Slide1

Hello, this is my presentation on my work on the python chatbot produced for the data mining and foundations of AI. Il get right into it.

Slide2

First step as always was to analyse the problem and identify the possible avenues this task can be approached.

Slide3

In the industry, AI controlled chatbots usually fall under two types. The first of which, generative, is where the model creates the responses from scratch, using its own training data to create personalised responses. The second method, retrieval, is where a model categorises an input and returns a predefined response to the input.

As the objective of this task is to make a customer service chatbot, the generative model was clearly less suited – the model needs to give clear and valid advice to a user, and generated data, while more personalised, has the risk of giving invalid or even malicious advice. Hence, for this task, I chose to use the retrieval model.

Slide4

One of the hardest parts of this task was determining what library would suit the system best. Initially, I attempted to work with libraries like chatterbot, but after finding them unsuited to the classification-based structure of the task, I ended up using a library incorporated within chatterbot – Spacy.

Spacy is a natural language processing library – that is, it is designed more as a framework to build chat bots from than a chatbot itself. Spacy provides one of the most important features for the project – the ability to split inputs into tokens and doc files.

Slide5

The primary function of spacy is to produce the aforementioned tokens and doc files. The library is able to split sentences into their component tokens and ascribe various characteristics to each token, most notably its type (such as noun, verb etc.). Once a stream of text is split into tokens, these tokens are then stored in a doc file – a collection of tokens with all their attached characteristics, stored in a vector format.

Slide6

The vectorisation of these tokens and doc files is what allows a model to learn. Spacy incorporates a function called “Similarity” – in which it uses a cosine function to produce a value representing how similar two inputs are. The exact formula and functionality for this can be seen in the attached diagram but the important thing to note is that this means, by providing a model more and more doc files relating to a topic we can teach the model to recognise similar sentences.

Slide7

Thus, here is a simple approach to designing a retrieval model using spacy similarity. By creating multiple models, each with their own topic, and then providing it data relating to said topic, we can train a model to recognise similar sentences. By creating an array of these models, each trained to recognise their own topic, we can then call on each model to report its similarity to an input and choose the highest similarity model as the expected topic.

Slide8

Its at this point I should note the approach I took to this assignment – Instead of the more traditional approach of conducting data analysis followed by building a model, I took these steps in reverse. This was because at this stage of development I was still unsure what approach to take, and therefore wanted to find a good option before I began to collate a dataset to use.

Slide9

My first step in developing the model was to ensure what I had theorised would work successfully, so I trained a simple model to recognise questions about purchasing cars, maintenance on vehicles and purchasing bikes, and gave 10 questions for each to train the three models on. This for the most part proved the approach was possible – an input could be categorised with some success into the correct topic. However, due to the small amount of training data, the system proved volatile with its accuracy, ranging between 20% and 100%.

I also just want to note on this slide that the images may appeared doctored, and that’s because while taking screenshots I accidentally had the software to report similarity in decimal form, so I just quickly removed the decimal point from the outputs for these images.

Slide10

Deciding to go further down this route, I next attempted to acquire a more suitable dataset to ensure the model could work with more topics and more training data. Using a dataset consisting of twitter questions, I trimmed the set down to include 1200 questions directed at apple support, and manually categorised them into eight categories.

Slide11

After training them, I found the success of the model dropped dramatically – averaging only 30%. From research I found the culprit – the language training of the model. One of the biggest features of Spacy is that it can recognise words that have similar meaning – connecting “old” and “elderly”. To do this, spacy provides two pre-trained language files, a small one and a large one. Up until this point I had been using the small language file, which worked fine with the manually created 30 question test, but for a more varied set, the limited small dataset proved incapable. After instead switching the model to use the large dataset, the results were immediately noticeable – the accuracy almost doubled. Hence, from this point onwards all models used the large language file.

Slide12

This is when I discovered a potential problem – one that might have made this entire solution invalid. The hypothesis was simple, does increasing the amount of training data provided lower the accuracy of the model? This is because, when spacy produces a similarity score, it takes the average similarity of all of its doc file, so excessively large training data means that each item in the set has a lesser impact.

For example, if we pretend that spacy only matches words to produce its similarity score (which is not how it works but it makes a good example), if we train it with the sentence “How can I buy a phone?” and then give it the exact same data as an input, the similarity would be 100% - an exact match. If we then give it another item – “where are phones available for purchase”, the match is only 16%, and so the average falls to 58%, despite being an exact match to an item in the dataset.

Slide13

I decided before proceeding to test this theory. I trained a model on 20% through 70% of a training data set in steps of 10%, recording the accuracy score provided. To make this test fair, I ensured the test data remained constant – giving each model 100 items of test data regardless of training data size. After completing 30 iterations of this test, I compiled the results and examined the product – the model retained a steady rise in accuracy as more training data was produced, meaning that the hypothesis had been wrong.

Slide14

This allowed me to move to the next stage of the model. Noticing that the model was fairly simple as is, only taking an input and matching it to a topic, and having no concept of conversation, I decided to improve the model by giving it more personality. The SpacyTextBlob library is a library that adds a component to the spacy pipeline – training it to recognise two aspects, polarity and objectivity. In this context, polarity refers to the sentiment of the input, how positive or negative the input is, whereas objectivity refers to how specialised a question is – is the input a general question or a specific request?

Slide15

Once incorporated onto the pipeline, I used these stats to construct a sentence building function. Simply, based on the polarity of an input, a different prefix for the input would be chosen – such as “Im sorry to hear youre having problems” if the input is toned negatively. Then, based on how objective a query was, a different response to the topic would be chosen – if a user asked an objective question, theyd be pointed to the FAQ, whereas if they asked a subjective question they would receive the opportunity to speak to a human operator to ask their particular question.

Slide16

With the model reaching a stage I was happy with, I then turned my focus to the data input.

Slide17

The biggest problem with the system thusfar has been the datasets – which have either been manually created or manually given a topic. Either way, both approaches meant that the training data was both small and biased towards my own thoughts. To properly create the model, I needed a dataset that had the two key components needed by the system – a topic and the text relating to said topic.

Slide18

The solution to this ended up being within stack exchange. Stack exchange is a variety of websites that allow users to submit a question or answer other people’s questions, but most notably, it requires a user to submit atleast three “tags”, dictating the topic of their question. If I could acquire this data, I’d have access to thousands of data points – each with their own topic and text.

Thankfully, stack exchange provides an API to allow users to undertake tasks like mine. By using the stack exchange API, you can download pages of questions in JSON format – after which it can easily be converted into a CSV format.

Slide19

There are a lot of stack exchange sites, but as a customer service chatbot, I needed the items to be business specific – hence I chose to use the finance and money stack exchange website. Using the API with inputs like that shown at the bottom of this slide, I was able to pull about 500 questions for seven topics, and then easily convert them into a CSV file.

Slide20

After this, I quickly removed all the unnecessary fields – minimizing the set down to only include the topic and the question. Plugging this newly formatted data into the model rose accuracy to average roughly 70% accuracy – improving on the previous 65% average of the old dataset. This score was then improved further by pre-processing the data – removing all the non-ascii characters and punctuation, as well as removing all the “stop words” – words like “the” or “and” which, by being common, may bias similarity away from more important keywords. After doing this, accuracy rose by an average of 3%.

Slide21

Now that the model had been constructed and a suitable dataset found and formatted, all that was left to do was examine the findings.

Slide22

When tested with the traditional 70/30 split on one hundred iterations, the model produced a 72.5% average accuracy rate. In manual tests, the model also performed fairly successfully – more often than not categorising valid inputs into the correct topic and providing the appropriate response.

However, a significant issue arose in the input of invalid questions. While the model is set to automatically provide an unknown-topic response when receiving an accuracy score of less than 50% - matches still occurred with completely irrelevant data. As can be seen in the examples, the model struggles to differentiate valid questions and nonsense due to its similarity approach.

One other thing I’d like to note about the accuracy of the system is that the model provides an option to reassess a question if the similarity score is less than 90% - using the second highest similarity topic as its prediction. As shown in the diagram, this second guess has a far lower accuracy score – only averaging around 32% - however, this does mean that the total chance of a user getting the correct response for their topic within the first or second guess is approximately 81%