Universidade do Minho
Departamento de Informática

Mestrado Integrado em Engenharia Informática

Inteligência Ambiente: Tecnologias e Aplicações

TP2 - Inteligência Ambiente com suporte de Processamento de Linguagem Natural

---

Grupo 5
a86617 Gonçalo Nogueira
pg42829 Francisco Borges
pg42846 Nuno Pereira
a82529 Carlos Afonso

Braga
Janeiro, 2021

Conteúdo

Lista de Figuras

1 Introdução

Com a realização deste trabalho prático pretende-se desenvolver um sistema inteligente com suporte de Processamento de Linguagem Natural, tendo como base o software desenvolvido aquando do primeiro trabalho prático, em particular o logger para captura de dados de sensores como o teclado e/ou rato.

O objetivo principal foi criar um socket entre dois loggers para que sejam utilizados na forma de um chat. Para cada mensagem enviada o sistema vai avaliar a emoção predominante no texto e exibe-a junto com a mensagem.

Quando o chat é finalizado o sistema apresenta um resumo da conversa.

2 Descrição da abordagem de chat escolhida para as sugestões

A abordagem do grupo relativamente á escolha de respostas perante cada frase introduzida pelo utilizador foi utilizar redes neuronais para treinar o bot e portanto dividimos esta parte do trabalho em quatro partes.

2.1 Ficheiro *json* com dados já pré-definidos

Inicialmente foi criado um ficheiro json designado de *intents*, este ficheiro está divido em *tags* que agrupam um conjunto de frases padrão bem como uma panópolia de sugestões de resposta possíveis a frases desse tipo introduzidas pelo utilizador.

```
{"tag": "greeting",
 "patterns": ["Hi", "How are you", "Is anyone there?", "Hello", "Good day", "Whats up"],
 "responses": ["Hello!", "Good to see you again!", "Hi there, how can I help?"],
```

Figura 1: Exemplo de frases padrão bem como sugestões possíveis de resposta

2.2 Treinar o chatbot com recurso a redes neuronais

Depois de criado o ficheiro json é preciso treinar o chatbot com recurso a uma rede neuronal, criando um modelo que posteriormente permita a previsão da tag associada á frase que o utilizador escreve.

No entanto, é preciso realizar um tratamento de dados sobre o ficheiro de padrões e sugestões visto que as redes neuronais funcionam apenas com valores numéricos e não com strings de caractéres.

O primeiro passo a tomar foi a *tokenização* de cada frase, este passo serve para dividir a mensagem num conjunto de palavras separadas individualmente numa lista. Isto foi possível através do uso do módulo *nltk* da linguagem **python** que fornece a função **word.tokenize** que transforma a *string "Hello world"* numa lista com dois elementos, *"Hello"* e *"world"*.

```
for intent in intents['intents']:
    for pattern in intent['patterns']:
        # pegar na frase e dividi-la pelas palavras
        word_list= nltk.word_tokenize(pattern)
        words.extend(word_list)
        documents.append((word_list, intent['tag']))
        if intent['tag'] not in classes:
            classes.append(intent['tag'])
```

Figura 2: Tokenização das palavras

O próximo passo passa lematizar as palavras agora separadas utilizando o método **WordNetLemmatizer** do módulo *nltk.stem* que consiste em reduzir todas as diferentes formas de uma palavra para a sua forma básica, por exemplo, *builds, building ou built*, reduzem-se todas a uma palavra básica e simples *build*.

```
# lemmatization
words =[lemmatizer.lemmatize(word) for word in words if word not in ignore_letters]
```

Figura 3: Lematização das palavras

De seguida, falta apenas um último passo, a transformação das palavras naquilo que é designado de modelo **bag of words**, que representa a multiplicidade das palavras guardadas, ou seja, quantas vezes estão presentes. Este passo é necessário pois os modelos neuronais apenas reconhecem este modelo como forma de *input*.

```
for document in documents:
    bag=[]
    word_patterns = document[0]
    word_patterns = [lemmatizer.lemmatize(word.lower()) for word in word_patterns]
    for word in words:
        bag.append(1) if word in word_patterns else bag.append(0)
```

Figura 4: Transformação no modelo bag of words

Após concluídos todos estes passos, estamos agora pronto a criar um modelo neuronal com a utilização do módulo **tensorflow** (módulo *open-source* introduzido pela *Google* que facilita a utlização de redes neuronais).

No nosso caso optamos por um modelo simples constituido por apenas três camadas, a primeira camada é a camada de *input*, uma camada intermédia e uma terceira camada de output que utiliza a função de ativação **softmax** responsável por normalizar o output para uma distribuição probabilística para prever a que *tag* corresponde uma frase. A função baseia-se *axioma de escolha de Luce* que resumidamente afirma que a probabilidade de escolha de um itém sobre outro de uma seleção de itens não é afetado pela presença ou falta de outros itens na seleção.

```
# modelo neural network
model = Sequential()
model.add(Dense(128, input_shape=(len(train_x[0]),), activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(64, activation='relu'))
model.add(Dropout(0.5))
model.add(Dense(len(train_y[0]), activation='softmax'))
```

Figura 5: Criação das três camadas do modelo

Finalmente para a compilação do modelo foi utlizado o método iterativo **Stochastic gradient descent** usualmente abreviado por **sgd** que é um método bastante comum de optimização de uma função objetivo.

```
sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
model.compile(loss='categorical_crossentropy', optimizer=sgd, metrics=['accuracy'])
```

Figura 6: Utilização do sgd

Utilizando o método **fit** com as opções descritas previamente e os valores de frases padrão como input e as sugestões para essas frases como output conseguimos agora treinar o chatbot para futuramente com recurso ao modelo criado prever a que tag corresponde uma frase introduzida pelo utilizado ao interagir com o nosso chatbot.

```
hist = model.fit(np.array(train_x), np.array(train_y), epochs=200, batch_size=5, verbose=1)
model.save('chatbotmodel.h5', hist)
```

Figura 7: Exemplo de frases padrão bem como sugestões possíveis de resposta

2.3 Previsão da tag correspondente

Após a introdução de uma frase pelo utilizador foi criado um método para prever a sua *tag* correspondente.

Novamente é necessário tratar as frases inseridas da mesma forma que anteriormente (*tokenização* e *lematização* e criação de um *bag of words*).

A previsão é feita através da função **predict** utilizada no modelo já criado previamente.

```
def clean_up_sentence(sentence):
    sentence_words = nltk.word_tokenize(sentence)
    sentence_words = [lemmatizer.lemmatize(word) for word in sentence_words]
    return sentence_words
```

Figura 8: Tratamento da frase introduzida pelo utilizador

```
#preve a que classe pertence a frase do input
def predict_class(sentence):
    bow = bag_of_words(sentence)
    res = model.predict(np.array([bow]))[0]
    ERROR_THRESHOLD = 0.2
    results = [[i, r] for i, r in enumerate(res) if r > ERROR_THRESHOLD]

    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
```

Figura 9: Método para previsão da tag

2.4 Escolha da resposta

Após feita a previsão da tag que mais probabilidade tem de ser da frase do utilizador é selecionada uma resposta aleatória do conjunto de respostas presentes no ficheiro *json*.

```
#consoante a tag devolve uma resposta random dessa tag
def get_response(intents_list, intents_json):
    tag = intents_list[0]['intent']
    list_of_intents = intents_json['intents']
    for i in list_of_intents:
        if i['tag'] == tag:
            result = random.choice(i['responses'])
            break
    return result
```

Figura 10: Método para escolha de resposta ao utilizador

3 Descrição da abordagem de análise de sentimentos utilizada

Relativamente à análise de sentimentos utilizamos o módulo NRCLex para determinarmos que emoção melhor se adequa a uma frase.

Após a tokenização da frase, utilizando o método NRCLex() e posteriormente o top-emotions obtemos uma lista de tuplos de emoções bem como a sua respectiva probabilidade. Neste módulo existem dez emoções contempladas, sendo elas:

- Medo
- Raiva
- Antecipação
- Confiança
- Surpresa
- Positivade
- Negativade
- Tristeza
- Repulsa
- Alegria
- Neutra

Além das dez emoções mencionadas sempre que a principal emoção indicada pelo módulo não ultrapassar uma probabilidade de 0.3 assumimos a frase como sendo neutra.

```
def predict_emotion(sentence):
    with open('logger.txt', 'a') as f:
        word_list = nltk.word_tokenize(sentence)
        for i in range(len(word_list)):
            emotion = NRCLex(word_list[i])
        emo = emotion.top_emotions
        emo2 = emo[0][1]
        if emo2 > 0.3:
            f.write(message + ' (' + emo[0][0] + ')' + '\n')
            print(emo[0][0])
        else:
            f.write(message + ' ' + '(neutral)' + '\n')
            print('(neutral)')
```

Figura 11: Determinação da emoção de uma frase

4 Descrição da abordagem de resumo escolhida

Relativamente ao resumo da conversa foi feita apenas a contagem do número total de frases escritas pelo utilizador, sendo esse o número de linhas no ficheiro logger.txt que guarda todas as frases introduzidas bem como a contagem de emoções presentes na conversa, procurando o número de vezes que a palavra correspondente a cada emoção surge no ficheiro.

```
def resumo_conversa():
    file = open('logger.txt', 'r')
    data = file.read()
    occurrences_fear = data.count("fear")
    occurrences_anger = data.count("anger")
    occurrences_anticipation = data.count("anticipation")
    occurrences_trust = data.count("trust")
    occurrences_surprise = data.count("surprise")
    occurrences_positive = data.count("positive")
    occurrences_negative = data.count("negative")
    occurrences_sadness = data.count("sadness")
    occurrences_disgust = data.count("disgust")
    occurrences_joy = data.count("joy")
    occurrences_neutral = data.count("neutral")
    num_lines = sum(1 for line in open('logger.txt'))
```

Figura 12: Método utilizado para indicar o resumo da conversa

```
print('Numero de mensagens', num_lines)
if occurrences_fear > 0: chatWindow.insert(END, "\n"+ str(occurrences_fear) + ' Frases com medo')
if occurrences_anger > 0: chatWindow.insert(END, "\n"+ str(occurrences_anger) + ' Frases zangadas')
if occurrences_anticipation > 0: chatWindow.insert(END, "\n"+ str(occurrences_anticipation) + ' Frases ansiosas')
if occurrences_trust > 0: chatWindow.insert(END, "\n"+ str(occurrences_trust) + ' Frases confiantes')
if occurrences_surprise> 0: chatWindow.insert(END, "\n"+ str(occurrences_surprise) + ' Frases surpresas')
if occurrences_positive > 0: chatWindow.insert(END, "\n"+ str(occurrences_positive) + ' Frases positivas')
if occurrences_negative > 0: chatWindow.insert(END, "\n"+ str(occurrences_negative) + ' Frases negativas')
if occurrences_sadness > 0: chatWindow.insert(END, "\n"+ str(occurrences_sadness) + ' Frases tristes')
if occurrences_disgust > 0: chatWindow.insert(END, "\n"+ str(occurrences_disgust) + ' Frases repulsivas')
if occurrences_joy > 0: chatWindow.insert(END, "\n"+ str(occurrences_joy) + ' Frases alegres')
if occurrences_neutral > 0:  chatWindow.insert(END, "\n"+ str(occurrences_neutral) + ' Frases neutras')
```

Figura 13: Informar no ecrã ao utilizador o resumo da conversa

5 Chatbot GUI

Para a criação go GUI foi usado o package *tkinter*, isto permitiu a criação de uma janela onde foram colocadas duas caixas de texto, uma para escrever o texto do utilizador e outra para mostrar o texto tanto do utilizador como a resposta do Bot e um botão para submeter o texto para ser analizado.

```
# window title
root.title('Chat Bot')

# window size
root.geometry('400x500')

#add scrollbar
scrollbar = Scrollbar(root)
scrollbar.place(x=375, y=5, height=385)

# create chat window
chatWindow = Text(root, bd=1, bg='black', width= 50, height= 8, fg='white', yscrollcommand=scrollbar.set)
chatWindow.place(x=6, y=6, height=385, width=370)

scrollbar.config(command= chatWindow.yview)

#Create messageWindow
messageWindow = Text(root, bg='black', width=30, height=4, fg='white')
messageWindow.place(x=120, y=400, height=88, width=260)

#Create enter button
Button = Button(root, text='send', bg='white', activebackground='light blue', width=12, height=5, font=('Arial', 20), command=sendYou)
Button.place(x=6, y=400, height=88, width=115)


root.mainloop()
```

Figura 14: Código do GUI para o chatbot

Para mostrar o que o utilizador escreve e as respostas do *bot* neste *GUI*, criamos uma função que está ligada ao botão **send** presente no *GUI* que sempre que é carregado transmite o que está escrito na caixa de texto para uma váriavel que é usada para o tratamento da resposta do bot. Em seguida, apresenta a mensagem no ecrã bem como a respetiva resposta como nos mostra a figura 15.

```
def sendYou():

    message = messageWindow.get("1.0",'end-1c')
    sendYou = "You : " + message
    chatWindow.insert(END, "\n"+ sendYou)
    if message == "quit":
        print('Resumo da conversa:')
        resumo_conversa()
        chatWindow.yview(END)
    else:
     predict_emotion(message)
     ints = predict_class(message)
     # saida
     res = get_response(ints, intents)
     sendBot = "Bot : " + res
     chatWindow.insert(END, "\n" + sendBot)
     chatWindow.yview(END)
```

Figura 15: Código que trata do display das mensagens

6 Conclusão

O trabalho foi concluido com êxito tendo o grupo concluido todas as tarefas propostas.

Ao longo do trabalho recolhemos e solidificamos conhecimentos na àrea do processamento de linguagem natural e ambientes inteligentes tal como aplicamos o conhecimento obtido nas aulas teóricas e práticas.

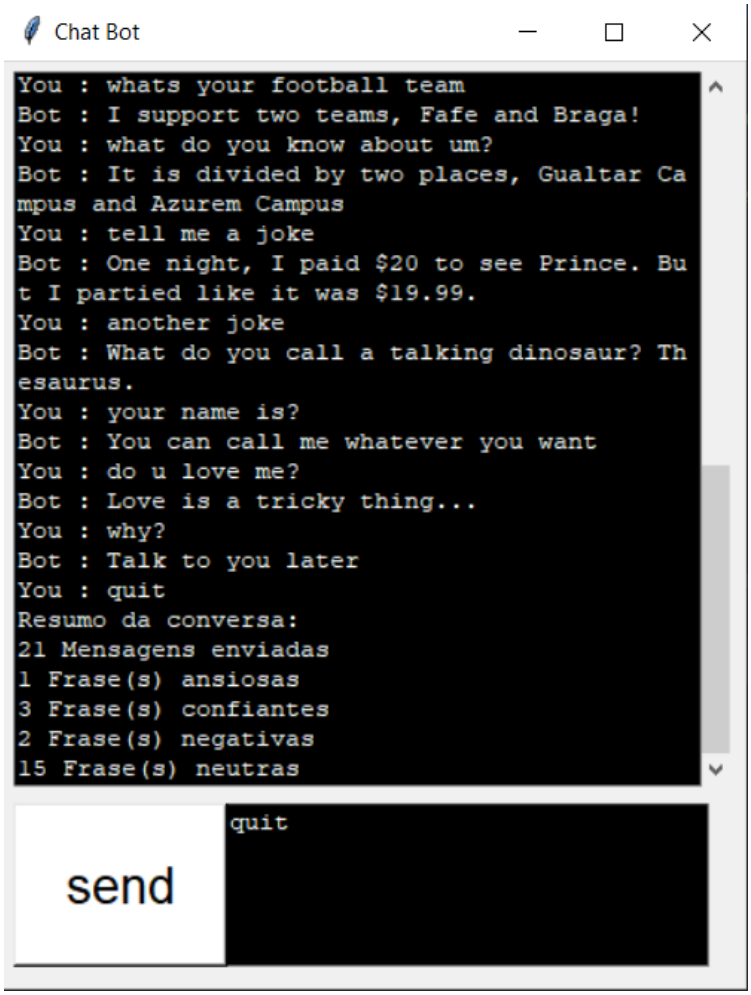

Figura 16: GUI a funcionar