

Developing an Intelligent Chatbot: the First Interim Report

Group UG14: Daniel Willis, Charlotte Anderson and Brandon Gous

January 29, 2022

Abstract

The report shall introduce some background to the subject as well as the groups motivations for creating the chatbot. It then discusses the group's aims, objectives and difficulties for both part 1 and 2 of the brief. In section 2, similar systems are discussed and evaluated by their relevance to the project. In part 3 methods, tools and frameworks used for this project are stated and explained. The design in section 4 and implementation in section 5 are then outlined in subsections of the different areas of work. Testing for this project is then covered briefly in section 6. Finally, the groups work is evaluated and concluded in section 7.

1 Introduction

For this coursework (Wang 2018) the group is developing an intelligent chatbot for customers to use which will find them the cheapest train ticket for a specified journey. The service shall also provide users with information on train delays, for a specified journey, to improve customer service. The chatbot will be able to recognise which service the user wishes to use, using Natural Language Processing (NLP) techniques. The chatbot will respond appropriately until it has gathered all information needed to answer the user's query.

1.1 Background and Motivation

Increasingly chatbots are used for customer service by companies and businesses. The first wave of chatbot systems could only recognise questions posed to them if they were written exactly as it was programmed to understand EBI (2022). Artificial intelligence is a hot topic at the moment, and has been used to develop seriously intelligent chatbots. Using AI, chatbots are able to understand and answer customer questions, which can be asked in a huge variation of ways. As chatbots these days are so capable, many companies favour using AI for customer service over an employee, as it can save money and time. Furthermore SINTEF et al. (2019) found in his study that many users do understand that chatbots work best when asked direct and precise queries; therefore often the user is doing some of the work for the chatbot by using simple language.

1.2 Aim and Objectives of this coursework

The aim of the group is to develop a prototype intelligent chatbot system that can be extended to handle a large variation of queries. The chatbot will have two functions:

1. Finding a user the cheapest ticket for a specified journey.
2. Reporting to a user the predicted train delay when prompted with their journey and current delay.

Our objectives are:

1. To keep to our outlined project plan in the Gantt Chart.

2. Have two weekly meetings on Microsoft Team to update the group on progress made.
3. Have all set separate work completed for the chatbot, ready to be integrated with each other, by the 10th of January. This will give the group a week to integrate their systems together and iron out any issues, ready for the demonstration and video.

1.2.1 MoSCoW Analysis

Must:

- The chatbot must be able to handle single-way tickets.
- The chatbot must prompt the user until enough information is obtained to complete their query.
- The system must be able to use the gathered information to complete the request.
 - In the case of a cheapest ticket query, it must return the cheapest ticket found and a hyperlink to a page where the customer may purchase it.
 - In the case of prediction, it must return the predicted train delay in minutes at the users destination station.
- Once the chatbot has received all the information it needs to search, it must check with the user it has understood the query correctly, so changes can be made if needed.

Should:

- The chatbot should be able to handle return tickets.
- The chatbot should be able to handle a large query with multiple pieces of information within it.
- The chatbot should be able to handle misspelt station names.
- That chatbot should return the user an answer to their question, once all the information is obtained, within 15 seconds.

Could:

- The chatbot could be able to handle multiple date and time formats.
- The chatbot could be able to ask the user if there is anything more they can help with.
- The chatbot could have an attractive user interface.
- The chatbot could search for multiple adults or children.

Won't

- The chatbot won't be able to handle all query formats.
- The chatbot won't be able to recognise badly spelt station names.
- The chatbot won't be able to use railcards in its search for the cheapest tickets.

1.3 Difficulties and Risks

A risk with this task could be poor project management. To try and reduce this risk, in our first group meeting we decided upon a group leader, Dan. Dan ensured that we had weekly meetings in which he encouraged us to all share what we'd been working on. A large portion of this work had to be completed over the Christmas break. It was important that the team continued to work together despite not being on campus at university. Our group continued to have regular meetings on teams during this time, with the intention that this would reduce the risk of slippage in the project.

Another risk is that deadlines will not be met. A member of the group not meeting a specific deadline could mean that the next piece of work cannot be started. To reduce this risk, the work was divided in such a way that each member could get on with a specific part of the brief. The work was delegated as follows:

Dan:

- Project manager
- Create the database
- Research predictive models
- Programme a few of these models and report back the most successful model
- Work with Brandon to get this work integrated with the chatbot.

Charlotte:

- Write the webscraper for the cheapest ticket
- Decided on the flow of conversation for the chatbot
- Write the NLPU component and work with Brandon to get this integrated with the chatbot.

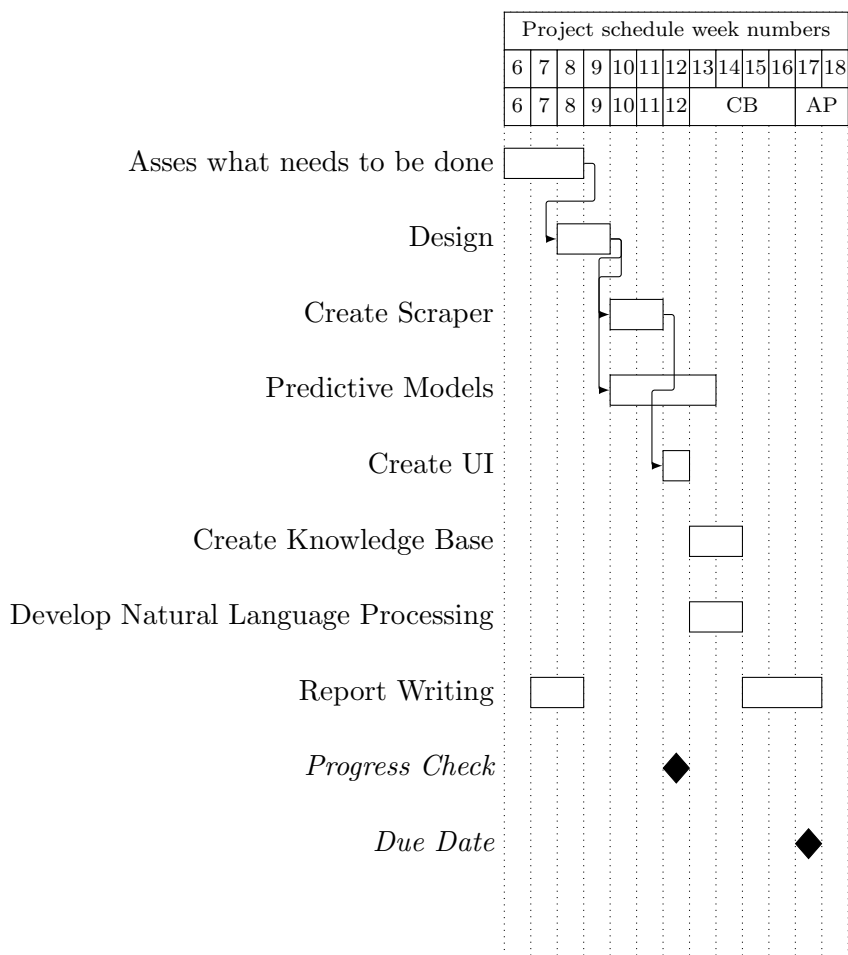
Brandon:

- Write the Knowledge-Base
- Reasoning Engine and User Interface.
- Worked with the other team members to get their contributions integrated with the chatbot.

Working this way meant that each member could begin programming without having to wait for another member of the group to have already finished a task. In the weekly meetings everyone shared what they had been working on since the last.

Once the individual work was complete the final risk was that our work would not be integrated together in time. This is a large and integral task, if all elements work separately but cannot work together the brief has not been met; for this reason one of the objectives was to begin integrating work by at least the 10th of January.

1.4 Work Plan



Project Gantt chart

2 Related Work

2.1 TFL chatbot

The TFL chatbot was very friendly and said “You’re welcome” (or equivalent) when you thanked it TFL (2020). It seemed to understand what was meant most of the time very well even when the input was not very specific. It was functionally useful as it could give live information about the next few buses if it knew the stop code for the bus stop that the user was standing by. It was fully integrated with facebook messenger so it is as easy as sending a facebook message.

Unfortunately it had very limited functionality especially for the tubes. Often it failed to select the correct station or it took a few attempts to get the correct station. For where there are two stations named similarly it did not offer suggestions of stations to the user to pick from. Planning was impossible as there was no way to enter a time, it only handled current time rather than a time in the future. Additionally, it couldn’t understand a user if all information was entered in one message, it had to separate the too and from stations.

Overall the chatbot worked well, but needed more functionality as it would only occasionally misunderstood. However, for tubes it could only tell the user what line the closures are on so, and the user has to decide if they want to see in more detail when/where they are. Additionally, a lot of the time when trying to extract information, especially when talking about cost, the chatbot would just send a link to the TFL website.

2.2 BlenderBot

Blenderbot or parlai is a open source, highly customisable chatbot developed by Facebook Stephen Roller (2020). Its relatively easy to install and use in the terminal, and if desired it comes with integration into Facebook messenger.

One can be assigned a random persona, set one manually, or just have a general conversation. This was very interesting to play around with, giving the chatbot different characters to play with and seeing how well it handled them.

Models can be uploaded from the “zoo”, or one can be created built off a base. The 80 million data points model is a little clunky and not as fluent. There is a definite pattern to its responses. The top level 9 billion data point weights is very impressive, and could almost pass as a human.

It’s very factual too, it will attempt to slip in facts in a “normal” manner rather than just blurting them out robotically.

It comes with a lot of documentation and a paper about how to build a good model.

Overall it was entertaining and enjoyable to speak to. If there was more time for this project, this is a great feature the group would have liked to incorporate it into the project.

2.3 Google Assistant

Google assistant is arguably a chatbot, it commonly has a speech to text, and text to speech software alongside Google (2016). However there is the option on phones to just type text to it and reply, but this conversation would be the same if verbal.

Google assistant is useful to perform tasks such as switching lights on and off, playing media, setting timers for cooking etc. But that is all it is, as it is designed to be a tool and to be as useful and functional as possible. It isn’t designed to hold a conversation with a user.

This is where the assistant is lacking; its responses are very robotic, blunt and to the point. No human speaks like that. Any questions asked will have an answer from crawling the web or doing the maths. Users can ask it to tell them a joke, ask about the weather, or how the traffic looks on your morning commute, but it cannot hold a conversation; it is being used as a tool to solve a users problem.

Its natural language processing is near perfect and in real time from audio which is incredible. It really understands what you are saying so long as it is a command. Overall, a very useful product to have around, but not the best chatbot.

3 Methods, Tools and Frameworks

3.1 Methods

A primarily text based user interface was selected, along with an aim to wrap the text-based system in a graphical user interface. The user's responses would be processed using a natural language processing library called Spacy. From the responses provided by the user, the knowledge engine would determine what steps are next based on rules built into the system and which of those match the user's responses.

For the delay prediction we used a K-Nearest-Neighbour (KNN) method and a neural network.

3.2 Languages, Packages, Tools

The two main languages that were selected were Python and C# for the main chat-bot and then some of the database interaction respectively. Within the chatbot the following packages were used:

- Experta - Used for the knowledge engine, it handled all of the questions relating to the state of the booking/delay prediction query.
- Fuzzywuzzy - Used to process station names that may have been entered incorrectly. Such as in the case of typos.
- Datetime - Was used for reading and formatting dates.
- Spacy - Spacy was used for all of the natural language processing.
- Json - JSON was the format that the text stemming was stored in, so the library was used to read and understand that.
- Pandas - Pandas is a data analysis library that we used to help round some timing to better work with the train data.
- Selenium - Was a web-scraping library that we used to extract data from websites to feed back to the user
- Numpy - Was used for most of the arrays in the delay prediction, and the matrix transformations required.
- Matplot - Was used for the graph plotting so that we could visualise how the two delay prediction methods were working and compare them.

For the database, we selected SQL Express. Because a member of our group had used this a lot on their placement year so was familiar with MYSQL.

4 Design of the Chatbot

4.1 The Architecture of the chatbot

In an early group meeting we decided to conduct the work for this project modularly. This meant creating contracts with one another on what work would be completed by each member, by when and detailing exactly what we expected from each piece of work. This meant we could work separately, which was necessary as much of this work was completed away from university. It was decided that Dan would create a database, as well as writing predictive models, Brandon would write the knowledge-base and inferring engine, and Charlotte would write the scraper and Natural Language Processing Unit (NLPU). Once all this work was completed, the group could come together and integrate the various pieces of work.

The chatbot is responsible for handling questions from a user and responding to these. It uses the NLPU to understand what a user has inputted, as well as to see if all fields have been filled for a search. If all fields have not been inputted, the chatbot will reply to the user asking for the information it is missing. Once the NLPU has all the information it needs to make a search, it delivers this to the scraper, or the prediction model depending on the task. If the task is for find the cheapest ticket, once this has been found the chatbot will relay this information to the user with a price and a link to buy the ticket. Similarly, for predicting train delays, once the predicted delay has been predicted, it is relayed through the chatbot to the user.

4.2 User Interface

For the user interface, we decided to emulate a chat window similar to your typical messaging app, consisting of a chat log, chat line and send button. The chat log would display a history of all the messages sent between the user and the chatbot, whilst displaying who sent each message and the time sent. The chat line and send button would be the way the user interacts with the system. Responses to messages sent by the chatbot would be written in the chat line and then “sent” for processing when the return key, or the send button are pressed.

4.3 Scraper

The scraper will input all necessary fields in Trainline’s website. Trainline’s ticket search will fail if the train station is spelt wrong so the scraper needs to work with the NLPU to ensure stations are correctly spelt. Additionally, the website only handles specific date formats so the NLPU will need to format differing date formats into the accepted version.

4.4 NLP

The NLPU shall be able to handle large queries. This means a user can input into the chatbot a sentence with lots of information in, and the NLPU shall be able to understand what the user is asking for. This may not be achievable for all structures of sentences as that would be very complicated, but annex A shows the structure of a sentence the NLP will be designed to work for. The NLPU shall also accept multiple date formats, time formats and understand words like ‘tomorrow’ and ‘today’.

4.5 Knowledgebase and Inferring Engine

The knowledge base will handle getting all of the information it requires from the user through the use of the inferring engine. The knowledge base will get information from the user in the form of questions, and the inferring engine will use a set of conditions to determine which question needs to be asked and when. The set of conditions will evaluate which information has been obtained from the user, the validity of the information, and what information is missing.

4.6 Delay Prediction Models

Three prediction models are planned to be tested. A Bayesian model that calculates the probability of the train being late to the second station, given that the train is delayed to the first station. A neural network that uses the delay and the normal time a train takes between the stations to predict how late the train will be. And KNN which predicts how late the train will be to the second station using similar data.

4.6.1 Bayesian

Bayesian Theorem is that the probability of event A happened given that B happened can be calculated by the probability of event B happening given that event A happened multiplied by the probability of event A happening all divided by the probability of event B happening.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B)}$$

Additionally this can be expanded so that the probability of event B happening doesn't need to be known. The probability of event B happening is equal to the probability of event B happening, given event A happened multiplied by the probability of event A occurring, plus the probability of event B happening given that event A does not, multiplied by the probability that event A does not happen.

$$P(B) = P(B|A) * P(A) + P(B|A') * P(A')$$

This means that the probability of event A happening, given that B happened, can be calculated from the probability of event A happening, the probability of event B happening given event A happened, and the probability of event B happening given that event A didn't happen.

$$P(A|B) = \frac{P(B|A) * P(A)}{P(B|A) * P(A) + P(B|A') * (1 - P(A))}$$

This is applied to the prediction of trains being late by letting event A be the train being late to the second station, and the event B being the the train being delayed at the first station.

4.6.2 Neural Network

A neural network is a collection of layers of neurons, each neuron holds a value known as activation of that neuron. There can be an unlimited number of layers and neurons and they can be used to solve very complex problems such as object recognition.

Each neuron is connected to all the neurons of the previous layer through numbers called weights (W) and a bias (b). The weights are organized in the form of a matrix of the shape number of units in current layer by number of units in previous layer. This means that to calculate a neuron on the next layer, all the previous values are needed.

If this is applied to the project there would be, two input nodes; one for the delay at the first station, and one for the normal time taken between the stations. And there is one output node for the prediction time that the train is late. These two layers cannot be changed change, but any of the hidden layers can be altered to still produce a neural network to accomplish the same goal. This is shown in figure 1 it shows all the neurons in the simple neural network that has been designed to solve the problem with two hidden layers of three neurons each.

For the maths behind the forward and backward propagation, W_{kij} refers to the weight of the connection from i'th neuron in k'th layer to j'th neuron in k-1 layer. The biases are organized in the shape of (no. of units in current layer, 1) so B_{ki} corresponds to the bias of the i'th neuron in k'th layer.

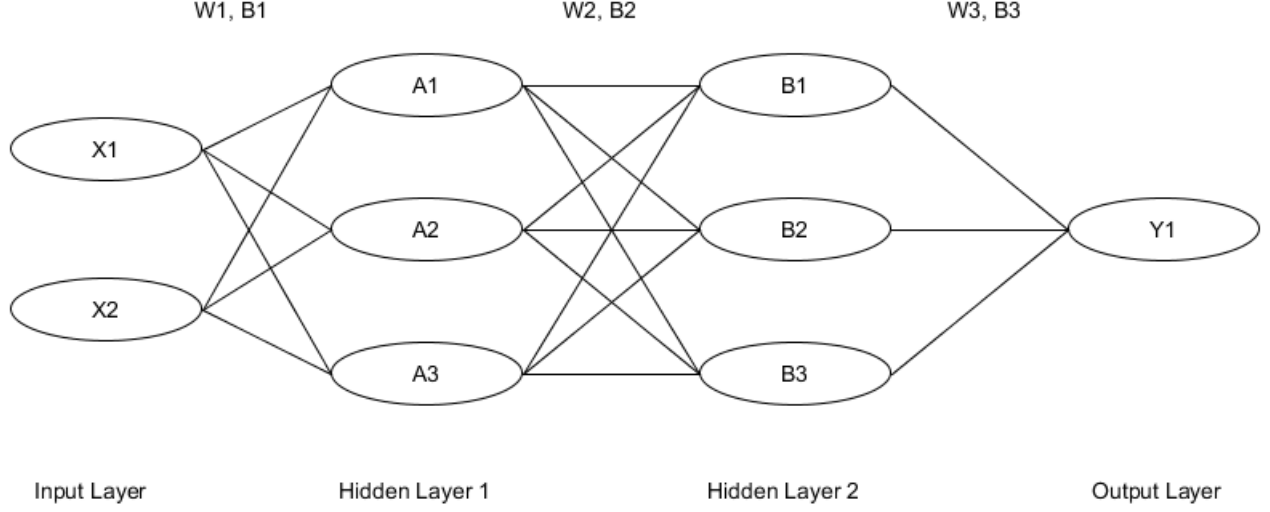


Figure 1: Diagram showing the neural network

Each neurons value can be calculated by:

$$A1 = g(f(X1, X2)) = g(W1_{11} * X1 + W1_{12} * X2 + b1_1)$$

$$A2 = g(f(X1, X2)) = g(W1_{21} * X1 + W1_{22} * X2 + b1_2)$$

$$A3 = g(f(X1, X2)) = g(W1_{31} * X1 + W1_{32} * X2 + b1_3)$$

$$B1 = g(f(A1, A2, A3)) = g(W2_{11} * A1 + W2_{12} * A2 + W2_{13} * A3 + b2_1)$$

$$B2 = g(f(A1, A2, A3)) = g(W2_{21} * A1 + W2_{22} * A2 + W2_{23} * A3 + b2_2)$$

$$B3 = g(f(A1, A2, A3)) = g(W2_{31} * A1 + W2_{32} * A2 + W2_{33} * A3 + b2_3)$$

$$Y = f(B1, B2, B3) = W3_{11} * B1 + W3_{12} * B2 + W3_{13} * B3 + b3_1$$

Where $g(x)$ is the activation function. Being as this is a regression problem and not classification $g(x) = \max(0, x)$ is used, and is more commonly called the ReLU. This returns 0 when x is negative and x when positive. Due to it's low saturation region, it is highly trainable.

The equations above can be vectorised using the dot product of the matrices:

$$A = g(W1.X + b1)$$

$$B = g(W2.A + b2)$$

$$Y = W3.B + b3$$

However for the neural network to become more accurate it has to learn by adjusting the weights an bias to bring the output closer to the target. This is done using a back propagation algorithm to calculate the gradients. These are the vectorized equations:

$$dY = \frac{1}{m} * (Y - T)$$

$$dW3 = \frac{1}{m} * (dY.W3^T)$$

$$db3 = \frac{1}{m} * \sum dY$$

$$dB = W3^T.dY$$

$$dW2 = \frac{1}{m} * (dB.W2^T)$$

$$db2 = \frac{1}{m} * \sum dB$$

$$dA = W2^T.dB * g'(W2.A + b2)$$

$$dW1 = \frac{1}{m} * (dA.W1^T)$$

$$db1 = \frac{1}{m} * \sum dA$$

Where T is the true value, m is the number of datapoints, $*$ is multiplication, $x.y$ is dot product of the matrices x and y and x^T is the transpose of matrix x . We do not need gradients with respect to the input layer neurons because they are not going to be changed.

Now using these gradients calculated, weights can be modified, as well as bias, which gets a result closer to what the true result would be.

$$W1 = W1 - lr * dW1$$

$$b1 = b1 - lr * db1$$

$$W2 = W2 - lr * dW2$$

$$b2 = b2 - lr * db2$$

$$W3 = W3 - lr * dW3$$

$$b3 = b3 - lr * db3$$

Where lr is the learning rate which is a configurable hyperparameter that is in the range between 0.0 and 1.0. The learning rate controls how quickly the model adapts to the problem.

4.6.3 K Nearest Neighbors

For the K nearest neighbour model lots of data has been extracted from the database for each query. A stored procedure has been used for this.

The amount the train was late to the first station in the historical data, is then compared to the amount of delay that the user had at the first station using a distance function; in this case euclidean distance is used.

These distances are then sorted and you get the mean of the delay at the second station for the closest K nodes in the dataset. This is simple algorithm, but heavily dependant on finding the correct K value. Too small, and the correct number will not be found it will be skewed as there isn't enough data, or it is too large and the result will be skewed by nodes that are not close enough.

5 Implementation

5.1 Scraper

The implementation of the scraper has been designed to complete the task of finding the cheapest ticket for a train booking. Trainline’s website *Search, Compare and Buy Cheap Train Tickets* (2022) was chosen to be scraped, this was because when inspecting the page when a search was completed there was a clear tag which indicated the cheapest ticket which could then be used to return to the user. The package ‘selenium’ (2022) was used to webscrape as it was found to be the most useful for what was trying to be achieved. The TrainLine class has many functions which find different elements within TrainLine’s page, which are then filled once the chatbot has retrieved information from the user.

The scraper clears all field boxes before entering data. This is because if a field was pre-filled by the website, i.e. defaulting to today’s date, when the scraper tried to enter the date which has been specified by the user, it cannot do so as it is already full and throws an error. The scraper uses selenium to find specific elements within the page that will be filled with the users query. For example if a user wants a single ticket, it is capable of choosing between the radio buttons for a ‘single’ or ‘return’ journey. This is done by using TrainLine’s element ID’s, for example the radio button for a single tickets element ID is ‘single’, and therefore can be found and clicked.

Once a full search has been made, the scraper waits for an element with an ‘aria-label’ that has the value ‘the cheapest fare’. The price which belongs to this value is the cheapest ticket for the specified journey, and this price and the link to the page is given as a reply to the user through the chatbot.

5.2 NLP

The NLP is essential to the scraper. Trainline’s website only works for certain date formats, 24 hour times, full and correctly spelt train stations, and will only complete a search successfully if it has all of the information. The NLP works with the chatbot to ensure all fields are full before passing this information to the scraper so it doesn’t fail. Not only this, but the chatbot has to be able to understand what a user is asking it to do. The NLP can identify, using SpaCy’s *spacy API documentation* (2022) parts of speech, when a user is inputting a cheapest ticket query or prediction query. It does this by looking for specific words in the sentence. To identify whether it is a cheapest ticket query it looks for words in the stems.json file, which matches similar words. If a query contains the words, once stemmed, ‘book’, ‘ticket’, ‘cheap’ or ‘buy’, it infers that the user is wanting to find the cheapest ticket, rather than asking about a delay prediction. The same goes for the prediction query.

One of the most important parts of the NLP for this project was the chatbot recognising train station names that were spelt incorrectly, or missing parts of the station name, as when these stations are inputted into the scraper they must be exact or the search will not run. To do this a package called ‘fuzzywuzzy’ was used, which does a ‘fuzzy match’ on station names and finds percentage matches to the station names inputted. If a station is not a 100% ‘fuzzy match’, the chatbot suggests any station that has an 85% match or higher. The unit also handles multiple different date formats, times and understands words like ‘today’ and ‘tomorrow’.

For sentence queries, like the one in figure 2, the NLP works by using SpaCy, looking at parts of speech tags like ‘NOUN’ or ‘VERB’ or for words within the stems.json file. For example, if the parts of speech tag is ‘ADP’ and one of the relating ‘from_station’ words in the stems.json file, the program extracts the following station name and sets it to the origin station.

5.3 KnowledgeBase and Inferring engine

The knowledge base and inferring engine were implemented with a library called Experta. Experta will store declared facts, such as where the user wants to go, or where they are coming from.

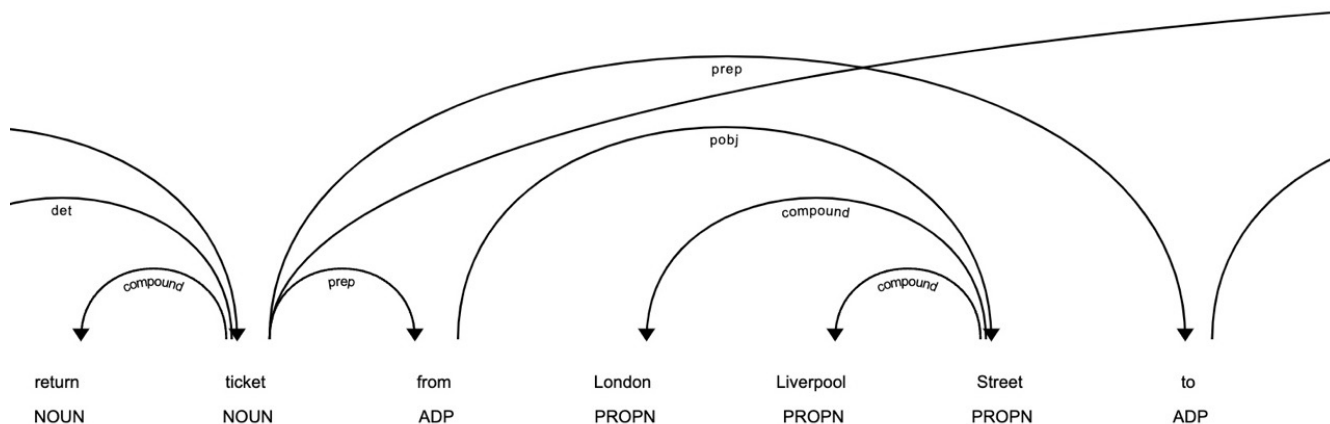


Figure 2: Example of SpaCy's parts of speech tagging on a cheapest ticket query

From the declared facts, the engine will infer which function should next be run based on rules associated with each. The rules will be evaluated, and if true the function runs. In the case of multiple rules being evaluated to be true, the engine will pick a random one.

After each function is run, all of the rules are re-evaluated so an updated set of available functions is available for selection. Once there are no remaining functions, the application will terminate.

5.4 Predictive Models

5.4.1 Bayes

To get the probabilities we can use stored procedures to extract the frequencies from the database.

Firstly the probability of A (the train being late to the second station) was easy enough to calculate as I simply extracted the frequency count of all the times the train arrived at the second station late and divided by the total pieces of data I had.

Secondly I calculated the probability of B given A (the probability the train was late to the first station by delay given it was late to the second station). To do this I extracted the frequencies of delays at the first station if the train arrived late to the second station. From here I summed all the frequencies that the train was late to the first station by at least the delay.

Next I calculated the probability of B given not A (the probability the train was late to the first station by delay given it was on time to the second station). To do this I extracted the frequencies of delays at the first station if the train arrived on-time or early to the second station. From here I summed all the frequencies that the train was late to the first station by at least the delay, then it was just a case of doing the calculation and returning the result as a probability.

5.4.2 Neural Network

Interestingly there are python libraries that will essentially build and train a neural network for you with limited input other than the shape and the training data. However we decided against using this as we felt it would be a hollow victory and not in the spirit of the coursework.

Therefore we thought that the best way to implement all the weights and values being stored was through dictionaries with string keys eg: "W1" with multidimensional arrays as the value.

This meant that I could implement a solution which has the capacity to expand the number of layers and neurons per layer easily with simple for loops, making the solution highly customizable.

To keep track of how well the network was learning we would test every single data point through it every 1000 iterations and get the root mean squared error (RMSE) for them all. This was then plotted

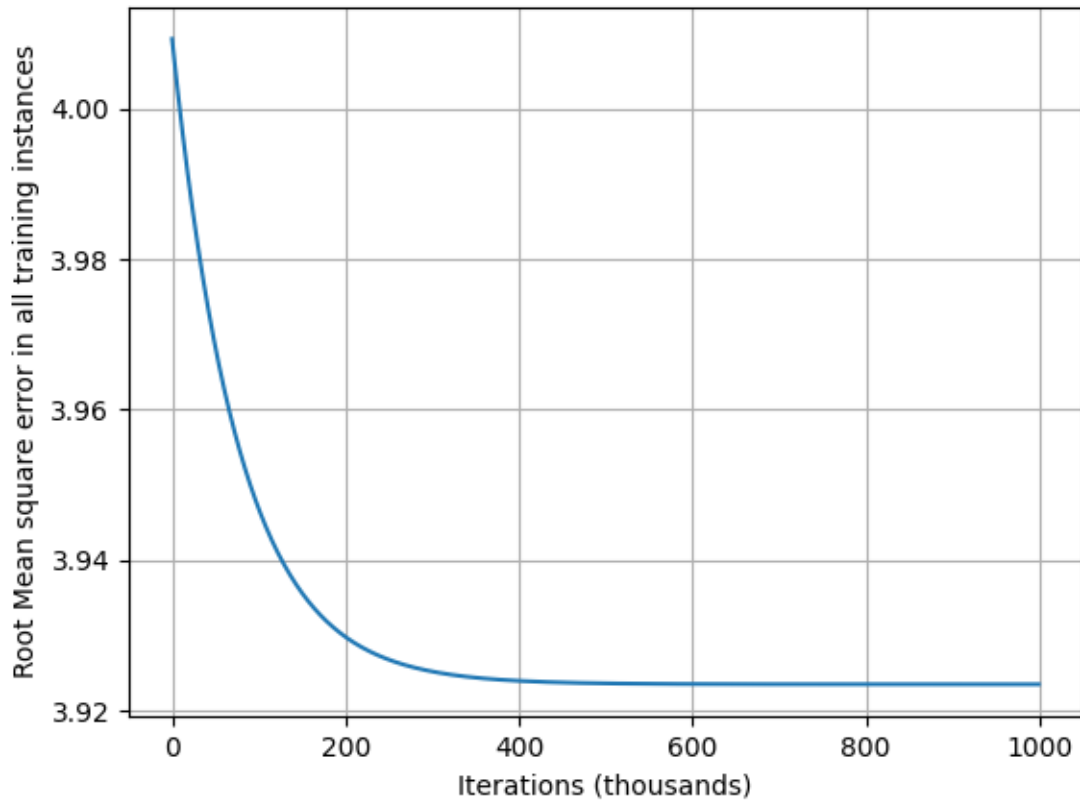


Figure 3: Neural network learning over one million iterations

on a graph seen in figure 3. This shows how quickly it could correct itself at the start and then converge to 0 with a shape similar to $y = x^{-1}$ graph. This was the shape that we expected to see.

We would also graph every hundred thousand iterations as seen in figure 4 and figure 5 to be able to see more clearly that is learning still. However this quickly has a very small range on the Y axis showing it is learning slower.

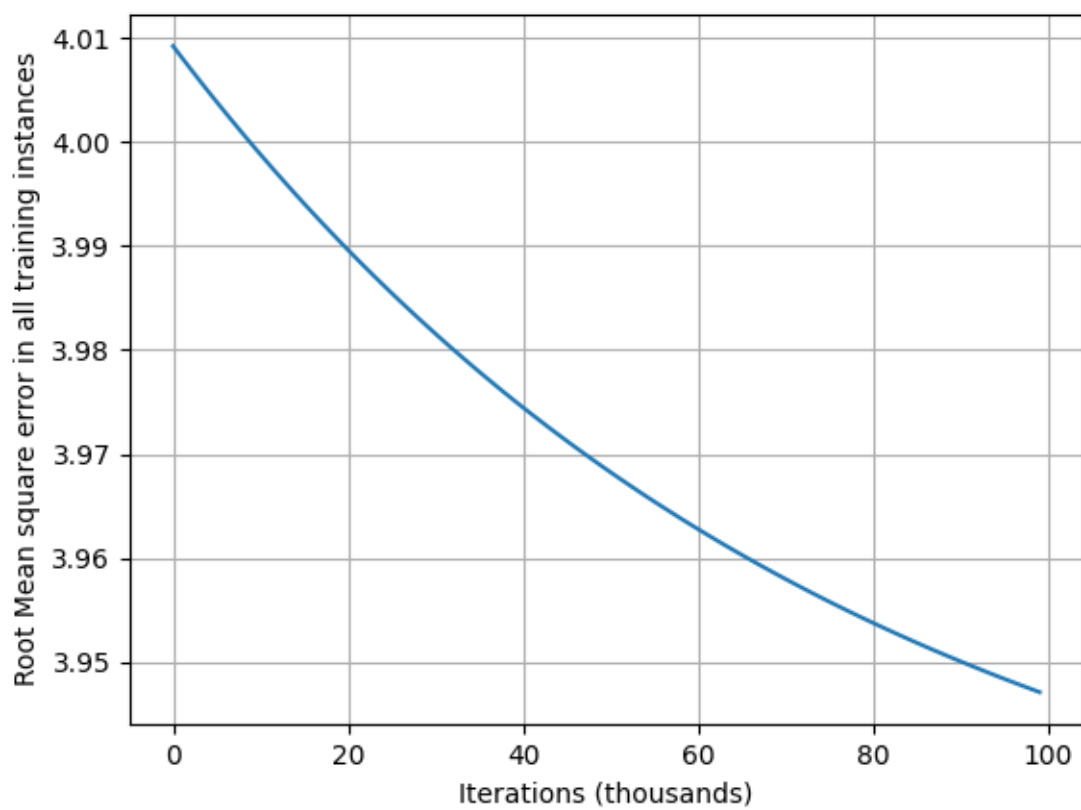


Figure 4: Neural network learning over first hundred thousand iterations

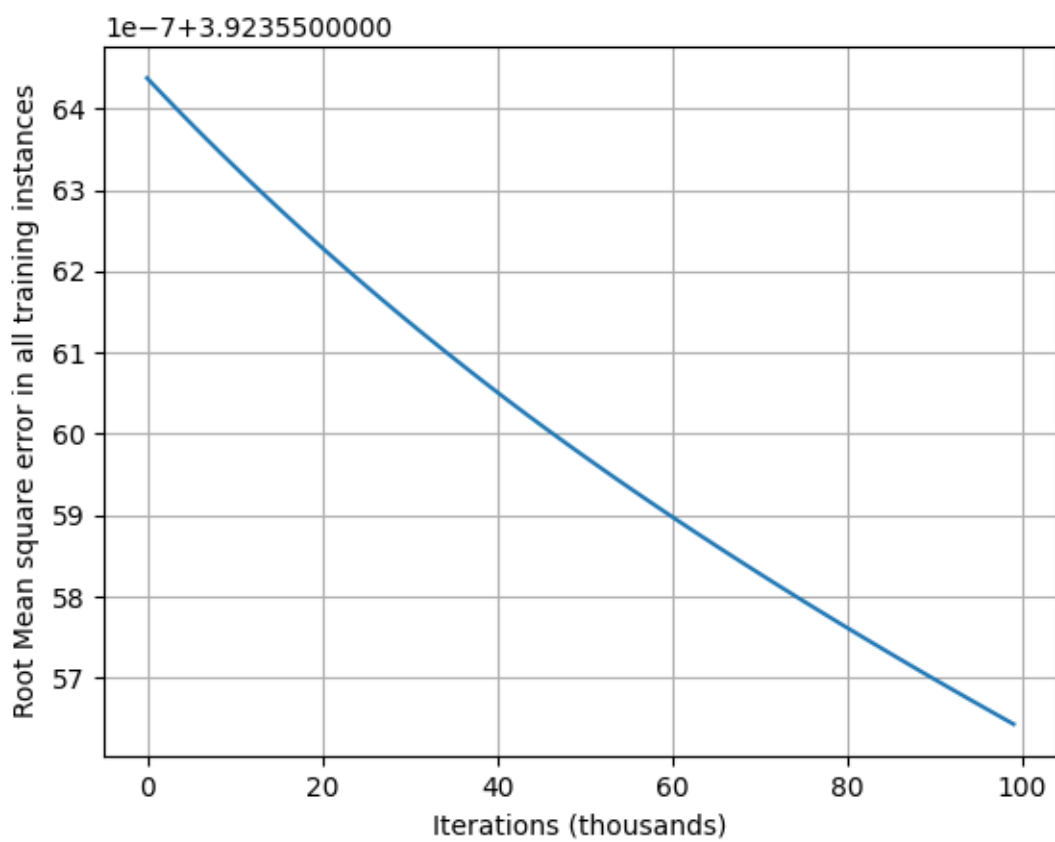


Figure 5: Neural network learning over last hundred thousand iterations

5.4.3 K Nearest Neighbour

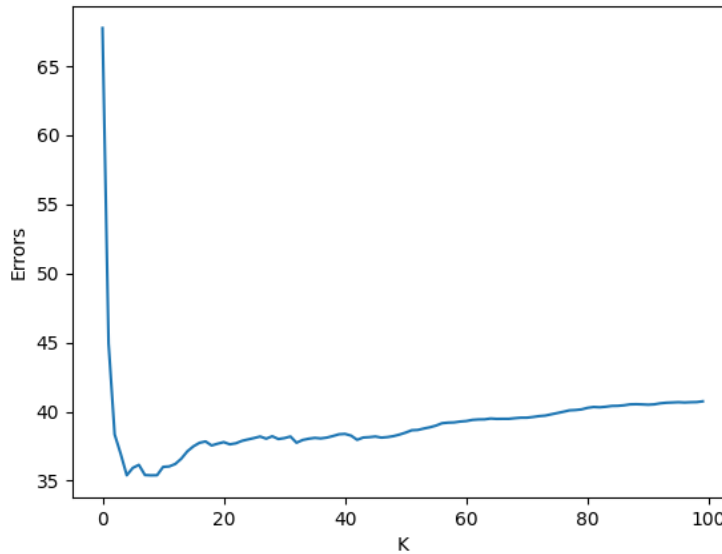


Figure 6: Searching for the correct K value (Y axis is multiplied by 10)

K	RMSE
9	3.538776543
5	3.539336
10	3.539625
8	3.54108125
6	3.592727778
11	3.600159504
12	3.60323125
7	3.614691837
13	3.620826627
14	3.659637245
4	3.6941
15	3.712949778
16	3.748832031
19	3.75489446
22	3.763205165

Figure 7: Sorted RMSE of K values

Once you have the data for the KNN it is a very simple algorithm but still heavily depends on the correct K value. So to find the best K value we set up a test to run one thousand iterations of selecting random data points from the data. Then for each piece of data I would create a KNN object with a different value of K ranging 1 to 100 (inclusive). Then it made a prediction from each object and compare it to the target outcome. From this I calculated the root mean square error for each value of K over the iterations.

The results plotted on a graph is in figure 6 and the top ranking results are included in the table in figure 7. The shape of the graph is exactly what I expected to see with very few neighbours having a very high RMSE and then having a dip where the ideal K value is with the RMSE gradually increasing again as the more neighbours are used.

From these results I concluded that 9 was the best value for K not only because it ranks at the top but because 10 and 8 rank 3rd and 4th in the list. However any value from 5 - 15 would still be accurate.

5.4.4 Comparison

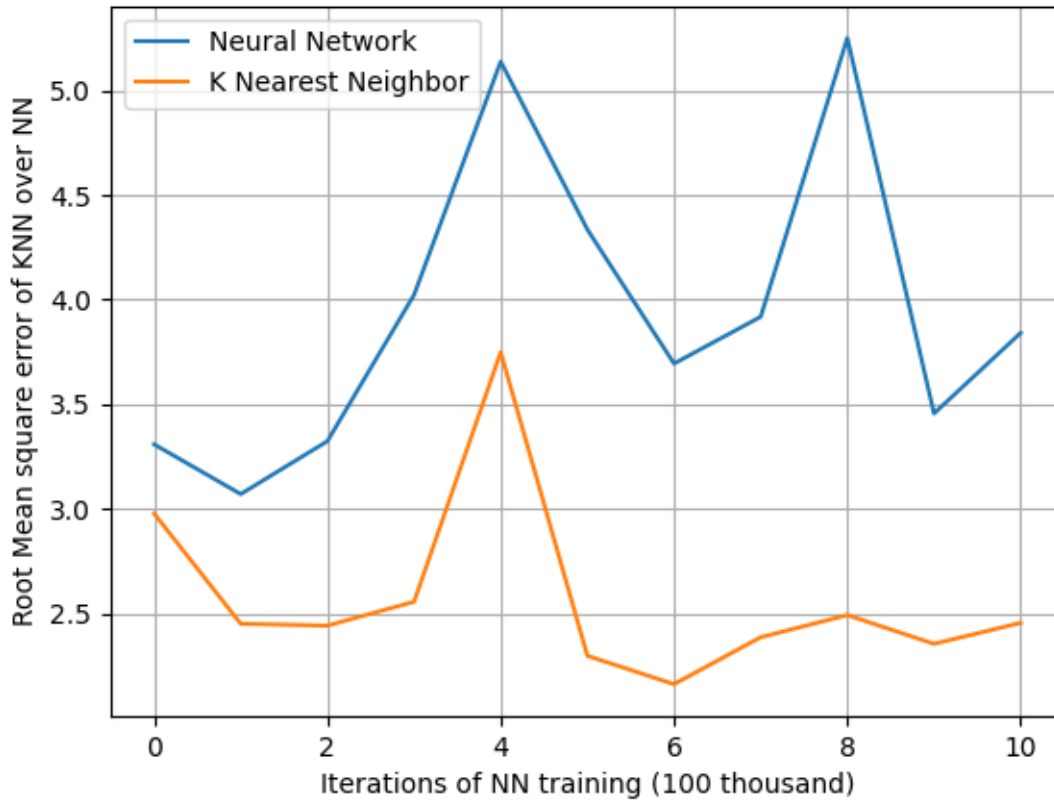


Figure 8: Comparison of neural network and K nearest neighbour for 1000 routes

We didn't think that the Bayesian model was very good as it is a classification model not a regression model, therefore not particularly useful to the user as they would want to know their expected time of arrival.

In figure 8 we compare the neural network and k nearest neighbours over one million iterations of training the neural network.

This was accomplished by randomly selecting a thousand data sets, running these datasets through both the models and comparing the predictions to the targets. Then train the neural network on the same dataset for a hundred thousand iterations and repeat.

As you can see the K nearest neighbour algorithm is consistently better than the neural network despite the improvements made so we decided to use that as our final predictive model.

Interestingly though when using 100 trains instead of 1000 the neural network learnt far faster and quickly overtook the k nearest neighbour as seen in figure 9 and figure 10.

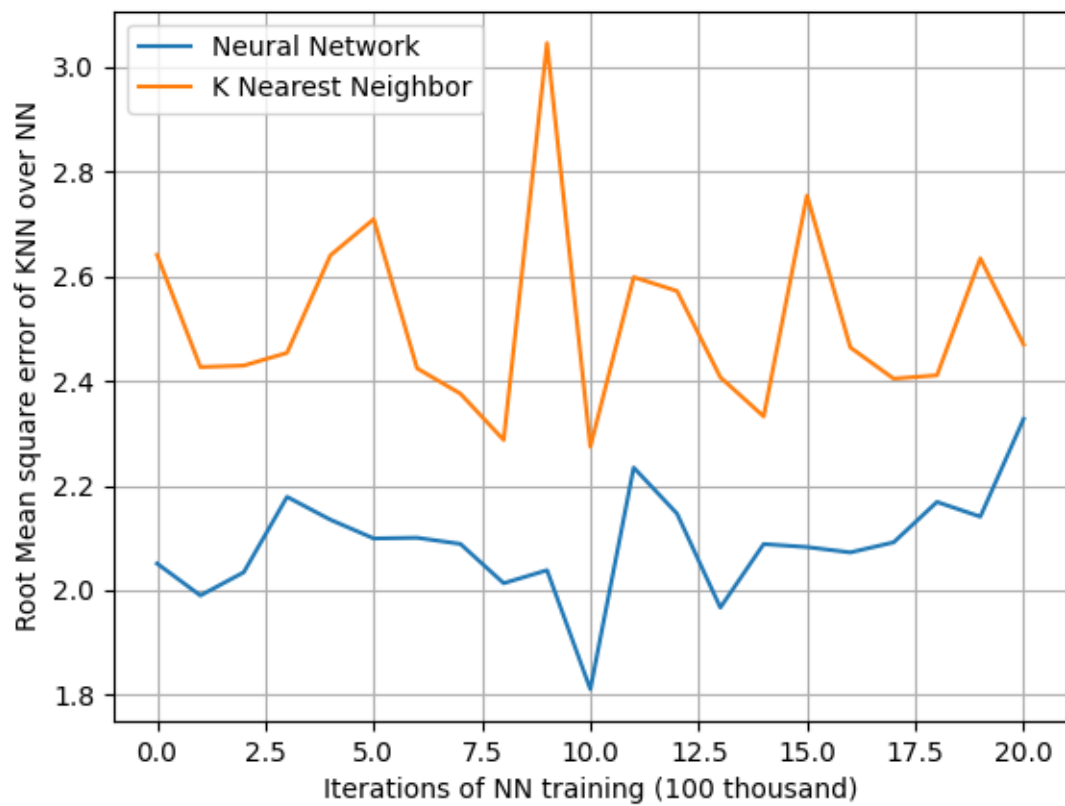


Figure 9: Comparison of neural network and K nearest neighbour for 100 trains

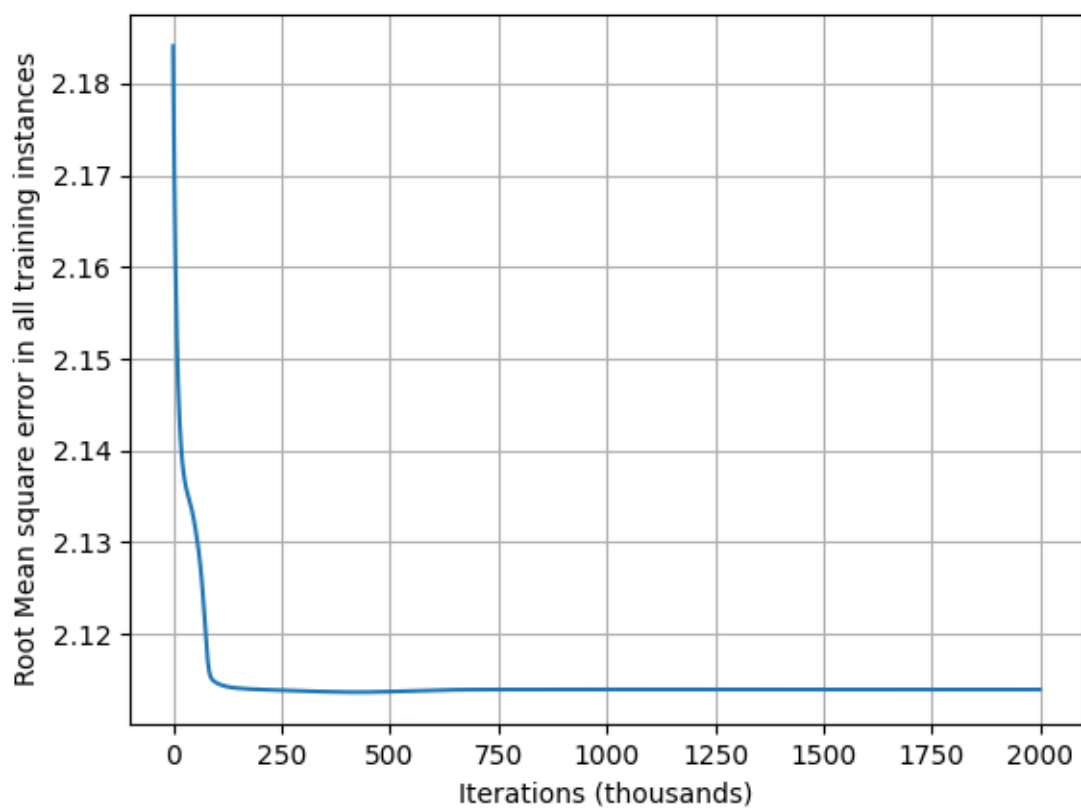


Figure 10: Neural network learning over two million iterations

6 Testing

6.1 Unit Testing

For the predictive models and NLP, unit tests were used to check that each function achieved its intended purpose; this approach is known as test driven development. Unit tests were written first, then code was developed to pass these tests. This meant the group was able to monitor which parts of development needed more work, and which were complete. Before each merge on GitHub, these tests were run to check that the code being pushed hadn't broken the previous commit. For the NLP, as the scope of what the chatbot could understand is massive, specific tests were written for what the group had outlined as chatflows it should accept.

6.2 Integration Testing

For our chatbot all the sections are very well encapsulated through design. This also means that they only interact in the knowledge base which is where all the crossover is.

To integrate the prediction models we mapped the station names to the TIPLOC in the database that was used throughout the prediction models. This also enabled us to validate that the stations were in the correct order and actually on the line being used. Because of the way the prediction models can be encapsulated this enabled us to simply add a handful of lines of code.

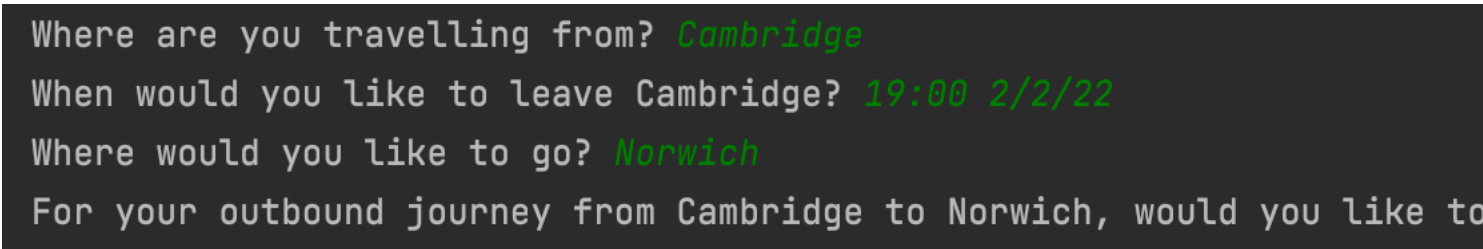
To integrate the NLP, all that was required was calling the correct functions when appropriate in order to process and validate the inputs provided by the user, along with appropriate messages detailing the progress towards extracting all of the information needed.

Integrating the web scraper was very similar to how the prediction models and NLP were integrated. As mentioned, and as a result of the encapsulation used, the extent of it was simply calling a function and using the values returned in a message to the user.

6.3 System Testing

System testing was completed through giving the fully integrated system a variety of initial queries, along with any subsequently required inputs to see if the output is as expected and to ensure that no other information is required from the user beyond what was initially supplied.

6.4 Usability Testing



```
Where are you travelling from? Cambridge
When would you like to leave Cambridge? 19:00 2/2/22
Where would you like to go? Norwich
For your outbound journey from Cambridge to Norwich, would you like to
```

Figure 11: Demo of question tailoring

To make the chatbot more user friendly we began by anticipating the formats what we expected a user to reasonably respond to each question. If the user responded in an unsupported format we would say the format that is required. This will make it easier for the user to enter the correct data.

The chatbot will also hold a conversation until the required information is collected.

The chatbot will ask very specific questions especially if the user states that the data was incorrect. This enables the user to respond correctly as they are unambiguous.

In order to make itself as clear as possible, the chatbot will use previous collected information in order to tailor the question, making it as clear as possible in what it is asking for as seen in figure 11.

7 Evaluation and Conclusion

The scope of this project is huge and open ended, and therefore an endless amount of additions could be made to the project to make it faster and smarter. However, for the time that the group had to work on it, we are extremely happy. The chatbot can handle full sentences, as well as prompting the user for any missing information. The information given to the chatbot is validated, and when appropriate the chatbot will make an attempt at what it thinks the user meant, or provide options the user can select from. The scraper is successful in finding a user the cheapest ticket for the parameters it has been given, however there is a significant delay when loading Trainline's website, which could be mitigated if we had been granted access to National Rails API. Additionally, delays are predicted with reasonable accuracy, but with more time could be improved further.

In conclusion, work was completed to a high standard and as a group we are pleased with what we have produced. The individual parts of this project were challenging, as was working as a group remotely. There were a few setbacks during the course of the project due to illness and extenuating circumstances. Despite this we have created a piece of work we are all proud of, and that we believe meets the required specification.

References

(2022).

URL: <https://www.selenium.dev/selenium/docs/api/javascript/index.html>

EBI (2022), ‘Go beyond an ai chatbot - try next generation artificial intelligence’, https://ebi.ai/ai-chatbot/?utm_term=ai+chatbot+customer+service&utm_campaign=ebi.ai-search&utm_source=adwords&utm_medium=ppc&hsa_acc=8425067570&hsa_cam=13742917407&hsa_grp=127019733598&hsa_ad=540213601615&hsa_src=g&hsa_tgt=kwd-486416788938&hsa_kw=ai+chatbot+customer+service&hsa_mt=e&hsa_net=adwords&hsa_ver=3&gclid=Cj0KCQiAraSPBhDuARIsAM3Js4rVF2asc7Q9fDTWeDI1qj5vyc0J-69hVEGRlJvaSpfJs_1EFWD-sJ0aAkRyEALw_wcB.

Google (2016), ‘Google assistant’, https://assistant.google.com/intl/en_uk/.

Search, Compare and Buy Cheap Train Tickets (2022).

URL: <https://www.thetrainline.com/>

SINTEF, A. F., Følstad, A., Sintef, SINTEF, M. S., Skjuve, M., Dublin, U. C. & MetricsExpand, C. (2019), ‘Chatbots for customer service: User experience and motivation’, <https://dl.acm.org/doi/pdf/10.1145/3342775.3342784>.

spacy API documentation (2022).

URL: <https://spacy.io/api/doc>

Stephen Roller, Emily Dinan, N. G. D. J. M. W. Y. L. J. X. M. O. K. S. E. M. S. Y.-L. B. J. W. (2020), Recipes for building an open-domain chatbot.

TFL (2020), ‘Tfl travelbot’, <https://www.facebook.com/tfltravelbot/>.

Wang, W. (2018), ‘Artificial intelligence modules (cmp6040, 7028) coursework specification’.

A Chat Flow

C: How can I help?

U: What is the cheapest return ticket for four adults and 2 children from Milton Keynes Central to Norwich, arriving at 13:00 on 15/01/2022 and returning on 17/01/2022 arriving by 17:00?

C: The cheapest ticket is £33.75 [LINK](#)

END

C: How can I help?

U: *I'd like to book a ticket*

C: What is your origin station?

U: *Norwich*

C: What is your destination station?

U: *Milton Keynes*

C: Single or return?

U: *Return*

C: When would you like to leave {Norwich}?

U: *16:00 22/02/2022*

C: When would you like to return to {Norwich} by?

U: *18:00 25/02/2022*

C: How many adults are traveling?

U: *1*

C: How many children are traveling?

U: *0*

C: You would like to book a return ticket to Milton Keynes Central from Norwich, leaving Norwich after 16:00 on 22/02/2022. You would like to return to Norwich by 18:00 on 25/02/2022. 1 Adult, 0 children.

Is this correct? Y or N

U: *Y*

C: The cheapest ticket for this journey is £78.99

Book your ticket here [LINK](#)

END

C: How can I help?

U: Predict delay

C: How much has your train been delayed so far?

U: 3 minutes

C: At which station?

U: Weymouth

C: What station would you like the delay to be predicted at?

U: Southampton

C: The predicted delay at {Southampton} is 6 minutes.

Or

C: The train is predicted to arrive at {Southampton} on time.

END

C: How can I help?

*U: What is the predicted delay at Southampton if the train was delayed 3 minutes
from Weymouth?*

C: The predicted delay at {Southampton} is 6 minutes.

or

The train is predicted to arrive at {Southampton} on time.

END