# ISYS30221 Artificial Intelligence 2021-22

## Artificial Intelligence - Coursework Documentation

## 1- About this submission

|  |  |
| --- | --- |
| Student Name | Ryder Franklin |
| Student ID | N0853033 |
| Chatbot Topic | Supernatural (the tv show) |
| Tasks implemented in this submission (a,b,c,or d) | C and D |
| Files inventory (excluding this file) | kb.csv  knowledge.csv  azureFaceRecognition.py  azureImageClassification.py  main.py  knowledgeProcessing.py  neuralNetwork.py  similarity.xml  spnChatbot1-aiml.xml  supernaturalCNNModel.h5 |
| Demo video URL |  |
| Checklist | I will submit this file separately (without compression) into DropBox  All other files are zipped and will be submitted into DropBox  The demo video is recorded as instructed, and the sharing link is inserted above  I have made sure that the demo video is shared according to the instructions, so that I allowed everybody in the university to view it.  All the sections below are populated accordingly. |

## 2- Design notes (shrink/grow as needed, add images where applicable)

|  |  |
| --- | --- |
| General explanations of the system and its goals | The Supernatural Chatbot – nicknamed Dave – aims to provide facts and details about various characters, species, and events within the Supernatural universe. This is to centralise information about the show as it has 15 years’ worth of content to sift through. Additionally, Dave should be able to identify a character from the show using an image provided by the user. |
| The system requirements, i.e., the list of what the system should do/have from a user’s perspective | Users can ask Dave for a joke, for it to access the Supernatural fandom Wiki, can ask some questions, can check some logical facts, and can have short conversations with Dave.  In the latest update, users can now ask to use Dave’s neural network and with this, identify if an image file provided is Sam Winchester, Dean Winchester or Castiel. This can be done locally or using Azure / the cloud. |
| The employed AI techniques, and the explanation of program codes and the supplied files. | **Task A and B**  My chatbot’s main features are that it uses AIML patterns, First Order Logic (FOL), TF-IDF and cosine similarity and it also accesses the Fandom webpage for data on my topic. It also speaks using Python’s inbuilt library – pyttsx3 – and can recognise a speech input from the user.  First, Dave introduces itself and lets the user decide whether they will input using the keyboard/typing or microphone/speaking. If they choose to use speech input/recognition, they will be prompted to use the correct microphone from the list of connected microphones to their computer. The user is then prompted to give an input within a while loop.  Each input is first processed by Dave’s AIML patterns. If they match a pattern specified within the AIML file, the template specified is followed and, in some cases, the function that matches the pattern is called to handle it. For example, one of the AIML patterns calls the py\_jokes library and tells the user a programming joke. These patterns are also used to communicate basic responses such as greetings.  This pattern recognition is used to call many of Dave’s features including FOL and the fandom wiki API. First Order Logic is used by the program to identify both “Check that x is y” and “I know that x is y” statements, using a given knowledge base (kb) csv file.  The final pattern recognition feature is the fandom wiki API. Currently, Dave retrieves a 3-sentence summary from a given page upon the fandom wiki for Supernatural. This is done by using the fandom-py library and makes use of regex to format the summary in a suitable manner.  Finally, if a pattern cannot be matched to the user input, the TF-IDF of the user’s query is calculated and the cosine similarity between it and each line of the knowledge.csv file to find the best answer / match to the user’s question.  **Task C and D**  Dave still contains all the previous functionality discussed in task A and B – telling jokes, using TF/IDF, using a knowledge base and the use of an AIML file to respond to users.  For task C, I have added a local Convolutional Neural Network (CNN) within the file neuralNetwork.py that is trained on images of three characters – Sam, Dean and Castiel – using 607 images for training data and 218 images for validation data. The CNN has nine layers. First, the input layer, with a shape of (300, 300, 3) and then a convolution layer with a filter size of 128 and an activation of relu. After each convolution layer, a max pooling is applied, with a size of (2,2). This is to reduce the dimensions of the feature maps and therefore the parameters to learn, along with the computation performed. I apply another set of Conv2D and max pooling before flattening the model, applying a dropout of 0.5 and then the final two layers. A dense layer of 32, with activation of relu is applied before the final dense layer of sigmoid activation. Sigmoid is applied last as it has the best impact of accuracy here, since the data and parameters have been flattened to an almost 1-1 scale. This model is then saved as supernaturalCNNModel.h5 for use within the chatbot.  Dave activates as previously discussed and, once within the while loop, the user may input “use neural network” which triggers a response from Dave using an AIML pattern. The program is directed to the handleNeural function, and the user prompted to input further information before continuing.  From here, the user may rerun the CNN, which is useful if the local dataset has been updated, or they can use a previously generated model. They are provided with the option to test an image locally or on the cloud.  Local testing of images occurs within runNN function, which uses the previously generated h5 file, and takes a file path input to an image from the user. To ensure a consistent input, the image is reshaped to 300x300 size and then reformatted to be predicted using the model. This returns a list of decimal values, Dave takes the maximum value (aka the most likely), selects the name of the class using the index and then converts the decimal to a percentage. The name and percentage are then returned to the user. However, if the max prediction value is under the threshold of 75%, the identified person / class is set to Unknown, as under this value, the model cannot confirm that the identified person is correct.  Cloud testing of images occurs within my runAzureNN function and is split into two sections with functions that are pulled from their individual python files. This is to ensure that main doesn’t become too large and confusing to navigate. First, the user must choose whether they wish to identify a character or detect a face. To identify a character, the function runImageClassification is called from azureImageClassification.py file and Azure’s image classification is used, with a model trained on Azure Custom Vision. This function is special and can accept a folder of images, rather than just one, and returns a sideview / popup window that shows the tag alongside each image. Additionally, Azure generates a series of graphs mapping the precision, recall and AP of the model’s iteration, shown below.    Secondly, the user can select to outline a face in a single image. Once a file path to an image is provided, this is passed on to the outlineDetectedFace function from my azureFaceRecognition.py file. The image is opened as a byte stream and, if a face can be found, it is outlined using a red box and shown in a popup to the user.  After any of these functions are used and an output provided, Dave informs the user that it has exited all neural network options and returned to the main section of code, allowing any AIML responses to be used once more. |

## 3- Conversation log (insert text, screenshots and/or images as required)

## 10 conversation pairs for each task are enough)

|  |
| --- |
|  |