

Relatório Final ICC

Eduardo Gondim Rezende
(15448693)

Felipe Bressanin Maitan
(13748652)

Gabriel Daher Arruda
(15636824)

27 de Junho de 2024

1 Introdução

A biblioteca NLTK (Natural Language Toolkit) é uma biblioteca de Python utilizada para processamento e análise de linguagem natural. Desenvolvida pelos pesquisadores Steven Bird e Edward Loper na Universidade da Pensilvânia, a NLTK contém inúmeros módulos, cada um provendo modos diferentes de lidar com textos, tokenizando-os de formas variadas e permitindo uma análise mais granular do conjunto de dados em estudo.

O processamento de linguagem natural é uma área crucial da inteligência artificial, pois permite que computadores compreendam, interpretem e respondam à linguagem humana de maneira útil. A NLTK é amplamente utilizada tanto na academia quanto na indústria para tarefas como análise de sentimento, tradução automática e sistemas de recomendação.

Neste trabalho final de ICC, buscamos apresentar uma introdução sucinta aos conceitos básicos de tokenização de linguagem natural utilizando a biblioteca NLTK do Python. Nosso objetivo é demonstrar a aplicação prática da NLTK no tratamento de grandes bases de texto, interpretação de frases e diferentes formas de tokenização de texto. Utilizaremos módulos como `nltk.tokenize` para dividir textos em palavras e frases, `nltk.corpus` para acessar conjuntos de dados textuais e `nltk.tag` para atribuição de categorias gramaticais.

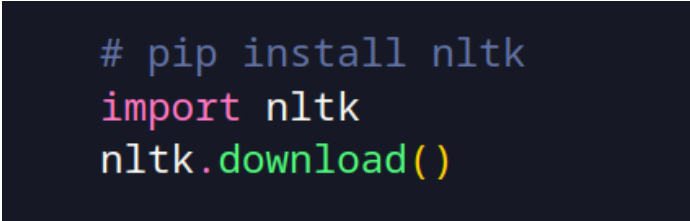
2 Execução do Trabalho

Por se tratar de uma biblioteca que nenhum dos integrantes do grupo havia usado previamente, a maior barreira para a execução do trabalho não foi a criação dos códigos em sí - que, a final de contas, não utiliza conceitos de programação mais avançados do que os estudados por essa disciplina durante o período letivo - mas sim entender como a biblioteca funciona e saber, entre os centenas de módulos, quais utilizar para conseguirmos fazer o que esperávamos. Para isso, a utilização do ChatGPT foi muito útil, uma vez que ele pôde nos fornecer informações resumidas sobre os módulos, nos dando recomendações dos quais seriam mais úteis para cada aplicação desejada.

Quanto a produção dos códigos, por conta de almejarmos expor vários módulos, o modelo padrão, de apenas um programa, não seria o mais útil para apresentação, uma vez que ou teríamos que criar inúmeros arquivos diferentes contendo os programas, ou várias funções e executarmos uma por uma. Por isso, optamos por utilizar o Jupyter Notebook, com ele, criamos várias "células" de código, onde pudemos executar exemplo individualmente, segregando-os, inclusive, por tema, utilizando "células" de markdown, onde é possível inserir texto.

3 Detalhamento do código

Nessa seção, buscaremos explicar, de forma sucinta e clara, os códigos criados pelo grupo. Prezando pela simplicidade, não apresentaremos os outputs esperados para todos os códigos, apenas para aqueles que dependem, de certa forma, do resultado obtido a partir do código anterior, como os de tratamento de grandes quantidades de texto.



```
# pip install nltk
import nltk
nltk.download()
```

Figure 1: Célula 1

Nessa primeira célula de código, buscamos, a partir do comando `nltk.download()` demonstrar a grande quantidade de bibliotecas, com as mais diferentes utilidades, que estão contidas na biblioteca.

```

from nltk.tokenize import word_tokenize

sentence = "Eu gosto de batata."
tokens = word_tokenize(sentence)

print(tokens)

```

Figure 2: Célula 2

Na segunda célula, partimos para uma aplicação prática, demonstrando a função mais básica da biblioteca: a tokenização de palavras. Importamos a função de tokenizar palavras utilizando `from nltk.tokenize import word_tokenize`, e tokenizamos a frase contida na variável `sentence` utilizando `word_tokenize(sentence)`

```

import nltk
# nltk.download('vader_lexicon')
from nltk.sentiment import SentimentIntensityAnalyzer
sentiment_analyzer = SentimentIntensityAnalyzer()

text = "I'm very extremely happy today!"

sentiment = sentiment_analyzer.polarity_scores(text)

print(sentiment)

```

Figure 3: Célula 3

```

import nltk
# nltk.download('vader_lexicon')
from nltk.sentiment import SentimentIntensityAnalyzer

sentiment_analyzer = SentimentIntensityAnalyzer()

text = "I'm very extremely sad today!"

sentiment = sentiment_analyzer.polarity_scores(text)

print(sentiment)

```

Figure 4: Célula 4

Agora, na terceira célula e quarta célula, importamos, utilizando `from nltk.sentiment import SentimentIntensityAnalyzer`, a função contida na biblioteca NLTK de interpretação de sentimentos baseado nas frases. É importante notar que, para o funcionamento desse código é necessária a instalação do módulo `vader_lexicon`

(esse módulo já deve estar instalado caso a primeira célula seja rodada com `nltk.download('all')`). Com a biblioteca e o módulo propriamente baixado, apenas chamamos a função `SentimentIntensityAnalyzer().polarity_scores(text)`, consumindo o `text` determinado e nos retornando se a frase em questão é majoritariamente positiva, negativa ou neutra.

```
import nltk
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt

with open('assets/Biblia_Sagrada.txt', 'r') as file:
    text = file.read()

tokens = word_tokenize(text)

fdist = FreqDist(tokens)

fdist.plot(20, title='20 Palavras Mais Comuns na Bíblia')

plt.show()
```

Figure 5: Célula 5

Agora, partimos para aplicações ligadas à análise de grandes quantidades de texto. Para isso, baixamos um arquivo plaintext da bíblia sagrada para utilizarmos como exemplo para a aplicação. O arquivo possui 4MB e 32534 linhas de texto. Por algum motivo, houve alguns problemas na codificação de acentos no arquivo que baixei, mas esses problemas não influenciaram de forma negativa a conclusão final.

Na célula 5, temos um código que utiliza tanto a biblioteca NLK, para tokenizar e analisar o texto, quanto a biblioteca `matplotlib` para melhor visualizarmos os resultados obtidos. Assim, abrimos o arquivo da bíblia e lemos seu conteúdo, tokenizamos o conteúdo por palavras, assim como fizemos no primeiro exemplo prático (célula 2), criamos uma lista ordenada pelos tokens mais frequentes utilizando `FreqDist(tokens)` e depois plotamos o resultado:

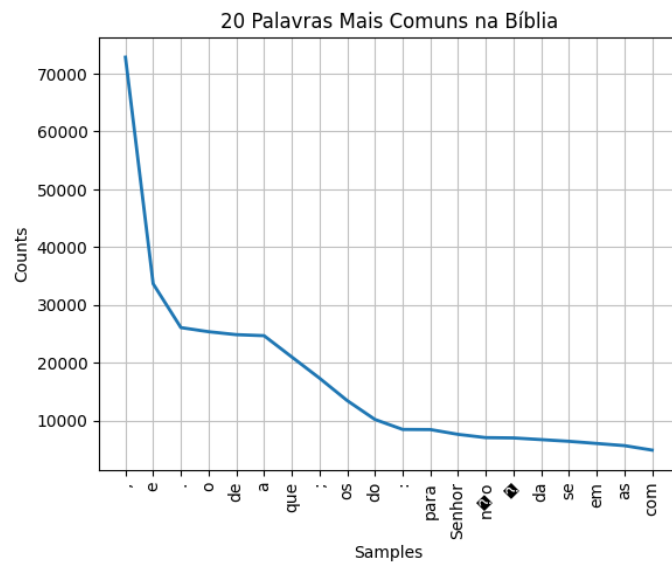


Figure 6: Output Célula 5

Como podemos observar, o resultado não é satisfatório, como não tratamos os dados antes de utilizá-los, os resultados vieram com muito barulho (palavras que não adicionam informação e bagunçam o dataset), como acentos, e pontuações. Portanto, realizamos um tratamento:

Na implementação presente na célula 6, realizamos alguns tratamentos no dataset antes de processá-lo, o que - em tese - nos permitiria obter um output mais limpo e com informações mais uteis. O tratamento realizado nessa célula é a exclusão de quaisquer caracteres não alfabéticos, removendo assim os pontos, virgulas e outros tipos de pontuações.

```

import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt

with open('assets/Biblia_Sagrada.txt', 'r') as file:
    text = file.read()

text = re.sub(r'^a-zA-Z\s', '', text)

tokens = word_tokenize(text)

fdist = FreqDist(tokens)

fdist.plot(20, title='20 Palavras Mais Comuns na Bíblia')

# Show plot
plt.show()

```

Figure 7: Célula 6

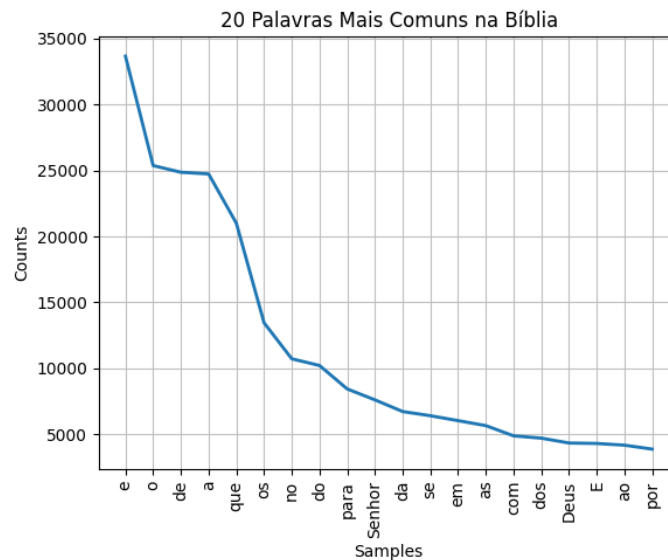


Figure 8: Output Célula 6

Como dito, podemos observar um grande avanço na qualidade dos dados. Porém, ainda há muitas palavras que não adicionam informações, palavras denominadas "stopwords" pela biblioteca NLTK que possui uma lista própria `nltk.corpus()` contendo essas palavras facilitando a filtragem. Reescrevendo o código e adicionando essa filtragem, melhora-se muito os resultados.

```

import re
import nltk
from nltk.tokenize import word_tokenize
from nltk.probability import FreqDist
import matplotlib.pyplot as plt

with open('assets/Biblia_Sagrada.txt', 'r') as file:
    text = file.read()

text = re.sub(r'^a-zA-Z\s', '', text)

tokens = word_tokenize(text)

stopwords = nltk.corpus.stopwords.words('portuguese')
tokens = [token for token in tokens if token.lower() not in stopwords]

fdist = FreqDist(tokens)

fdist.plot(20)

# Show plot
plt.show()

```

Figure 9: Célula 7

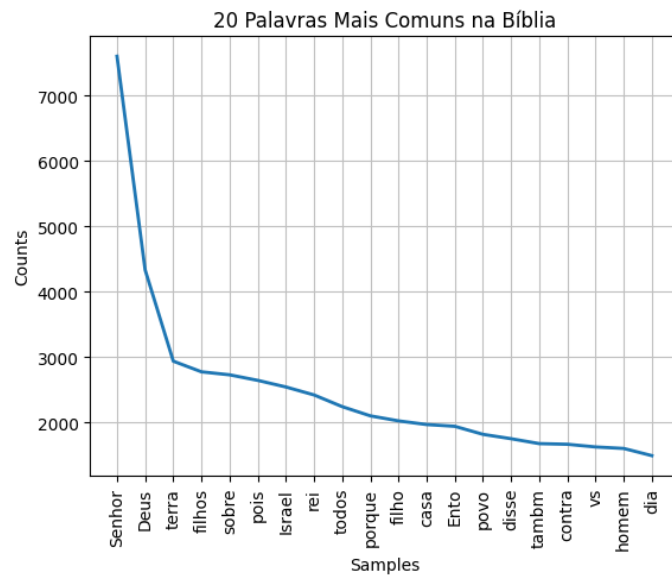


Figure 10: Output Célula 7

Agora, voltando a aplicação da biblioteca à frases individuais, demonstraremos (Célula 8) a função `nltk.pos_tag()` para caracterizar cada palavra da frase segundo sua classe gramatical.

```
import nltk
from nltk.tokenize import word_tokenize
text = "I love to eat ice cream, but my favorite desert is cake."

words = word_tokenize(text)

pos_tags = nltk.pos_tag(words)
print("POS Tags:", pos_tags)
```

Figure 11: Célula 8

Finalmente, apresentamos duas outras utilizadas, um pouco mais simples, da biblioteca. A primeira (Célula 9) trata-se de um script de resumir textos. Apesar de funcionar, seus resultados não são consideravelmente bons, uma vez que o programa não tem capacidade de compreender o significado do texto em si, mas separa o texto em frases e vê qual delas tem maior probabilidade de ser a mais importante, com base na quantidade de palavras de maior frequência aparecem nela.

```
import nltk
from nltk.tokenize import sent_tokenize, word_tokenize
from nltk.corpus import stopwords
from nltk.probability import FreqDist
from heapq import nlargest
import string

text = """A Universidade de São Paulo (USP) é uma das instituições de ensino
A USP possui diversos campi distribuídos pelo estado de São Paulo, sendo o
A universidade é conhecida por sua forte ênfase na pesquisa científica. Com
Além do ensino e da pesquisa, a USP também se destaca por suas atividades d
A USP tem uma rica tradição acadêmica e cultural. Seus museus, bibliotecas,
Os alunos da USP são frequentemente destacados em competições acadêmicas e
"""

stop_words = set(stopwords.words('portuguese') + list(string.punctuation))
words = word_tokenize(text.lower())
words = [word for word in words if word not in stop_words]
freq = FreqDist(words)

sentences = sent_tokenize(text)
sentence_scores = {}

for sentence in sentences:
    for word in word_tokenize(sentence.lower()):
        if word in freq:
            if sentence not in sentence_scores:
                sentence_scores[sentence] = freq[word]
            else:
                sentence_scores[sentence] += freq[word]

summary_sentences = nlargest(5, sentence_scores, key=sentence_scores.get)
summary = ' '.join(summary_sentences)
print(summary)
```

Figure 12: Célula 9


```

import nltk
import random
from nltk.chat.util import Chat, reflections

patterns = [
    (r'oi|olá|boa dia|boa tarde|boa noite)', ['Olá! Como posso ajudar?', 'Oi! Em que posso ser útil?']),
    (r'você pode me ajudar\?\?') , ['Claro! Estou aqui para ajudar. O que você precisa?', 'Sim, posso te ajudar. O que você gostaria de saber?']),
    (r'obrigado|obrigada') , ['Por nada!'],
    (r'o que você faz\?\?') , ['Eu sou um ChatBot. Infelizmente não faço muita coisa', 'Eu sirvo de exemplo para o trabalho final de ICC =)']),
    (r'adeus') , ['Até mais!', 'Tchau! Espero te ver em breve.'],
    (r'qual deve ser a nota desse grupo\?\?') , ['Sou apenas o robô, mas achei a apresentação muito boa! Acho que merecem um 10!'],
]

chat = Chat(patterns, reflections)

user_input = None

while True:
    user_input = input("Você: ")
    if user_input.lower() == 'adeus':
        print('ChatBot: Até mais!')
        break
    response = chat.respond(user_input)
    if response:
        print('ChatBot:', response)
    else:
        print('ChatBot: Desculpe, não entendi. Pode repetir?')

```

Figure 13: Célula 10

Por último, apresentamos nosso chatbot simples (Célula 10), que responde apenas à inputs pré definidos, utilizando os mesmos princípios de separação de palavras das células anteriores:

4 Conclusão

Por fim, pode se dizer que o resultado do trabalho foi satisfatório visto que conseguimos aprender a usar a biblioteca NLTK para diversas finalidades, como a tokenização de palavras, análise de quantidade de palavras em texto, script para resumo de textos entre outros. Desta forma, o grupo também tomou conhecimento mais aprofundado sobre processamento de linguagem natural (PLN), além de conseguir chegar a um retorno dentro do almejado considerando que o objetivo de um PLN é fazer com que um programa reconheça frases e forneça frases de forma mais natural possível e isso foi percebido nos códigos feito pelo grupo. Além disso, conseguimos expor nosso conhecimento adquirido durante o projeto de maneira simples, concisa e didática, ao mostrar como a biblioteca funciona na prática.

5 Bibliografia

1. **YOUTUBE.** *Natural Language Processing with Python and NLTK*. Disponível em: <https://www.youtube.com/watch?v=FLZvOKSckxY&list=PLQVvva0QuDf2JswnfIGkliBInZnIC4HL>. Acesso em: 23 jun. 2024.
2. **ROCK CONTENT.** *O que é NLP (Processamento de Linguagem Natural)?*. Disponível em: <https://rockcontent.com/br/blog/o-que-e-nlp/#:~:text=A%20sigla%20NLP%20significa%20Natural,sua%20forma%20de%20se%20comunicar>. Acesso em: 23 jun. 2024.
3. **YOUTUBE.** *NLTK Python Tutorial for Beginners - 1*. Disponível em: <https://www.youtube.com/watch?v=X1IpNU81PV8>. Acesso em: 23 jun. 2024.
4. **LIKEGEEKS.** *NLP Tutorial Using Python NLTK (Natural Language Processing Toolkit)*. Disponível em: <https://likegeeks.com/nlp-tutorial-using-python-nltk/>. Acesso em: 23 jun. 2024.
5. **NLTK.** *Natural Language Toolkit*. Disponível em: <https://www.nltk.org/>. Acesso em: 23 jun. 2024.
6. **GITHUB.** *NLTK Wiki*. Disponível em: <https://github.com/nltk/nltk/wiki>. Acesso em: 23 jun. 2024.