Report

Introduction:

My system is a chatbot which answers a user's questions, learns the user's name, and can have basic conversations with it. It's capable of distinguishing between each of these intents too. I chose to implement it since it demonstrates many of the key functions of natural language processing such as normalisation, weighting, similarity matching and intent matching.

Background:

Pre-processing is an important beginning in natural language processing. Our chatbot's datasets must be normalised. To begin with, as most developers would agree, stopword removal should be implemented.

Following this a decision of whether stemming or lemmatisation should be applied to our text. Paper (Dwitam and Rusli, 2020) uses a stemming algorithm, it has it's benefits such as speed but sacrifices accuracy which could be used to increase it's 83.03% user acceptance score. I have decided to apply lemmatisation to increase my systems accuracy. This is because stemming simply stems the last few characters of a word which can cause non-existent words to be created, whilst lemmatisation accurately reduces the words accurately to its dictionary/lemma form using a large corpus.

Weighting measures mentioned in the International Conference on Data Management Technologies and Applications (Domeniconi et al, 2015) include binary term weighting and log frequency term weighting. However, I chose to go for TF-IDF weighting since it takes into consideration all the data to calculate its relevance.

Similarity in vector spaces can be measured in different ways. One way, implemented in paper (Su et al, 2017), is through Euclidean distance. Another method is Jaccard index which is used in paper (Oaffas, 2019), the paper compares Jaccard results to cosine results, which is what I will use in my project since it was discovered in their project the cosine similarity answered questions at a 2% higher accuracy.

I used cosine similarity for our functions. It has been used to determine how close the input is to the data in the database and select an intent and particular data such as a question. However, other methods are available, such as neural networks, multinomial naïve bayes algorithm and logistic regression. The latter two were used in the paper (Seyaan et al, 2018) for intent classification in a chatbot. Both were good options, however logistic regression was concluded to be the more accurate method.

Proposed system:

Functionality:

My system includes four main features: identity management, small talk, and information retrieval & question answering which is complemented by intent matching.

It also includes smaller features like saving inputs to grow the database and allow for more accurate future predictions. Another is asking if the user wants a further explanation of answers. Similarity ranking allows for the next best answer to be used if the user determines the question isn't answered in the question answering feature.

Our chatbot continuously runs, whilst ensuring any input with a low similarity score response with a misunderstanding message. The while loop ends once we input goodbye/bye.

Originality:

My system asks if the chatbot's answer is acceptable (if it's a question answer intent), to continuously improve our system, if the user inputs 'y', the question is added to the database (providing the input isn't identical to any questions in the database). For example, a user may input "How does one play spades", which may be correctly suggested to be "How do you play spades" by the system. "How does one play spades" is added to the database to increase accuracy of other future question predictions which may not have been identified with the original question, but is now with our second.

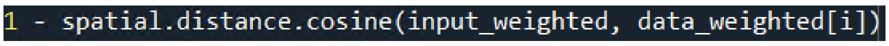
Response 'n' moves onto the next best response using the rankings providing the similarity is above a particular threshold.

Another original feature is the system asks the user if it would like to continue its explanation of a question or not, if it has multiple parts (as seen in the QA dataset) an explanation can continue with the next row.

The vast majority of code is unique (excluding a couple of nested loops from the labs). The code was developed over multiple files to provide clarity.

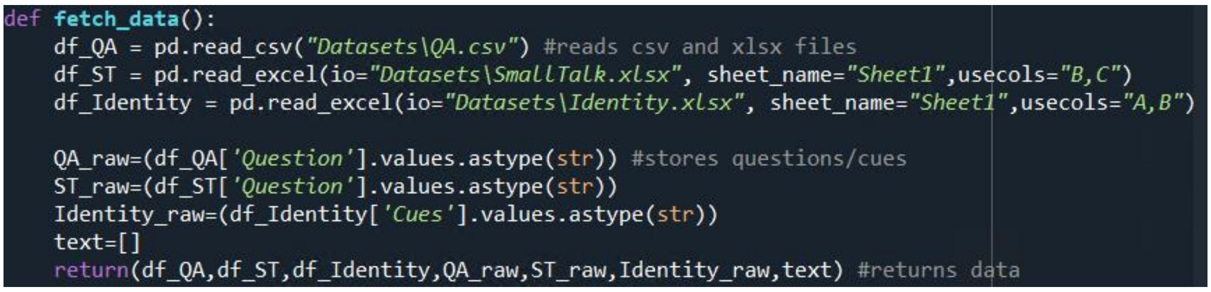
Implementation:

The epicentre of the system is similarity matching using the cosine similarity function. This can be used to determine the intent of the user (identity, small talk or question answering). The similarity is calculated by comparing the user's normalised input to each question/cue in the database. All these similarity calculations are ranked from highest to lowest, the top three are selected (on the condition there are three sentences above a threshold) so if the highest rank doesn't answer a user's question, the system can output the next answer and the one after that. The similarity calculations are formulated from the module scripy:



```
1 - spatial.distance.cosine(input_weighted, data_weighted[i])
```

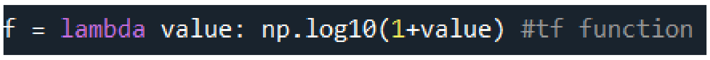
To access our data we used the module pandas, data frames are stored and the questions and cues are stored. The removal of stopwords is implemented with help from the nltk module (for question answering only, we don't want this for small talk and identity management) and the addition of lemmatisation to the data and the user's input. Lemmatisation was preferred over stemming due to it being more consistent. I felt like a small increase in time was worth the payoff for accuracy providing the time isn't too high.



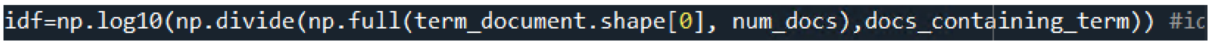
```
def fetch_data():
    df_QA = pd.read_csv("Datasets\QA.csv") #reads csv and xlsx files
    df_ST = pd.read_excel(io="Datasets\SmallTalk.xlsx", sheet_name="Sheet1",usecols="B,C")
    df_Identity = pd.read_excel(io="Datasets\Identity.xlsx", sheet_name="Sheet1",usecols="A,B")

    QA_raw=(df_QA['Question'].values.astype(str)) #stores questions/cues
    ST_raw=(df_ST['Question'].values.astype(str))
    Identity_raw=(df_Identity['Cues'].values.astype(str))
    text=[]
    return(df_QA,df_ST,df_Identity,QA_raw,ST_raw,Identity_raw,text) #returns data
```

To improve the accuracy of our similarity, tf-idf weighting is implement prior, on the data and user input. I used lambda to create a function for this including np.log10 which uses the module numpy. Numpy is also used to calculate the idf by generating a matrix with each element containing the quantity of documents and dividing it by the quantity of each term found in a document.



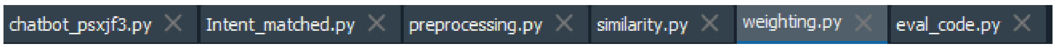
```
f = lambda value: np.log10(1+value) #tf function
```



```
idf=np.log10(np.divide(np.full(term_document.shape[0], num_docs),docs_containing_term)) #ic
```

[4 4 4 4] / [1, 0 ,2, 1]

We have, functions, comments and representative variable names for good coding practise, making it more readable and efficient. The same goes for multiple files, we have created a central file which accesses pre-processing, similarity, weighting and an intent matched file.

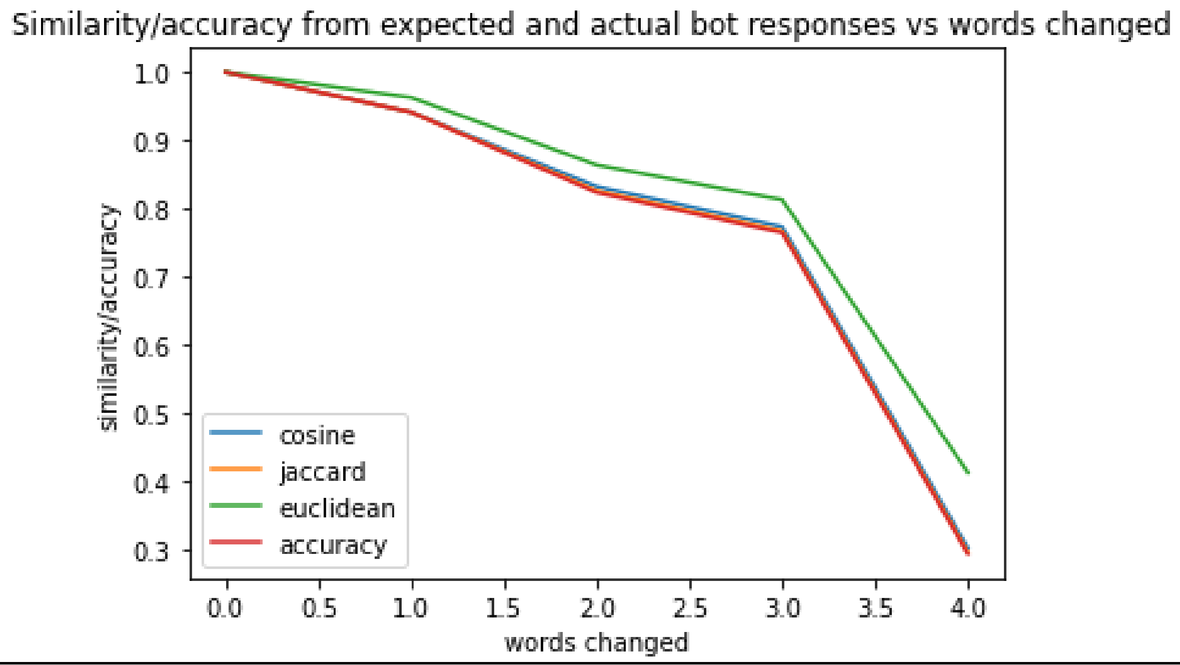


Evaluation:

I tested my chatbot by inputting questions and analysing it's answers and using similarity measures and accuracy (correct predictions/all predictions). The similarity measurements I used were cosine, Jaccard and Euclidean distance. Each provide similar reasonable results (see discussion section).

Testing methodically, I recorded these measurements for the same question with different amounts of difference to the data it was trained on. Zero, one, two, three and four words changed to a sentence were applied, there is no use having the exact same new input as the trained data since a user is unlikely to word the question the exact same way.

I then plotted each of these similarities/accuracies against the words changed using the module matplotlib.



I created confusion matrices for each quantity of the word changed to allow me to calculate the f1-score, precision and recall and I have plot that too. I found the results has a resemblance to the graph above too.

The confusion matrices contain two classes, the correct response and the bot shouldn't know class. The don't know output is when the bot communicates it's unsure, whilst the "correct" output is when the bot thinks it's correct (although it may not be).

Zero words changed:

	Shouldn't know	**Correct response**
Don't know output	**0**	**0**
"Correct" output	**0**	**17**

F1-Score:1

Precision: 1

Recall: 1

One word changed:

	Shouldn't know	**Correct response**
Don't know output	**0**	**0**
"Correct" response	**1**	**16**

F1-Score: 97%

Precision: 94.1%

Recall: 1

Two words changed:

	Shouldn't know	**Correct response**
Don't know output	**1**	**0**
"Correct" response	**2**	**14**

F1-Score: 93.3%

Precision: 87.5%

Recall: 1

Three words changed:

	Shouldn't know	**Correct response**
Don't know output	**2**	**0**
"Correct" response	**2**	**13**

F1-Score: 92.9%

Precision: 86.6%

Recall: 1

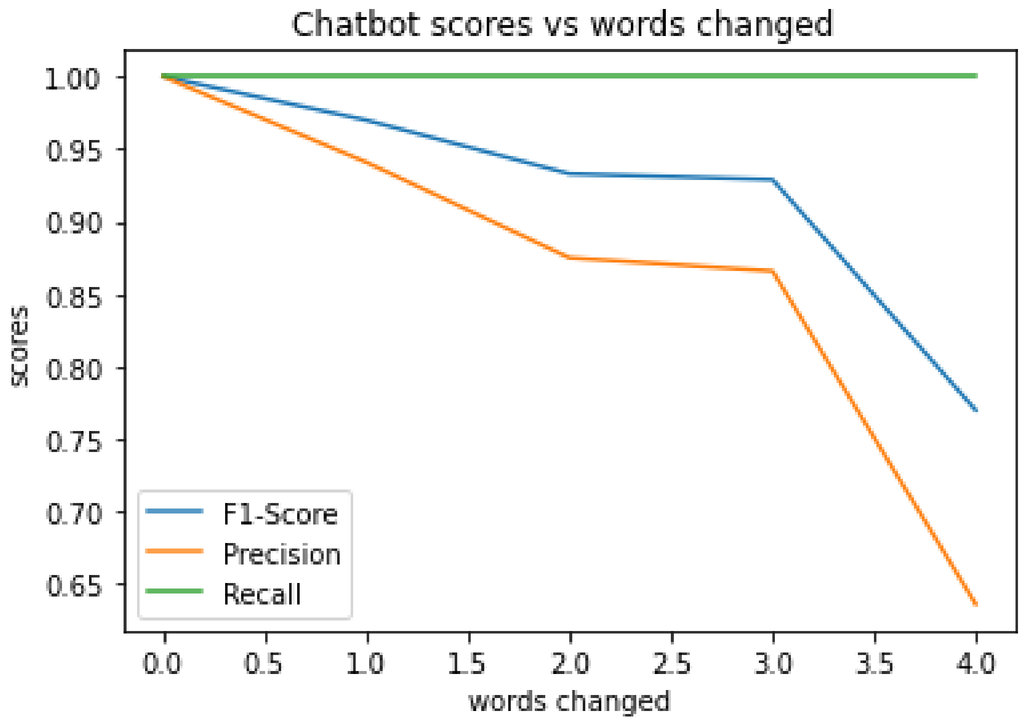
Four words changed:

	Shouldn't know	**Correct response**
Don't know output	**6**	**0**
"Correct" response	**4**	**7**

F1-Score: 77.7%

Precision: 63.63%

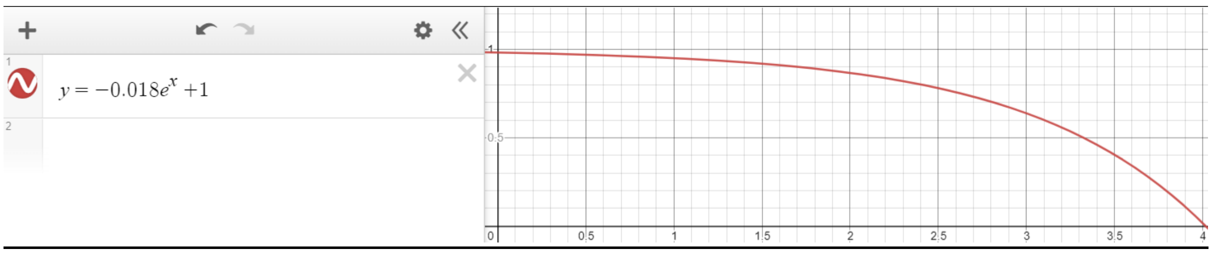
Recall: 1



Finally, I inputted values that should respond with a "I don't know message". 4/5 times this was successful as seen 110-114 in the evaluation data spreadsheet. Once with the word "six" it responded with an answer from the QA database.

Discussion:

As you can see our similarity graph provides similar results regardless of measurement (Euclidean only having a slightly higher similarity measurement). The chatbot is perfect for perfect user results, it decreases exponentially (fitted similar to a y=-0.018e^x + 1 graph).



This is because after a few changes there's fewer keywords. Keywords are important due to weighting; a simple word being changed won't affect much but if a unique word is changed it become very difficult to predict the answer because the similarity score becomes too low. I also found that there was a greater struggle with shorter sentences because there become few words which are similar. The average similarity for three words changed is 77% which is a respectable score, but a change to four words changed takes the similarity down to about 30% and the accuracy to 40% which isn't good enough. I didn't ask the chatbot completely random questions since we wouldn't learn much, all the questions the bot wouldn't understand anything as you can predict which the estimated graph above.

Precision and f1-score follow a similar shape to the similarity scores. Precision predicts the response percentage success when it believes it's correct. My recalls are always 1 since the amount returned as the response don't know isn't ever the desired response. However, if we use our 4/5 results where we wanted the result to be incorrect ("I'm not sure message"), we can say the recall is more like 0.8.

My chatbot approximately evenly says don't know and correct when it was in fact incorrect. However, the most common response was true positives in that it predicted a correct answer and it was the correct answer.

TF-IDF, the normalization of data and cosine similarity is likely why the results are good, however I'm sure there are better algorithms and techniques to be explored.

One potential improvement we can explore is the use of machine learning and classification such as K-Nearest Neighbour classifiers, neural networks, logistic classification and naïve bayes algorithm. My small talk function works however the conversations are short, creating a database with more possible inputs and responses would allow for better conversations. Another improvement should be reduction in racial bias which I will discuss in further detail next.

Racial bias may exist in our system depending on our data. For example, if the user inputs a word like "arrest" and our algorithm responds with data on crime statistics regarding a particular race due to the similarity being high without even mentioning a race, this will appear as racist. To avoid this, we need to ensure our accuracy is higher for unseen sentences, as well as ensuring our data itself isn't racist. This doesn't mean race should be excluded from our data however, there are solutions such as having diversity in who builds the database and having debiasing algorithms. Crowdsources, if used, should be treated fairly but we should also ensure their incentives are good and they have been properly trained. Being transparent and open regarding bias in our system is key too since they show these limitations are here and not hidden so they won't cause any inflict unconscious biases. This can be done through data statements.

Conclusion:

My chatbot has the core components in normalisation, weighting, and similarity matching which are used to accurately respond to a user's input providing it doesn't stretch too far away from the data in the database. Question answering, intent matching and identity matching are all implemented to a strong level. Small talk could have a wider variety of inputs and responses.

My system contains original features, the continue function (if the user wishes to learn about the answer in more detail) and the does this answer your question function which can go to the ranked documents and select the next response if the previous one doesn't suit it. The continuous improvement of the database through this feature (where if it's a correct response but it's a slightly different input, it saves the new input and answer to improve similarity readings and accuracy in the future).

References:

[1] Dwitam, F. and Rusli, A., 2020. User stories collection via interactive chatbot to support requirements gathering. Telkomnika, 18(2), pp.890-898.

[2] Domeniconi, Giacomo & Moro, Gianluca & Pasolini, Roberto & Sartori, Claudio. (2015). A Study on Term Weighting for Text Categorization: A Novel Supervised Variant of tf.idf. 10.5220/0005511900260037.

[3] Su, M.H., Wu, C.H., Huang, K.Y., Hong, Q.B. and Wang, H.M., 2017, December. A chatbot using LSTM-based multi-layer embedding for elderly care. In 2017 International Conference on Orange Technologies (ICOT) (pp. 70-74). IEEE.

[4] Qaffas, A.A., 2019. Improvement of Chatbots semantics using wit. ai and word sequence kernel: Education Chatbot as a case study. International Journal of Modern Education and Computer Science, 11(3), p.16.

[5] Setyawan, M.Y.H., Awangga, R.M. and Efendi, S.R., 2018, October. Comparison of multinomial naive bayes algorithm and logistic regression for intent classification in chatbot. In 2018 International Conference on Applied Engineering (ICAE) (pp. 1-5). IEEE.