain and suffering

Sentiment: negative (-1.3) :(
Confidence: 43.37%

Planning to add the word list, word count, and confidence bar later. Need to figure out a way to extend this to at least four screens, and make each screen look nicer.

Added word list, and changed the textboxes to rich textboxes (include scroll bar if content is too large for box).



this ui is really not great

```
ui: positive
realli: intensifier
not: negation
great: positive to negative
```

Sentiment: negative (-0.69) :(

Confidence: 22.97%

Finished the confidence bar. Planning to add a confidence bar cap at 100% to prevent the program from exceeding 100%.

10/5

Implemented confidence percentage cap. Made multiple forms (main menu, statement input/output pages). Also changed input processing so that newline characters are treated as spaces. Created a form just for testing a database to find the accuracy of my program (planning to add word count module to this page later). However, it's very resource intensive, and the UI tends to freeze during the testing. Forms now open in the same location each time.

11/5

Program can now take in an address for a document, and split each line into sentences. For now, the mainline for document input can only return an array of unprocessed sentences from the input document.

Can scan through document, then return an array of arrays (an array for each sentence, with an array with every processed word inside). Gives the user the option of separating each element by sentence, or by newlines. The sentence function is meant for paragraphs of related text, while the newline function is meant for lines of unrelated text (such as a list of tweets).
To help with the program's inefficiency, removed some loops and arrays within the sentence cleaning modules, and replaced them with much more efficient lists that are converted to arrays later in the program. Next, I need to create the document output page, and clean up the GUI (fix labels/positions of screen elements) and make everything look more consistent and just nicer.

Also tried testing the accuracy of the program using the new testing module page, but within 10 seconds of running, the program crashes, and, when it is running, is only showing a 50% accuracy rate. Need to look into that

13/5

Implementing the output page for document input. Looking for a way to calculate the overall document sentiment. Since each sentence is being calculated separately, should I make each sentence weigh in equally? Or change the weighting of each sentence based on the intensity of the sentiment of that sentence? Or count the number of positive/negative sentences and display which one outnumbers the other?
https://blogs.oracle.com/javamagazine/post/java-sentiment-analysis-multisentence-text-block

https://medium.com/swlh/we-need-to-talk-about-sentiment-analysis-9d1f20f2ebfb

Maybe I should just implement all 3 - make the weighted option the main sentiment score, and add the average and count method in the further results section.

Also want to incorporate a shift in tone detection.

19/5

To implement a shift in tone detection, any words in between “but” and any sentence ending punctuation should have their sentiment increased. This way, a sentence like “I heard the movie was terrible, but it was not bad at all.” will have an overall positive tone, rather than negative. The token “terrible” will have less of an impact than “not bad at all”.

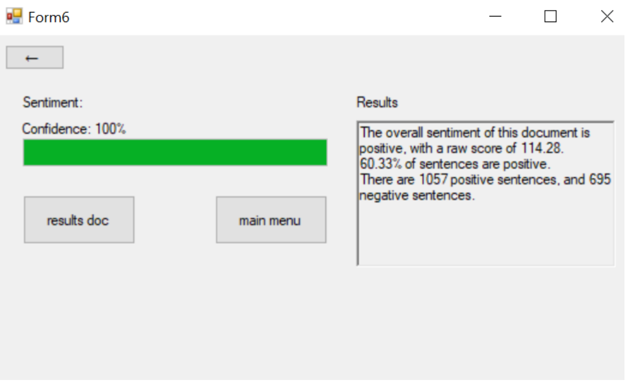
24/5

Implemented a tone shift detector for both statement and document scan! In a statement scan, the tone detector is triggered by “but”, and is applicable for the next words until either a .?! or a newline is detected. In document scan, the tone shift can only be stopped by .?!. Got rid of a ton of logic errors within the tone shift detector. To do next:

- Finish document mode: loop the calculations for each line/sentence
- Gather and present output from document scan, including an overall confidence level, overall sentiment, and sentiment count (decided not to include weighted option for now)
- Loading bar
- Input output (each *original* sentence + sentiment score + rating of each word) into a text doc and attach document to a button in ui
- Improve gui design (only if I have time)
- Fix testing page

25/5

Document mode can now scan through all sentences or lines in a document, and provide a brief overview of the sentiment of the document. As for the efficiency, it could process a movie script in around 25 seconds:

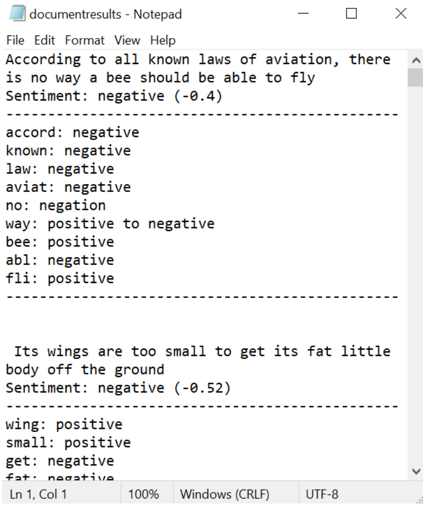


Sentiment:

Confidence: 100%

The overall sentiment of this document is positive, with a raw score of 114.28.
60.33% of sentences are positive.
There are 1057 positive sentences, and 695 negative sentences.

Program can output results into a text document, which can be opened using the button.



```
According to all known laws of aviation, there
is no way a bee should be able to fly
Sentiment: negative (-0.4)
-----------------------------------------------
accord: negative
known: negative
law: negative
aviat: negative
no: negation
way: positive to negative
bee: positive
abl: negative
fli: positive
-----------------------------------------------


 Its wings are too small to get its fat little
body off the ground
Sentiment: negative (-0.52)
-----------------------------------------------
wing: positive
small: positive
get: negative
fat: negative
```

Testing page and document scan page are working, but not efficiently. Looking into background workers, allowing processes to be run while the GUI is updating.

27/5

Implemented a background worker that runs during document scan and the testing page. This means that the GUI will stay responsive (including the progress bar) during processing.

The testing page shows that the program is running at a constant 75-76% accuracy rate, after analysing around 5% of the testing document in 3 mins (from a database of around 100,000 tweets, a rate of roughly 28 tweets/sec). The program is currently better at correctly detecting negative tweets than positive ones - planning to tweak the sentiment analysis program later to try to fix this. Adding in a word count meter to statement mode, to limit processing time so I don’t need to add a background worker.

3/6

Replaced testing file with a newer, shorter file, so that the testing page can finish within 10 seconds, easier to see accuracy of program + removes any prior bias. Finished with a 78% accuracy rate. To try to improve the accuracy of the program, will try to test:

- Ignore any words with a sentiment rating between -0.4 to 0.4 to try to take out the neutral nouns that don’t add much to the sentiment, giving more weight to the more extreme sentimental words (ended up reducing accuracy)
- The word with the most extreme sentiment in the sentence has its rating multiplied (reduced accuracy)
- Tweaking negations and intensifiers to work as tone shift: instead of just affecting the word after it, it will affect all words until the next punctuation token (reduced accuracy)

Not sure what to do to raise the program’s accuracy percentage (preferably by 5%, to reach goal of 80%)

7/6

Changed the way negations, intensifiers and tone shifts work by calculating overall sentiment before the actual intensifying. Accuracy rate up to 81.67%.

8/6
Modified the UI to look better, only on the main menu. Planning to implement changes across the whole system. Changed colours, fonts and buttons. Also want to embed text files into the program, to make it able to be accessed through any device. Testing program also keeps running even when the testing form exits.

10/6
Embedded text files now work! Had to change the way my program scanned through the document, by splitting the whole document, instead of reading line by line. This in turn has made my program very slow. Improved this by setting wordcount files to be global variables, so that accessing the file doesn't need to happen every iteration, only once when the form is opened.

11/6
Brought embedded files into the whole program, the program should now be able to run independently, not relying on local files. Output file for document mode should also not rely on a file path, but now will be written onto C://Temp. Finally, I changed the GUI to make it look a bit better.

18/6
Fixed a bug in document mode that caused the program to crash when the document output page was opened then closed.

Social and Ethical Considerations:

Quality:

- Correctness: at least 80% accuracy rate while testing, usage of real-world collected data, large datasets
- Reliability: extensive debugging to ensure software will not crash
- Efficiency: Minimised processing time for both statement and document mode (for most files will take < 20 secs), embedded database files, pre-processed
- Maintainability: use of intrinsic (indenting, meaningful variable names) and internal (comments) documentation
- Flexibility: able to take in wide range of data (in text form), with a large dictionary of words able to recognised
- Portability: all external files are embedded into the program, so program should be able to work on any device (other than the document output file, may not work on non-Windows OS)

Intuitiveness:

- Consistency: colours, fonts, buttons, element locations have been kept consistent throughout each form
- Appropriate labels and messages: labels should be all self explanatory to their corresponding screen elements
- Logical paths between forms: back button, main menu button, etc. all lead to appropriate screens

Ergonomics:

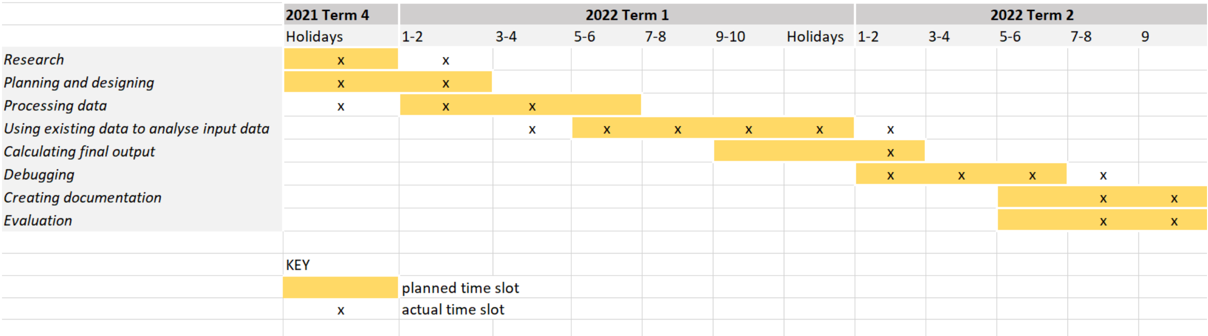
- Interface elements: appropriate use of different UI elements, such as textboxes, buttons, labels, dropdown boxes, progress bars, to communicate information to and from user
- Colour, font: attempted to make program look nice
- Alignment: aligning elements in middle of the screen, inline with neighbouring elements, to keep the flow of screen design interrupted
- White space: not cluttering screen with elements, keeping everything almost minimalistic, only include needed elements
- Consistent method of reversing actions: all screens but main menu have a back button, to enable users to undo or go back. Give users control, more freedom

Piracy and copyright/intellectual property:

- Once compiled into executable file, source code should not be accessible, protecting against plagiarism
- External code used within project, all have licences that allow for free use by developers:
    - https://stackoverflow.com/questions/1297264/using-custom-fonts-on-a-label-on-winforms
    - https://www.nuget.org/packages/Porter2Stemmer/

Privacy: no personal information collected, data not accessible from other devices - inputs and outputs either not stored or stored in local temporary files.

Gantt Chart:



	2021 Term 4	2022 Term 1						2022 Term 2				
	Holidays	1-2	3-4	5-6	7-8	9-10	Holidays	1-2	3-4	5-6	7-8	9
Research	🟨 x	x										
Planning and designing	🟨 x	🟨 x										
Processing data	x	🟨 x	🟨 x	🟨	🟨							
Using existing data to analyse input data			x	🟨 x	🟨 x	🟨 x	🟨 x	x				
Calculating final output						🟨	🟨	🟨 x				
Debugging								🟨 x	🟨 x	🟨 x	x	
Creating documentation										🟨	🟨 x	🟨 x
Evaluation										🟨	🟨 x	🟨 x
	KEY											
	🟨	planned time slot										
	x	actual time slot										

Documentation

Needs

Data analysts can use user generated data (from social media, reviews, comments, etc.) to analyse the wider public opinion on certain topics, and derive meaning from a wide overview of users' general attitudes. This is especially helpful in companies that want to focus on customer service or marketing. However, the amount of data cannot be manually sorted, and instead needs an automated system to monitor and process the constant stream of data. A sentiment analysis program should be able to scan through statements, and be able to accurately classify each statement as either **positive or negative based on the overall tone of the text**, which can then be manually analysed by companies. Because sentiment analysis can be automated, decisions can be made based on a significant amount of data rather than just intuition.

The needs involved in this sentiment analysis program would be for the program to be able to take in input sentence(s) from the user, and output a sentiment rating (positive, neutral, negative). This sentiment rating should be calculated from cleaning the statement (removing unnecessary punctuation, words, spaces, etc.), and checking each word against a database of words and their respective sentiments. This is to ensure an accurate output, which is also a need - constant inaccurate output would lead to an inadequate program.

Furthermore, response time of the program is important, since the purpose is to automate the process of analysing the sentiment of data (needs to be faster than manual sorting). This means that an almost instant return is needed for every statement, so that the program is able to sort through many statements in a short time.

Finally, a user interface is needed for the program to be able to take in input, and display an output for users. This can either be a command line interface or a graphical user interface (harder to design, but easier to use, more user friendly). This project will most likely have a GUI.

Objectives

1. To take in a statement, and clean it into its root form
2. Use existing datasets to create a dictionary/weighting system of positive and negative words
3. Use the data to analyse the input statement word by word, and calculate a final rating
4. Output a positive or negative rating for the input statement

Boundaries

- Basic processing features of any sentiment analysis program: taking in user input, and calculating/displaying a sentiment output
- Text stemming and cleaning for all inputs to allow for easier and more accurate processing
- Output calculated from a pre sorted database using probability
- User input the program can process from individual input from textboxes (statements input one by one), and from document scan (user input address of document, program scans document line by line, outputs sentiment summary)
- As part of the output in document scan, create a new output text document with processed sentences and their sentiments

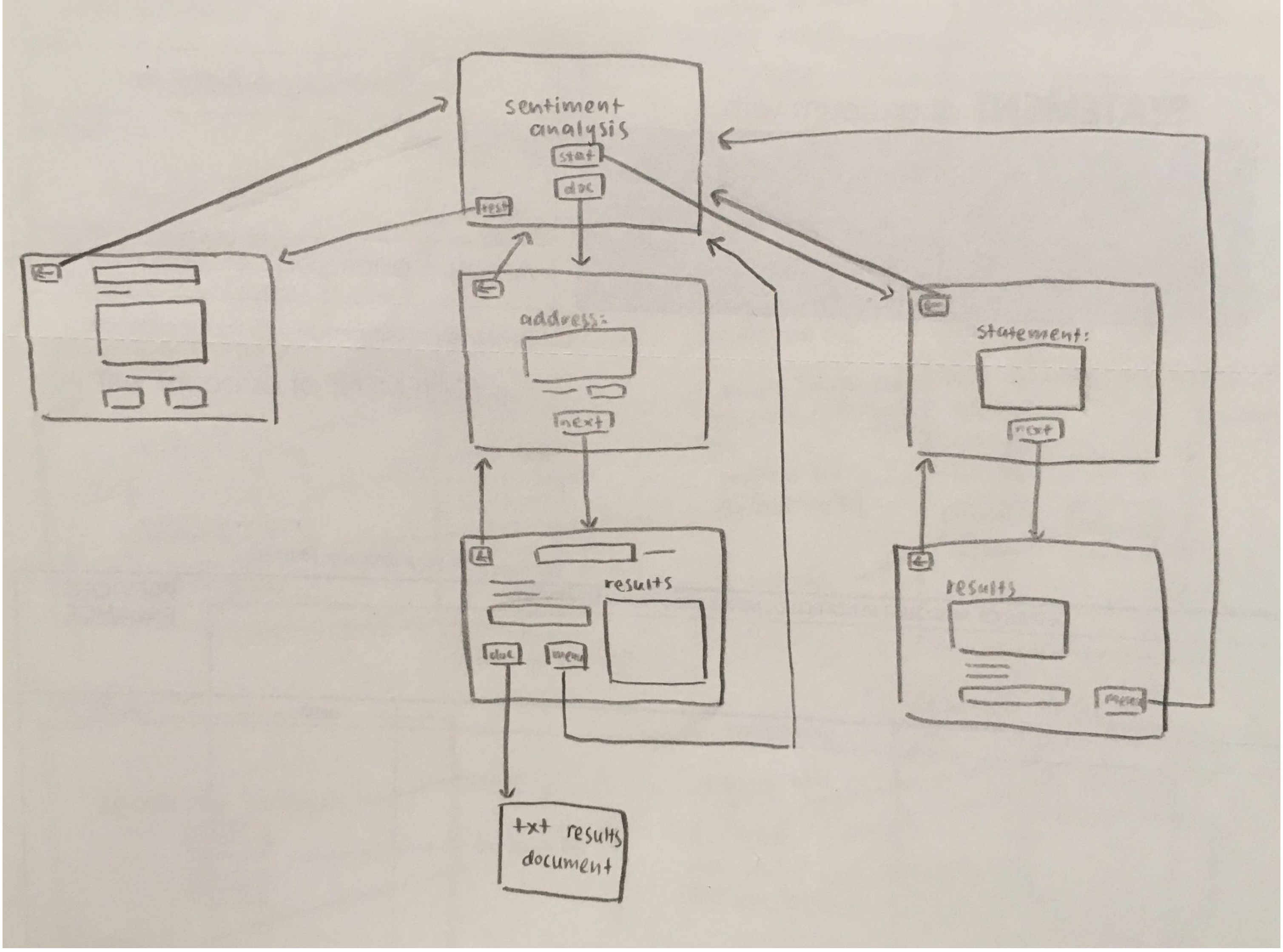
- During document scan, a progress bar that displays how far through the program is in the document (helpful during longer documents, give estimation of time frame needed for user)
- Confidence bar: shows how confident the program is in the sentiment output (the intensity of the sentiment in the statement) as a percentage
- Graphical user interface: buttons, textboxes, etc. to allow for ease of use, more intuitive design, multiple screens, and overall looks
- Rating of individual words (in individual user input mode): output includes display of words in sentences and their corresponding sentiments. Allows user to see how sentence is analysed
- Only be able to take in and calculate input of english sentences
- To be run on computers (Windows OS)

Evaluation Criteria

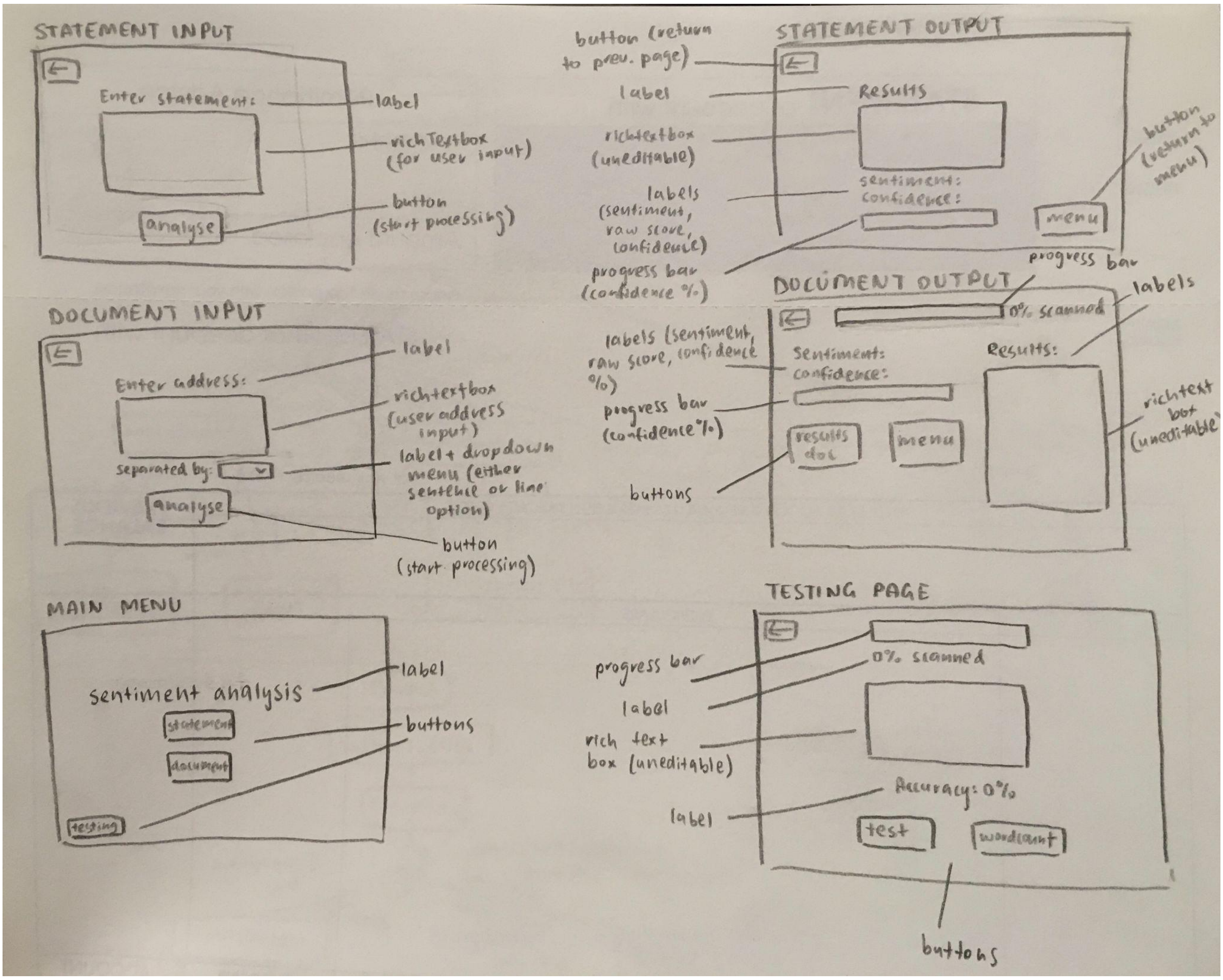
If the project is successful it should:

- Have an accuracy rate of at least 80% (from testing subprogram), to ensure an overall accurate output.
- Be able to sort through documents as well as taking in individual statements: should have two modes, document (get user input of text document address, scan through document, analyse sentiment of every line, and output overall document sentiment), and user input (analyse sentiment of user input into textbox).
- During document scanning, have < 1 min processing time for reasonably sized documents - ensuring program is efficient and faster than manual sorting
- Be able to take in input, produce sentiment output based on calculated weightings from database
- Be able to clean a statement into its root form + remove punctuation, emojis, stopwords, twitter handles/hashtags - to make it easier for program to process (also ensures better program efficiency: unneeded words/characters are filtered out, minimising the necessary processing, and accuracy: identical words in different forms are combined (eg. “walk”, “walking”, “walked”, “walks”), able to be treated and calculated as same word.)
- Have a graphical user interface that has four or more screens, making use of textboxes, buttons and progress bars, etc. Should have intuitive (easy to find) locations for user to enter in input, and view output

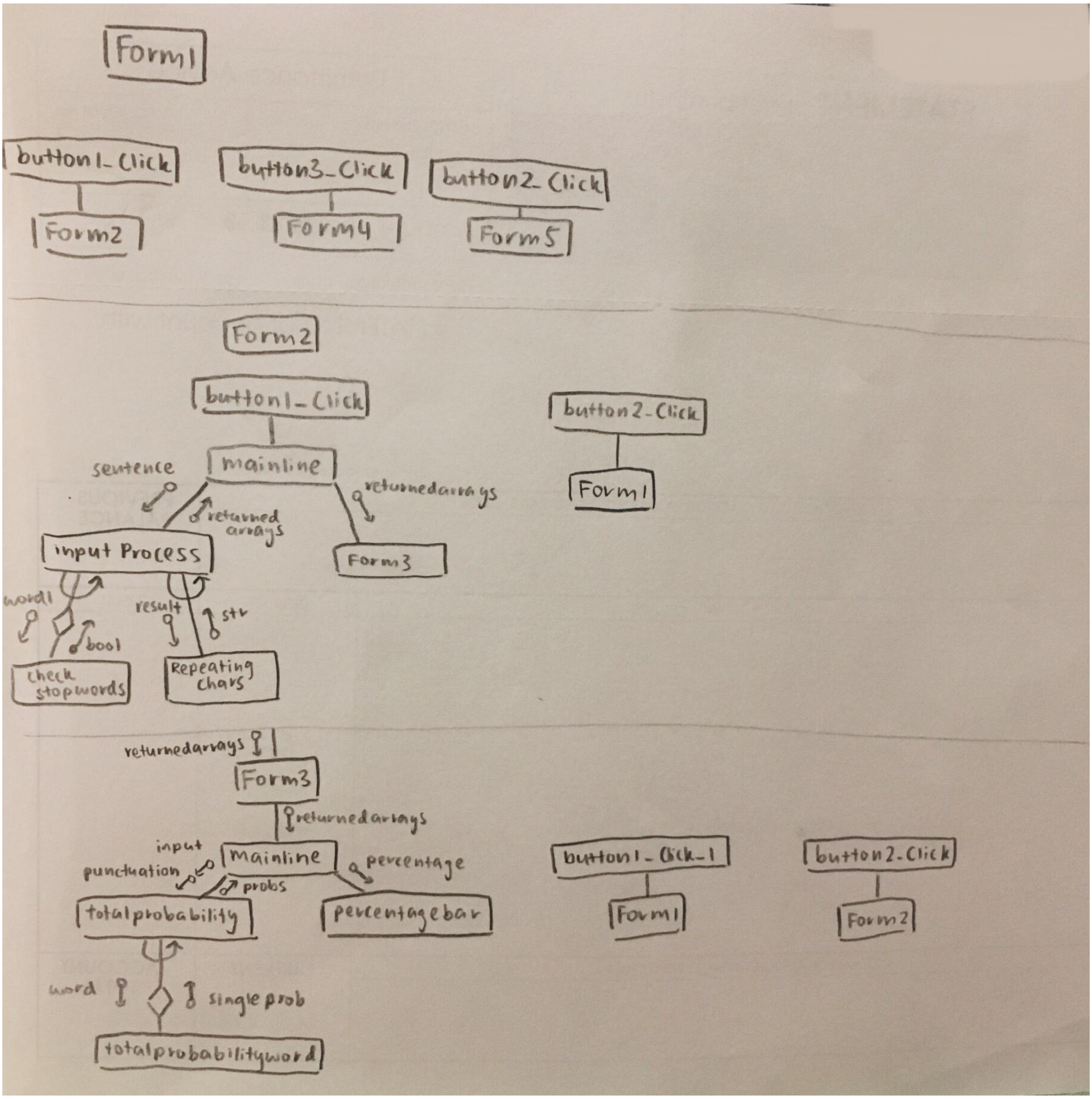
Storyboard

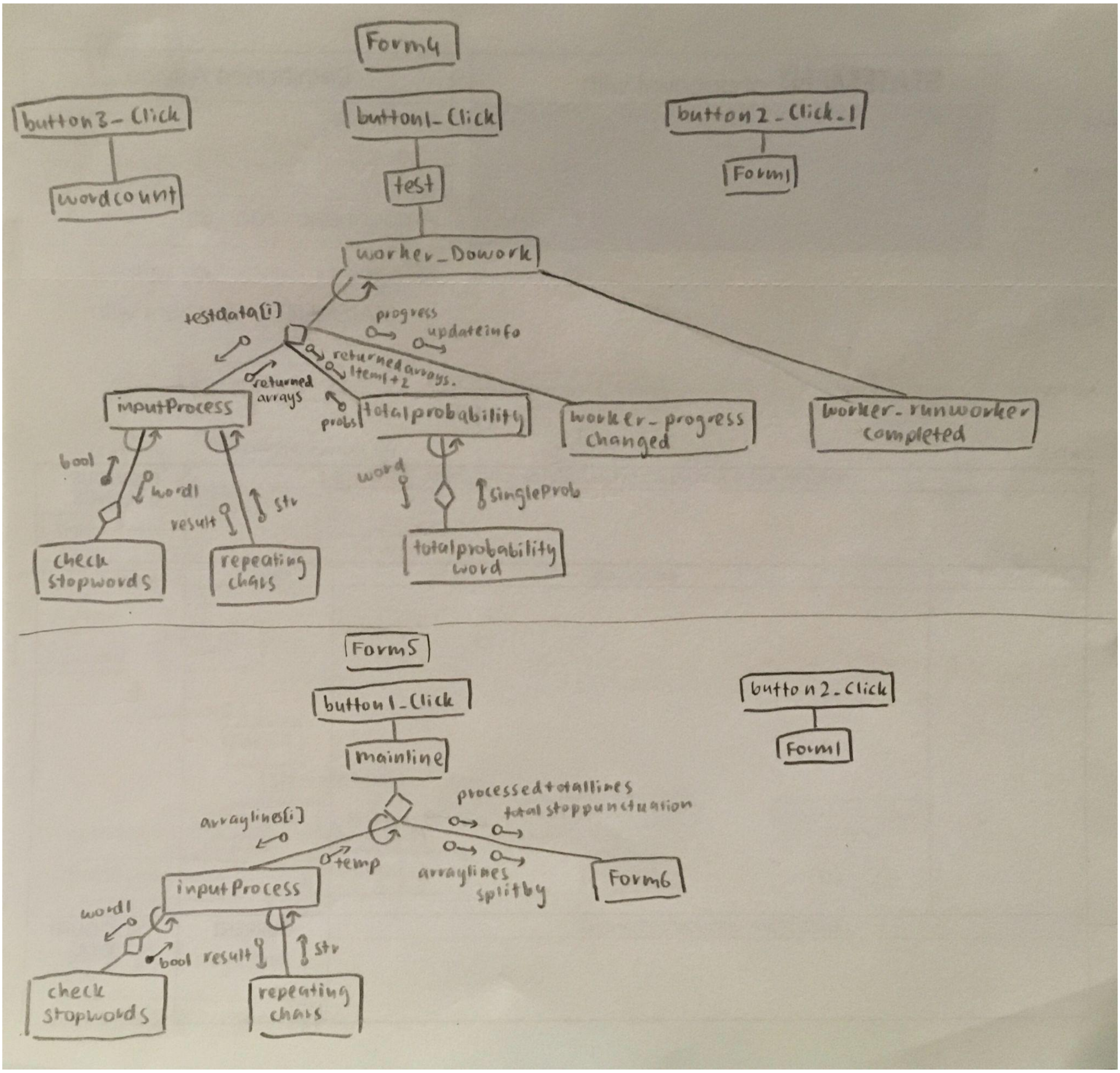

sentiment analysis
stat
doc
test
address:
next
results
doc
menu
txt results document
statement:
next
results
menu

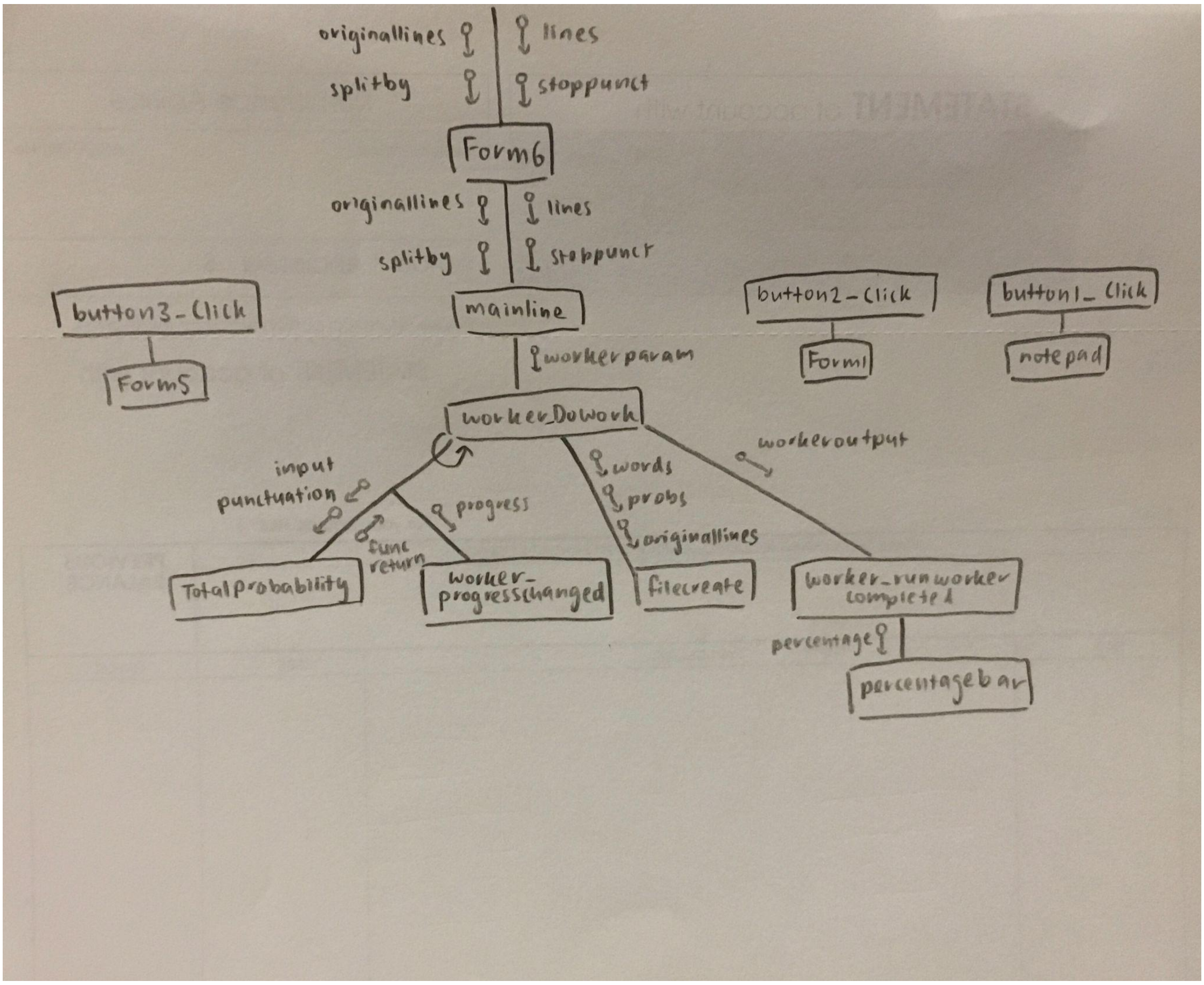
Screen Designs



Structure Charts




Form4
button3-Click
wordcount
button1-Click
test
worker_Dowork
testdata[i]
progress
updateinfo
returnedarrays.
Item1+2
returned
arrays
probs
inputProcess
totalprobability
worker-progress
Changed
worker-runworker
completed
button2-Click-1
Form1
bool
word1
result
str
check
stopwords
repeating
chars
word
singleProb
totalprobability
word
Form5
button1-Click
mainline
processedtotallines
total stoppunctuation
arraylines[i]
temp
arraylines
splitby
Form6
button2-Click
Form1
inputProcess
word1
bool
result
str
check
stopwords
repeating
chars


originallines
lines
splitby
stoppunct
Form6
originallines
lines
splitby
stoppunct
button3_Click
mainline
button2_Click
button1_Click
Form5
workerparam
Form1
notepad
worker_DoWork
workeroutput
input
punctuation
progress
words
probs
func return
originallines
Totalprobability
worker_progresschanged
filecreate
worker_runworkercompleted
percentage
percentagebar

Data Dictionary

Name	Data Type	Form	Subprogram	Scope	Description
pfc	PrivateFontCollection	1	None	Global	For custom font
fontlength	int	1	Form1_Load	Local	Font length, for custom font
fontdata	byte[]	1	Form1_Load	Local	Buffer to read custom font into
data	IntPtr	1	Form1_Load	Local	Memory block for custom font data
statementinput	Class, Form2	1	button1_Click	Local	To open Form2
test	Class, Form4	1	button3_Click	Local	To open Form4
documentinput	Class, Form5	1	button2_Click	Local	To open Form5
chars	string[]	2, 4, 5	None	Global	Array of punctuation for text filtering
stopwords	string[]	2, 4, 5	None	Global	Array of stopwords for text filtering
alphabet	string[]	2, 4, 5	None	Global	Array of letters and newlines for text filtering
sentence	string	2	mainline	Local	Statement user input
returnedarrays	var	2	mainline	Local	Arrays returned from inputProcess
statementouput	Class, Form3	2	mainline	Local	To open Form3
input	string	2, 4, 5	inputProcess	Local	User input
count	int	2, 4, 5	inputProcess	Local	To keep track of index when adding stemmed words into array

letter	char	2, 4, 5	inputProcess	Local	Character in input in loop
output	string[]	2, 4, 5	inputProcess	Local	Cleaned but not stemmed input
stemmer	var	2, 4, 5	inputProcess	Local	Stemmer
newOutput	List<string>	2, 4, 5	inputProcess	Local	List to place words that can be stemmed (all words that are not blank spaces, user tags or hashtags)
punctuationarray	List<int>	2, 4, 5	inputProcess	Local	List of indexes of words with stopping punctuation
word	string	2, 4, 5	inputProcess	Local	Element of output, for loop
word1	string	2, 4, 5	inputProcess	Local	To be able to modify word
newoutputarray	string[]	2, 4, 5	inputProcess	Local	Stemmed and cleaned words in sentence
word	string	2, 4, 5	inputProcess	Local	Element of newoutput array, for loop
result	var	2, 4, 5	inputProcess	Local	Stemmed word
word	string	2, 4, 5	checkStopwords	Local	Word to be checked against
stopword	string	2, 4, 5	checkStopwords	Local	Element in stopwords, for loop
str	string	2, 4, 5	RepeatingChars	Local	Word to be processed
i	int	2, 4, 5	RepeatingChars	Local	Index of word
chars	char[]	2, 4, 5	RepeatingChars	Local	Array of characters in word
x	int	2, 4, 5	RepeatingChars	Local	Number of indexes to check forward for repeating characters
temp	char	2, 4, 5	RepeatingChars	Local	Temporary variable to hold character to compare next characters to
wordcount	string	2	openDatabase	Local	File address to write database wordcounts to
fileName	var	2	openDatabase	Local	Database address

linecount	float	2	openDatabase	Local	To keep track of how many lines program has scanned
linecountstop	int	2	openDatabase	Local	Line count at which the program should stop, end of database
unique_words	Dictionary<string, int[]>	2	openDatabase	Local	Unique word, array of word counts (instances where word appears in negative/positive tweets)
fs	Class, FileStream	2	openDatabase	Local	FileStream
sr	var	2	openDatabase	Local	StreamReader
line	string[]	2	openDatabase	Local	Current line read on database
processed_line	string[]	2	openDatabase	Local	Stemmed and cleaned line from database
word	string	2	openDatabase	Local	Foreach loop iteration, elements in processed_line
word1	string	2	openDatabase	Local	Each word in processed_line
temp_neg	int[]	2	openDatabase	Local	New negative entry template in unique_words dictionary
temp_pos	int[]	2	openDatabase	Local	New positive entry template in unique_words dictionary
key	string	2	OpenDatabase	Local	Foreach loop iteration, key in unique_words
mainmenu	Class, Form1	2	button2_Click	Local	To open Form1
returnedarrays	Tuple<string[], int[]>	3	Form3	Local	To pass stemmed output from previous form
assembly	var	3	Form3	Local	To access embedded resources
resourceName	var	3	Form3	Local	Embedded resource addresses
stream	Class, Stream	3	Form3	Local	To access streamreader

reader	Class, StreamReader	3	Form3	Local	
wordcount	string[]	3, 4, 6	None	Global	Array of wordcounts from file 1
wordcount1	string[]	3, 4, 6	None	Global	Array of wordcounts from file 2
negations	string[]	3, 4, 6	None	Global	Array of negations
intensifiers	string[]	3, 4, 6	None	Global	Array of intensifiers
returnedarrays	Tuple<string[], int[]>	3	mainline	Local	Stemmed array of input sentence + array of punctuation indexes
input	string[]	3	mainline	Local	Stemmed input sentence
punctuation	int[]	3	mainline	Local	Array of indexes of words with stopping punctuation
range	int	3	mainline	Local	Boundaries of sentient score, used to calculate confidence percentage
probs	double	3	mainline	Local	Probability of sentence
percentage	double	3	mainline	Local	Confidence percentage
sentence	string[]	3, 4, 6	TotalProbability	Local	Sentence to calculate probability from
punctuation	int[]	3, 4, 6	TotalProbability	Local	Array of indexes of words with stopping punctuation, used for tone changes
prob	double	3, 4, 6	TotalProbability	Local	Total probability of sentence
singleprob	double[]	3, 4, 6	TotalProbability	Local	Probability of a word (positive, negative)
negationstatus	bool	3, 4, 6	TotalProbability	Local	Whether to negate next word
intensitystatus	int	3, 4, 6	TotalProbability	Local	Factor to intensify next word by

toneshift	bool	3, 4, 6	TotalProbability	Local	Whether word should be intensified because of a tone shift
index	int	3, 4, 6	TotalProbability	Local	To keep track of index while iterating through each word in sentence
index1	int	3, 4, 6	TotalProbability	Local	To keep track of index while iterating through each element in punctuation array
wordratings	string[]	3, 4, 6	TotalProbability	Local	The sentiment/type/process carried out on each word
word	string	3, 4, 6	TotalProbability	Local	Each word in sentence, used for loop
boolean	string	3, 4, 6	TotalProbability	Local	The sentiment/type/process carried out on the word
wordprob	double	3, 4, 6	TotalProbability	Local	The overall probability of a word
word	string	3, 4, 6	TotalProbability Word	Local	Word to find in database
negcount	double	3, 4, 6	TotalProbability Word	Local	Number of times word appears in negative tweets
poscount	double	3, 4, 6	TotalProbability Word	Local	Number of times word appears in positive tweets
totalnegcount	double	3, 4, 6	TotalProbability Word	Local	Total number of words in negative tweets
totalposcount	double	3, 4, 6	TotalProbability Word	Local	Total number of words in positive tweets
linecount	int	3, 4, 6	TotalProbability Word	Local	Keep track of how many lines scanned in document
negprob	double	3, 4, 6	TotalProbability Word	Local	Probability of the word, given that the word is negative
posprob	double	3, 4, 6	TotalProbability Word	Local	Probability of the word, given that the word is positive
mainmenu	Class, Form1	3	button1_Click_1	Local	Open Form1
statem	Form2	3	button2_Click	Local	Open Form2

entinput					
assembly	var	4	Form4_Load	Local	To access embedded resources
resourceName	var	4	Form4_Load	Local	Embedded resource addresses
stream	Class, Stream	4	Form4_Load	Local	To access streamreader
reader	Class, StreamReader	4	Form4_Load	Local	
worker	Class, BackgroundWorker	4	test	Local	Background worker
negcount	int	4	wordCount	Local	Total count of words in negative tweets
poscount	int	4	wordCount	Local	Total count of words in positive tweets
word	string	4	wordCount	Local	Each unique word in wordcount database, for loop
splitline	string[]	4	wordCount	Local	Split each line in wordcount file
word1	string	4	wordCount	Local	Each unique word in wordcount1 database, for loop
assembly	var	4	worker_DoWork	Local	To access embedded files
resoureName	var	4	worker_DoWork	Local	Embedded file address
testdata	string[]	4	worker_DoWork	Local	Array of lines from file of tweets, used for testing
stream	Class, Stream	4	worker_DoWork	Local	StreamReader
reader	Class, StreamReader	4	worker_DoWork	Local	

sentimentdata	string[]	4	worker_DoWork	Local	Array of sentiments of file of tweets, used to verify testing
false_neg	int	4	worker_DoWork	Local	Count of actual positive lines that program has flagged negative
false_pos	int	4	worker_DoWork	Local	Count of actual negative lines that program has flagged positive
true_neg	int	4	worker_DoWork	Local	Count of actual negative lines that program has flagged negative
true_pos	int	4	worker_DoWork	Local	Count of actual positive lines that program has flagged positive
linecount	double	4	worker_DoWork	Local	Keep track of how many lines program has scanned through
actualsentiment	int	4	worker_DoWork	Local	Actual sentiment of tweet
sentiment	int	4	worker_DoWork	Local	Calculated sentiment of tweet
i	int	4	worker_DoWork	Local	Count iterations, used in for loop
returnedarrays	var	4	worker_DoWork	Local	Arrays returned from inputProcess
probs	double	4	worker_DoWork	Local	Probability of sentence returned from TotalProbability
filesentiment	string	4	worker_DoWork	Local	Actual sentiment of tweet, taken from file
progress	int	4	worker_DoWork	Local	Percentage of file completed
updateinfo	int[]	4	worker_DoWork	Local	Counts of false neg/pos, true neg/pos and progress to send to worker to update GUI to display
updateinfo	int[]	4	worker_ProgressChanged	Local	Counts of false neg/pos, true neg/pos and progress to update GUI to display
false_neg	int	4	worker_ProgressChanged	Local	Count of actual positive lines that program has flagged negative

false_pos	int	4	worker_ProgressChanged	Local	Count of actual negative lines that program has flagged positive
true_neg	int	4	worker_ProgressChanged	Local	Count of actual negative lines that program has flagged negative
true_pos	int	4	worker_ProgressChanged	Local	Count of actual positive lines that program has flagged positive
accuracy	double	4	worker_ProgressChanged	Local	Accuracy percentage of program
mainmenu	Class, Form1	4	button2_Click_1	Local	To open Form1
mainmenu	Class, Form1	5	button2_Click	Local	To open Form1
address	string	5	mainline	Local	Input address of txt file
string	splitby	5	mainline	Local	Whether to split by sentence or line
fs	Class, Filestream	5	mainline	Local	Filestream
sr	var	5	mainline	Local	StreamReader
line	string	5	mainline	Local	Each line from document
totallines	List<string>	5	mainline	Local	List of all lines in document
currentline	List<string>	5	mainline	Local	All sentences in line
arraylines	string[]	5	mainline	Local	totallines but as an array
processedtotallines	string[][]	5	mainline	Local	Arraylines but processed and also split into words
totalstoppunctuation	int[][]	5	mainline	Local	Array of lines of locations of stopping punctuation
i	int	5	mainline	Local	Iteration through for loop
temp	var	5	mainline	Local	Temporary variable to hold output

					processed line from inputProcess
documentoutput	Class, Form6	5	mainline	Local	To open Form6
lines	string[][]	6	Form6	Local	Lines of processed input split into words
stoppunct	int[][]	6	Form6	Local	Array of lines of index locations of stopping punctuation
originallines	string[]	6	Form6	Local	Original lines of input
splitby	string	6	Form6	Local	Whether input split by sentences or lines
outputfileName	string	6	None	Global	The file address for the output file
lines	string[][]	6	mainline	Local	Lines of processed input split into words
stoppunct	int[][]	6	mainline	Local	Array of lines of index locations of stopping punctuation
originallines	string[]	6	mainline	Local	Original lines of input
splitby	string	6	mainline	Local	Whether input split by sentences or lines
worker	Class, BackgroundWorker	6	mainline	Local	Background
workerparam	dynamic[]	6	mainline	Local	Parameters to pass to background worker
originallines	string[]	6	filecreate	Local	Original lines of input
wordratings	string[][]	6	filecreate	Local	The sentiment/type/process carried out on each word
sentenceratings	double[]	6	filecreate	Local	Sentiment ratings of each sentence
index	int	6	filecreate	Local	To keep track of how many lines

					have been written onto document
rating	string	6	filecreate	Local	Sentiment of sentence (pos/neg/neut)
sw	Class, StreamWriter	6	filecreate	Local	StreamWriter
line	string	6	filecreate	Local	Each line in originallines, used for loop
sentiment	double	6	filecreate	Local	Raw sentiment rating of each line
word	string	6	filecreate	Local	Each word in wordratings, used for loop
documentinput	Class, Form5	6	filecreate	Local	To open Form5
mainmenu	Class, Form1	6	filecreate	Local	To open Form1
parameters	dynamic[]	6	worker_DoWork	Local	Parameters from mainline
lines	string[][]	6	worker_DoWork	Local	Lines split into processed words
stoppunct	int[][]	6	worker_DoWork	Local	Locations of punctuations, separated by lines
splitby	string	6	worker_DoWork	Local	Whether lines are separated by sentences or lines
originallines	string[]	6	worker_DoWork	Local	Original unprocessed lines
lineindex	int	6	worker_DoWork	Local	The index of current line, to keep track of lines being processed
probs	double[]	6	worker_DoWork	Local	Probabilities of sentences
words	string[][]	6	worker_DoWork	Local	Lines of words and their individual ratings
poscount	double	6	worker_DoWork	Local	Number of positive lines in document
negco	double	6	worker_DoWor	Local	Number of negative lines in

unt			k		document
ncount	double	6	worker_DoWork	Local	Number of neutral lines in document
percentage	double	6	worker_DoWork	Local	Confidence percentage
range	double	6	worker_DoWork	Local	Boundaries for sentiment score, used in confidence percentage
line	string[]	6	worker_DoWork	Local	Each line in lines, used for loop
input[]	string[]	6	worker_DoWork	Local	Line in lines
punctuation	int[]	6	worker_DoWork	Local	Array of locations of stopping punctuation in stoppunct
funcreturn	var	6	worker_DoWork	Local	Returned variables from TotalProbability
probability	double	6	worker_DoWork	Local	Probability of line
progress	int	6	worker_DoWork	Local	Percentage of document scanned
totalprob	double	6	worker_DoWork	Local	Total probability of whole document
score	double	6	worker_DoWork	Local	Each probability of each line in probs, used for loop
workeroutput	dynamic[]	6	worker_DoWork	Local	Parameters passed as worker finishes to be displayed onto GUI
sentiment	string	6	Worker_RunWorkerCompleted	Local	Overall sentiment rating of entire document
workeroutput	dynamic[]	6	Worker_RunWorkerCompleted	Local	Parameters to be displayed onto GUI
percentage	double	6	Worker_RunWorkerCompleted	Local	Overall confidence percentage of document
totalprob	double	6	Worker_RunWorkerCompleted	Local	Overall raw sentiment score of document
poscount	double	6	Worker_RunWorkerCompleted	Local	Total positive statements in document

negcount	double	6	Worker_RunWorkerCompleted	Local	Total negative statements in document
ncount	double	6	Worker_RunWorkerCompleted	Local	Total neutral statements in document
splitby	string	6	Worker_RunWorkerCompleted	Local	Whether document split by sentences or lines
pospercent	double	6	Worker_RunWorkerCompleted	Local	Percentage of sentences/lines that are positive
assembly	var	6	Form6	Local	To access embedded resources
resourceName	var	6	Form6	Local	Embedded resource addresses
stream	Class, Stream	6	Form6	Local	To access streamreader
reader	Class, StreamReader	6	Form6	Local	

Evaluation

In comparison with my initial objectives set at the beginning of the process and my evaluation criteria, my program fulfils the overwhelming majority of the goals. Since the objectives were written last year, I overestimated the amount of time each section would take to program, so my objectives were less detailed than what I really managed to achieve. This meant that my program easily aligns with all the objectives. Similarly, my program fulfils all my evaluation criteria, except for:

- Have < 1 min processing time for reasonably sized documents - for most documents, my program can finish processing within 30 seconds. However, in the context of what I designed my program for (scanning through thousands of tweets to gain user generated feedback for companies), my program needs to be much more efficient. This could be improved in the future by better hardware (processing power) or by further optimising the code: one way could be by removing the need for some processing (by pre-processing words into a database).
- Be able to clean a statement into its root form. I replaced lemmanisation with stemming early on in the process to increase the program's efficiency, which results in some words being processed into incorrect root forms (since stemming is rule based, while lemmanisation is more complex, takes into account context)
- 80% accuracy rate. Although my program does have an 80% accuracy rate for this particular testing document, I suspect more thorough testing with different documents will show that my program has a lower accuracy rate (especially for negative tweets/reviews, where sarcasm and more complex phrases and sentence structures are more common)

My sentiment analysis program can successfully analyse text, either by user input or by extracting text from a document. It has an 81% accuracy rate, with a higher likelihood of classifying positive tweets correctly than negative. Furthermore, it can recognise different types of words, including negations, intensifiers and "but", and calculate the sentiment accordingly. Its processing time for medium sized documents (such as movie scripts), is mostly less than 30 seconds. The program has a graphical user interface, which means that different screen elements, like progress bars (to track progress and confidence), buttons, and textboxes can be utilised. Through embedded resources, the program is also more portable: is able to be used on any computer, without having to relink its files. Finally, the use of background workers increases the program's efficiency, so that the GUI doesn't freeze or crash during processing, especially during large files.

However, some disadvantages of the project that have yet to be fixed include: the GUI in my project is usable, but looks dated (C# and Visual Studio makes it easy to set up a GUI, but has limitations to how customisable elements in the GUI can be). The program also can't handle complex natural language such as irony or sarcasm, or emojis and misspelt words, all of which are common in digital environments like Twitter. Similarly, words with multiple meanings based on its context are not calculated differently, since the program does not take into account context or placement of any words. Excessively large files (such as the Bible, 2% scanned in a minute) will take extended periods of time to complete.

There were many major problems encountered during implementation. Originally, scanning through the database would've taken half a month. This was fixed by changing lemmanisation to stemming (sacrificing accuracy for program speed) and fixing a statement within a loop, leaving the total processing time at just over 10 minutes. Furthermore, the program crashed when scanning over large documents and during processing in the testing page. This was fixed by implementing background workers, which moved processing to a different thread, ensuring that the GUI and processing can run separately at the same time. Finally, bringing accuracy rate up to 80% was a major issue, which was only temporarily fixed by changing the testing file to a smaller file, and by tweaking some bugs (eg. calculating the sentiment of a word before intensifying or negating).

Throughout the project, not only have I learned how to more efficiently and correctly develop, plan and debug a program, but I have also learned how to use background workers, how to embed files, and how to use multiple forms (creating multiple forms, linking forms and passing parameters). Since my project is based around sentiment analysis and processing natural language, I've been able to research and learn how to clean text, calculate probability, and recognise and manipulate sentiment based on negations, intensifiers and tone shifts.