Chatbot report

This code was split into 3 files, one for intents, one for training data and one for the actual chatbot itself.

File 1

This shows the intents file which contains the pre-programmed lines of conversation. These are split by a tag which sorts user inputs into potential groups, the patterns contain what the user is likely to say and finally the responses contain what the chatbot will print in return.

```
{"intents": [
  {"tag": "greetings",
  "patterns":  ["hello", "hi", "hey", "yo", "sup"],
  "responses":  ["Hello mortal", "Hi", "Greetings"]},

  {"tag":  "goodbye",
  "patterns":  ["goodbye", "bye", "later", "ciao"],
  "responses": ["Goodbye", "Be gone", "Perish", "See you later"]},
```

More lines of conversation can be programmed in following this convention. There are several more in the project however they weren't all displayed in this report in order to save space.

File 2

```
import random
import json
import pickle
import numpy as np

import nltk
from nltk.stem import WordNetLemmatizer
from nltk.stem.lancaster import LancasterStemmer
stemmer = LancasterStemmer()

from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Activation, Dropout
from tensorflow.keras.optimizers import  SGD

lemmatizer = WordNetLemmatizer()

intents = json.loads(open('intents.json').read())
```

Files and modules being imported.

```
19      words = []
20      classes = []
21      documents = []
```

This creates empty classes for sorting purposes.

```
for intent in intents ['intents']:
    for pattern in intent ['patterns']:
        word_list = nltk.wordtokenize(pattern)
        words.extend(word_list)
        documents.append((word_list, intent['tag']))
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

This section of code splits sentences into individual words in order to sort them into their respective classes.

```
22      ignore_letters = ['.', ',', '!', '?']
```

```
32      words = [stemmer.stem(w.lower()) for w in words]
33      words = [lemmatizer.lemmatize(word) for word in words if word not in ignore_letters]
34      words = sorted(set(words))
```

These pieces of code ignore punctuation to avoid confusion, remove duplicate words and finally convert words into an inflected form so they can be more easily processed.

```
68      #Neural network
69      model = Sequential()
70      model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
71      model.add(Dropout(0.5))
72      model.add(Dense(64, activation = 'relu'))
73      model.add(Dropout(0.5))
74      model.add(Dense(len(train_y[0]), activation = 'softmax'))
75
76      sgd = SGD(lr=0.01, decay =1e-6, momentum = 0.9, nesterov = True)
77      model.compile(loss = 'categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
78
79      model.fit(np.array(train_x), np.array(train_y), epochs = 1000, batch_size = 5, verbose = 1)
80      model.save('Chatbot.h5', hist)
81      print("Training complete")
```

Here the code for the neural network can be seen, this code starts by creating a sequential model. This type of model was chosen as there will only be one input at a time and there is no need for layers to be connected beyond the previous and next layer.

The dropout function is used to ignore the weighting of randomly selected neurons to prevent the model from overfitting which would make it unable to perform against uncoded data.
Stochastic gradient descent is used for optimisation purposes.
Finally the model is then trained 1000 times using the number that is set to epochs and then saved to a file. A debug message is then printed when the training is completed.

```
def bag_of_words (j, words):
    bag = []

    j_words = nltk.wordtokenize(j)
    j_words = [lemmatizer.lemmatize(word.lower()) for word in j_words]

    for j1 in j_words:
        for j2, w in enumerate(words):
```

This section of code was my first attempt at creating the response framework for the chatbot, however I struggled to figure out how to work this method so I instead opted for a different approach.

File 3

```
15      intents = json.loads(open('intents.json').read())
16
17      words = pickle.load(open('words.pkl', 'rb'))
18      classes = pickle.load(open('classes.pkl', 'rb'))
19      model = load_model('Chatbot.h5')
```

This section loads the training data that was created and saved to a pkl file as well as the model that was trained on the data and saved as an h5 file.

```
22      def clean(sentence):
23          input_words = nltk.word_tokenize(sentence)
24          input_words = [lemmatizer.lemmatize(word) for word in input_words]
25          return input_words
```

This snippet of code creates a function for cleaning sentences by splitting it into individual words and then converting the words into their inflected form.

```
def bag_of_words(sentence):
    words_in_sentence = clean(sentence)
    bag = [0 for _ in range(len(words))]
    for w in words_in_sentence:
        for i, word in enumerate(words):
            if word == w:
                bag[i] = 1
    return np.array(bag)
```

Here the sentence cleaning function is called and run for the number of words in the sentence, these words are then appended to the bag of words to allow the neural network to calculate which class of words the sentence likely applies to.

```
def predict_class(sentence):
    bow = bag_of_words(sentence)
    res = model.predict(np.array([bow]))[0]
    margin_of_error = 0.1
    results = [[i, r] for i, r in enumerate(res) if r > margin_of_error]
```

The bag of words is then run through the model so calculations can be made to determine possible results, these results are then filtered out if the margin of error is above 10%.

```
results.sort(key=lambda x: x[1], reverse=True)
return_list = []
for r in results:
    return_list.append({'intent': classes[r[0]], 'probability': str(r[1])})
return return_list
```

From here the classes are sorted into a list in descending order of probability in order to allow for the most likely class to be selected.

Initially I planned to have the chatbot output the following if the input didn't have a strong probability of belonging to any of the classes but I was unable to find a workable solution for this.

```
    if r > margin_of_error:
        results = [[i, r] for i, r in enumerate(res)]
    else:
        print "Sorry I didn't understand that"
```