NATURAL LANGUAGE PROCESSING

AULA 5

Prof. Luciano Frontino de Medeiros

CONVERSA INICIAL

Esta aula tem como objetivo apresentar a forma introdutória da linguagem Python com a biblioteca NLTK (*Natural Language Toolkit*), que é amplamente utilizada para a criação de códigos e scripts em tarefas de processamento de linguagem natural. Algumas tarefas de NLP são abordadas, tais como a tokenização, a derivação (*stemming*), a segmentação e a lematização. A classificação e a avaliação de documentos são tarefas bastante relevantes, sendo explanados os aspectos envolvendo o processo genérico de classificação supervisionada. Por fim, um exemplo de classificação de documentos é apresentado, de maneira a ilustrar de forma prática como uma classificação pode ser modelada e executada dentro de um ambiente de programação Python.

TEMA 1 – A LINGUAGEM PYTHON

A linguagem Python tem se tornado uma referência em aplicações que utilizam Inteligência Artificial e Machine Learning, devido ao alto nível de interatividade e flexibilidade que oferece na construção de modelos. Apesar de ter se popularizado nos últimos anos, a linguagem Python surgiu em 1991. Como uma linguagem aberta, diferente de outras linguagens como C e Java, ela é **interpretada**: o comando do usuário é executado logo após ser digitado no software interpretador, sem necessidade de compilação do código.

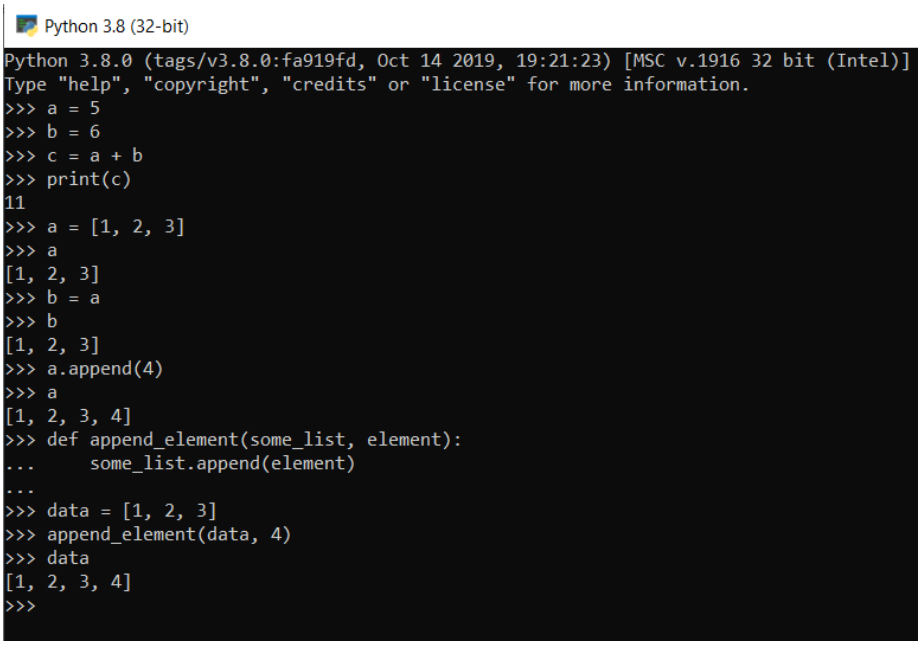
Python possui uma implementação básica de comandos de programação. Mas a linguagem é enriquecida por várias bibliotecas ou API (criadas por desenvolvedores que formam uma grande comunidade ativa de processamento científico e análise de dados). Dessa forma, além das bibliotecas padrão, outras bibliotecas podem ser instaladas de acordo com a necessidade de processamento do usuário (Müller; Guido, 2015).

A linguagem Python implementa estruturas de dados de alto nível, orientadas para o processamento eficiente dos dados e possui ainda uma implementação simples, porém eficaz de programação orientada a objetos. A partir de uma sintaxe elegante e com menos símbolos, a linguagem é ideal para a execução de pequenos e rápidos códigos para atingir objetivos específicos (*scripts*). Python foi uma linguagem cuja sintaxe foi se modificando ao longo de versões. A versão atual é a 3.X. Ainda que seja possível desenvolver scripts na

versão 2.X, tal procedimento não é recomendável por não haver retrocompatibilidade.

O interpretador Python permite a execução do *shell*, por meio do qual é possível inserir comandos (Figura 1). Os comandos podem ser fornecidos na própria tela, com o resultado sendo apresentado logo em seguida. Entretanto, para tarefas mais complexas de análise, é interessante editar os comandos em um arquivo com extensão *.py* (Kopec, 2019).

Figura 1 – Shell do Python 3.8



Para uma abordagem em alto nível, Python também pode ser utilizado a partir de IDEs desenvolvidas para aumentar a produtividade. Algumas IDEs mais utilizadas são PyCharm, PyDev, Pyhton Tools para Visual Studio, Spyder (Anaconda) e Komodo. Algumas IDEs permitem a instalação de plugins que permitem a customização das plataformas.

O poder da linguagem Python provém em grande parte da instalação de bibliotecas para o processamento eficiente de dados e execução de diversas tarefas. Algumas bibliotecas mandatórias para a maioria dos processos relacionados com ciência de dados e inteligência artificial são:

- **Numpy** – nome proveniente da abreviatura de *Numerical Python*. Constitui na pedra angular do processamento numérico em Python. Oferece funções para efetuar processamentos em todos os elementos de *arrays* ou operações matemáticas entre *arrays*. Possui ainda ferramentas

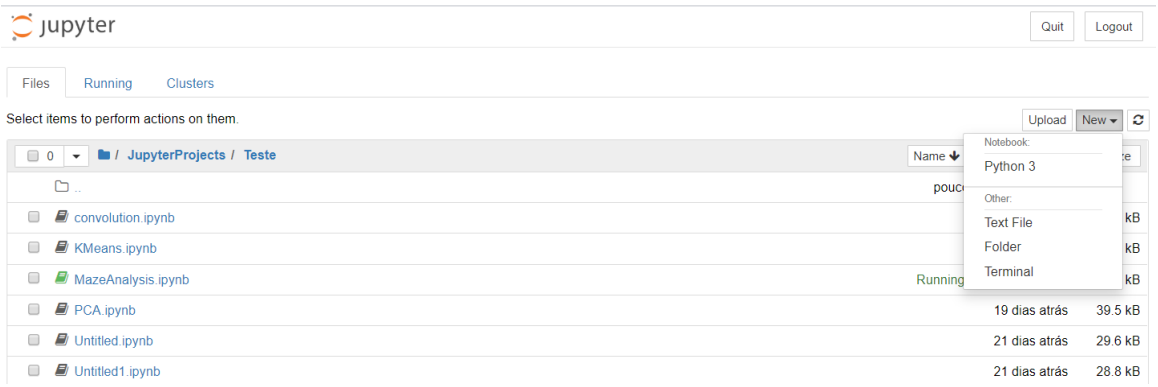
para ler e gravar conjuntos de dados baseados em arrays de disco. A biblioteca Numpy tira proveito do hardware instalado (GPU, processamento paralelo etc.) para maximizar a sua performance.

- **Pandas** – de forma complementar ao numpy, a biblioteca Pandas oferece estruturas de dados de alto nível e funções projetadas para dados estruturados ou tabulares, de forma semelhante a tabelas de banco de dados. Duas estruturas de dados muito utilizadas da biblioteca Pandas são o *DataFrame* (estrutura tabular orientada a colunas, com rótulos tanto para linhas quanto para colunas) e *Series* (objeto array unidimensional com rótulo).
- **Matplotlib** – uma das bibliotecas mais populares para a construção de gráficos e plotagens e que gera ainda diversas visualizações de dados bidimensionais. Ela foi projetada para criar plotagens apropriadas para publicação de dados. Ela permite a integração direta com formatos de dados de outras bibliotecas para visualização dos dados.
- **Scipy** – uma coleção de pacotes voltada para uma série de diversos domínios de problemas padrões em processamento científico. Possui um ecossistema de software *open source* para matemática, ciências e engenharia.
- **Scikit-learn** – é a principal “caixa” de ferramentas de propósito geral para aprendizado de máquina dos programadores Python. Ela inclui vários submódulos próprios para as tarefas de preparação de dados em ciência de dados e inteligência artificial: classificação, regressão, agrupamento, pré-processamento, redução de dimensionalidade e seleção de modelos.
- **Statsmodels** – consiste em um pacote de análise estatística e contém algoritmos para estatística clássica, modelos de regressão, análise de variância, séries temporais, visualização de dados. Possui um foco maior em inferência estatística.

Uma das ferramentas mais práticas para a execução de scripts em Python é **Jupyter Notebook**. O Jupyter Notebook é um aplicativo da web de código aberto que permite criar e compartilhar *notebooks*: documentos que contêm código ativo, equações, visualizações e texto narrativo. Seu uso inclui: limpeza e transformação de dados, simulação numérica, modelagem estatística,

visualização de dados, aprendizado de máquina e muito mais. Como plataforma de execução para Python, grandes empresas oferecem ambientes para o desenvolvimento no formato *notebook*, como o Google Colaboratory[1]. Entretanto, podem ser instalados pelo usuário em ambiente local, utilizando ferramentas como Anaconda Navigator.

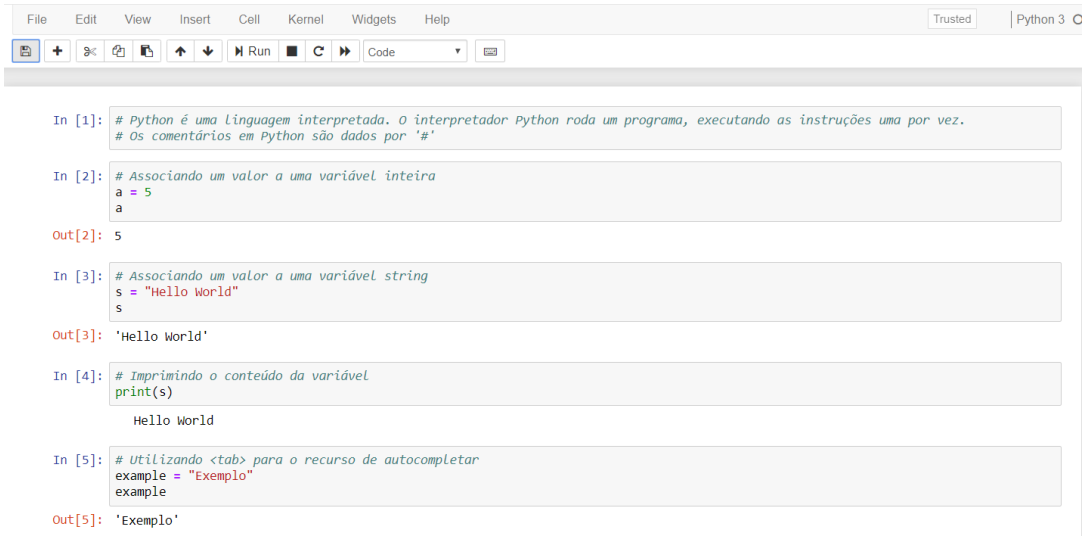
Figura 2 – Ambiente do Jupyter Notebook, mostrando uma pasta específica com arquivos de scripts



Jupyter Notebook implementa uma maneira diferenciada de execução de scripts. Cada script pode ser colocado em uma **célula de entrada de dados**, com o resultado da sua execução em uma célula de saída apresentada logo a seguir (Figura 3). O arquivo com diversos scripts é salvo no formato *.ipynb*, podendo ser recuperado mais tarde com a mesma configuração da última execução efetuada.

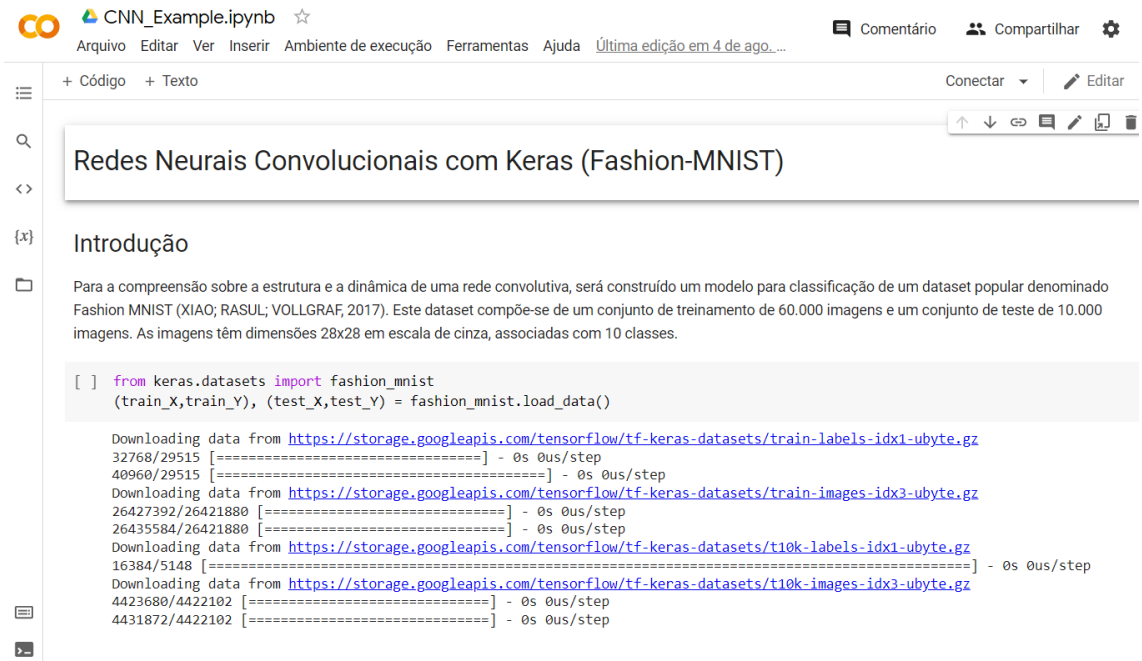
[1] Disponível em: <https://colab.research.google.com/notebooks/intro.ipynb?utm_source=scs-index>.

Figura 3 – Exemplo de um script em Jupyter Notebook, mostrando diversas células de entrada de dados e a execução de cada uma logo abaixo



Na figura 4 está o exemplo de uma tela do Google Colaboratory (ou como é referenciado de forma mais sucinta, Google Colab). Os códigos são colocados no mesmo formato de scripts (notebook) conforme visto no Jupyter Notebook, sendo que os arquivos de notebooks podem ser colocados na nuvem (Google Drive) para acesso, mediante o cadastro de uma conta respectivamente.

Figura 4 – Tela demonstrando o uso do Google Colaboratory



Fonte: Medeiros, 2022.

A forma de trabalho em notebooks, seja no Jupyter Notebook ou no Google Colaboratory, permite um modo bastante interativo de desenvolvimento de código. Os scripts colocados em células são executados individualmente, e cada script executado disponibiliza os dados na memória para serem utilizados pelos próximos scripts, sem a necessidade de executar códigos de carga dos dados, limpeza ou transformação. Por um processo de tentativa e erro, os scripts podem ser aprimorados até que se alcance uma versão que seja a desejada ou a melhor possível.

TEMA 2 – A BIBLIOTECA NLTK

NLTK (*Natural Language Toolkit*) é uma biblioteca para uso em tarefas de linguagem natural na linguagem Python. NLTK define uma infraestrutura que pode ser usada para construir programas de linguagem natural em Python. Ela fornece classes básicas para representar dados relevantes para processamento de linguagem natural; interfaces padrão para executar tarefas como marcação de parte da fala, análise sintática e classificação de texto; e implementações padrão para cada tarefa que podem ser combinadas para resolver problemas complexos (Bird; Klein; Loper, 2019).

A biblioteca NLTK assume que o usuário esteja utilizando a versão Python 3.2 ou superior (ainda que a biblioteca rode nas versões obsoletas 2.6 e 2.7). Foi originalmente criada em 2001 como parte de um curso de linguística computacional no Departamento de Ciência da Computação e da Informação da Universidade da Pensilvânia. Desde então, foi desenvolvida e expandida com a ajuda de dezenas de colaboradores. Já foi adotada em cursos em dezenas de universidades e serve de base para muitos projetos de pesquisa.

Tabela 1 – Tarefas de NLP, os módulos da NLTK respectivos e suas funcionalidades

Tarefa de NLP	**Módulos da NLTK**	**Funcionalidade**
Acessando corpora	Corpus	Interfaces padronizadas para corpora e léxicos
Processamento de strings	tokenize, stem	Tokenizers, tokenizers de sentenças, lematizadores

Descoberta de colocação	collocations	Teste t, qui-quadrado, informações mútuas pontuais
Marcação de parte do discurso (Part-of-speech tagging)	Tag	n-gram, backoff, Brill, HMM, TnT
Machine learning	classify, cluster, tbl	Árvore de decisão, entropia máxima, Bayes ingênuo, EM, k-means
Chunking	Chunk	Expressão regular, n-gram, entidade nomeada
Parsing	parse, ccg	Gráfico, baseado em recursos, unificação, probabilístico, dependência
Interpretação semântica	sem, inference	Cálculo lambda, lógica de primeira ordem, verificação de modelo
Métricas de avaliação	Metrics	Precisão, recall, coeficientes de concordância
Probabilidade e estimativa	probability	Distribuições de frequência, distribuições de probabilidade suavizadas
Formulários	app, chat	Analisadores, navegador WordNet, chatbots
Trabalho de campo linguístico	Toolbox	Manipular dados no formato SIL Toolbox

Fonte: Elaborado com base em Bird *et al*.,2019.

Na tabela 1 estão descritos os principais módulos da NLTK relacionados com as tarefas de linguagem natural e as funcionalidades respectivas. A biblioteca NLTK foi desenvolvida tendo quatro objetivos primários em mente.

- **Simplicidade** – para fornecer uma estrutura intuitiva com blocos de construção substanciais, dando aos usuários um conhecimento prático de PNL sem ficar atolado na tediosa manutenção geralmente associada ao processamento de dados de linguagem anotados.
- **Consistência** – para fornecer uma estrutura uniforme com interfaces e estruturas de dados consistentes e nomes de métodos fáceis de adivinhar.

- **Extensibilidade** – fornecer uma estrutura na qual novos módulos de software possam ser facilmente acomodados, incluindo implementações alternativas e abordagens concorrentes para a mesma tarefa.
- **Modularidade** – para fornecer componentes que podem ser usados de forma independente, sem a necessidade de entender o restante do kit de ferramentas.

Portanto, a biblioteca NLTK consiste em uma ferramenta indispensável tanto para o aprendizado quanto para o desenvolvimento de soluções em linguagem natural, utilizando a linguagem Python.

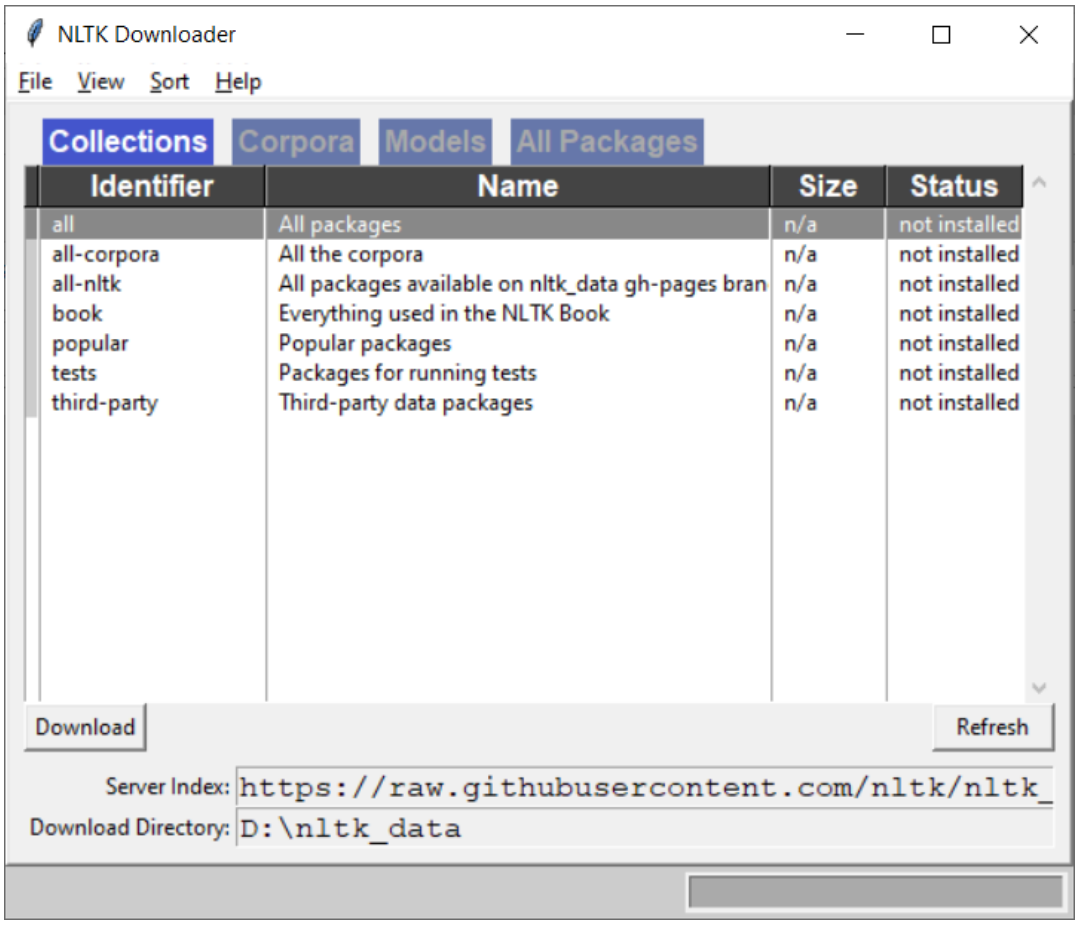
2.1 Primeiros comandos

Para instalar a biblioteca NLTK em um notebook, deve-se inserir o seguinte script:

```
import nltk
nltk.download()
```

A partir da importação da biblioteca NLTK, executa-se o NLTK Downloader (figura a seguir). Esse aplicativo permite tanto a instalação quanto a atualização das coleções, corpora e modelos. É importante o usuário verificar se uma coleção, corpora ou modelo específico esteja instalado previamente, para que depois possa seguir com a execução dos scripts desejados. Após a instalação/atualização, clica-se em “Exit” para sair do aplicativo e retornar ao notebook (o qual retorna uma célula de saída com o valor booleano “True”).

Figura 5 – Instalação a partir do NLTK Downloader



Fonte: Medeiros, 2022.

Depois do download, pode-se carregar alguns livros que constam na NLTK para uso como exemplos com o interpretador Python. O primeiro passo é digitar um comando especial no prompt do Python que diz ao interpretador para carregar alguns textos para explorarmos:

```
from nltk.book import *
```

Esse script pode ser interpretado como "no módulo de livros do NLTK, carregue todos os livros disponíveis". Após imprimir uma mensagem de boas-vindas, ele carrega o texto de vários livros (isso pode levar alguns segundos):

```
from nltk.book import *
```

```
*** Introductory Examples for the NLTK Book ***
Loading text1, ..., text9 and sent1, ..., sent9
Type the name of the text or sentence to view it.
Type: 'texts()' or 'sents()' to list the materials.
text1: Moby Dick by Herman Melville 1851
text2: Sense and Sensibility by Jane Austen 1811
text3: The Book of Genesis
text4: Inaugural Address Corpus
text5: Chat Corpus
text6: Monty Python and the Holy Grail
text7: Wall Street Journal
text8: Personals Corpus
text9: The Man Who Was Thursday by G . K . Chesterton 1908
```

Pode-se escolher algum texto especificamente para execução de algum script. Por exemplo, digitando “text2”, o livro correspondente é mostrado:

```
text2
```

```
<Text: Sense and Sensibility by Jane Austen 1811>
```

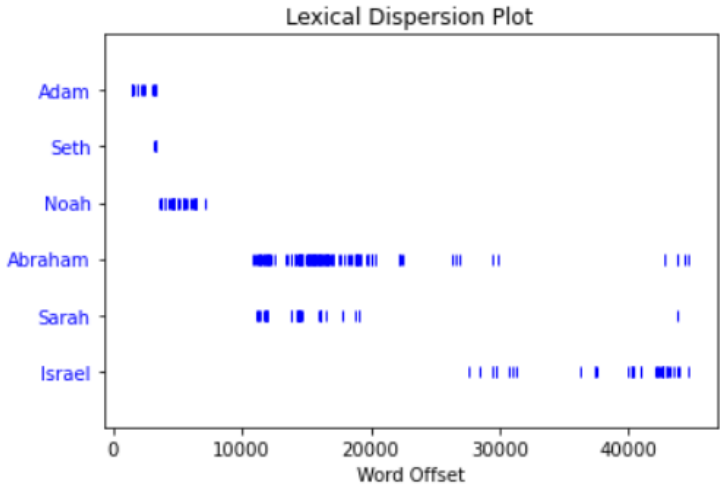
Para buscar um termo específico dentro do texto (por exemplo, a palavra “gentle”), usa-se o método “concordance” com a palavra colocada como argumento, sendo mostrado a seguir as ocorrências da palavra buscada, dentro do livro escolhido:

```
text2.concordance("gentle")
```

```
Displaying 8 of 8 matches:
 well - bred , well - informed , of gentle address , and , I believe , possess
lf with merely giving her husband a gentle reprimand on the subject five or si
wo minutes ; " And she is always so gentle and quiet -- Never was there such a
o . Her sister ' s earnest , though gentle persuasion , however , soon softene
 the means of this continual though gentle vent , was able not only to see the
r own -- her own happy thoughts and gentle diction . But what could I do !-- w
e spontaneous , more natural , more gentle , less dignified , forgiveness . Te
 in his happiness -- and added some gentle counsel as to the behaviour most li
```

Assim, é possível detectar automaticamente uma determinada palavra ocorrendo em um texto e dessa forma exibir algumas palavras que aparecem no mesmo contexto. No entanto, também é possível determinar a localização de uma palavra no texto: quantas palavras desde o início ela aparece. Essas informações posicionais podem ser exibidas usando um **gráfico de dispersão lexical**. Cada faixa representa uma instância de uma palavra e cada linha representa todo o texto:

```
text3.dispersion_plot(["Adam", "Seth", "Noah", "Abraham", "Sarah", "Israel"])
```



Pode-se descobrir o comprimento de um texto do início ao fim, em termos de palavras e símbolos de pontuação que aparecem. Usamos o termo “len” para obter o comprimento de algo, que aplicaremos aqui ao livro de Gênesis:

```
len(text3)
```

```
44764
```

Portanto, o livro do Gênesis tem 44.764 palavras e símbolos de pontuação, ou ***tokens***. Um *token* é o nome técnico de uma sequência de caracteres, tais como “ele”, “alegre” ou “:)”, das quais se deseja tratar como um grupo. Quando contamos o número de *tokens* em um texto, digamos, a frase “ser ou não ser”, estamos contando ocorrências dessas sequências. Assim, na frase de exemplo, há duas ocorrências de “ser”, uma de “ou” e uma de “não”. Mas há apenas três itens de vocabulário distintos nessa frase. Quantas palavras distintas contém o livro de Gênesis? Para resolver isso em Python, temos que colocar a questão de forma um pouco diferente. O vocabulário de um texto é apenas o conjunto de *tokens* que ele usa, pois em um conjunto, todas as duplicatas são recolhidas juntas. Em Python, é possível obter os itens de vocabulário do livro do Gênesis com o comando:

```
print(sorted(set(text3)))
```

Com o seguinte resultado:

```
['!', "'", '(', ')', ',', ',)', '.', '.)', ':', ';', ';)', '?', '?)', 'A', 'A
bel', 'Abelmizraim', 'Abidah', 'Abide', 'Abimael', 'Abimelech', 'Abr', 'Abra
h', 'Abraham', 'Abram', 'Accad', 'Achbor', 'Adah', 'Adam', 'Adbeel', 'Admah',
'Adullamite', 'After', 'Aholibamah', 'Ahuzzath', 'Ajah', 'Akan', 'All', 'Allo
nbachuth', 'Almighty', 'Almodad', 'Also', 'Alvah', 'Alvan', 'Am', 'Amal', 'Am
alek', 'Amalekites', 'Ammon', 'Amorite', 'Amorites', 'Amraphel', 'An', 'Ana
h', 'Anamim', 'And', 'Aner', 'Angel', 'Appoint', 'Aram', 'Aran', 'Ararat', 'A
rbah', 'Ard', 'Are', 'Areli', 'Arioch', 'Arise', 'Arkite', 'Arodi', 'Arphaxa
d', 'Art', 'Arvadite', 'As', 'Asenath', 'Ashbel', 'Asher', 'Ashkenaz', 'Ashte
roth', 'Ask', 'Asshur', 'Asshurim', 'Assyr', 'Assyria', 'At', 'Atad', 'Avit
h', 'Baalhanan', 'Babel', 'Bashemath', 'Be', 'Because', 'Becher', 'Bedad', 'B
eeri', 'Beerlahairoi', 'Beersheba', 'Behold', 'Bela', 'Belah', 'Benam', 'Benj
amin', 'Beno', 'Beor', 'Bera', 'Bered', 'Beriah', 'Bethel', 'Bethlehem', 'Bet
```

Com esse comando, o comprimento do conjunto de *tokens* agora considera todos os *tokens* sem repetição (devido ao uso de conjuntos – set). A contagem de *tokens* agora será bem menor que o total de palavras no livro:

```
len(set(text3))
```

```
2789
```

A partir de agora, algumas tarefas específicas de processamento de linguagem natural podem ser estudadas a partir de implementações de scripts em Python.

TEMA 3 – TAREFAS: TOKENIZAÇÃO, DERIVAÇÃO, SEGMENTAÇÃO, LEMATIZAÇÃO

É comum, na execução de tarefas e avaliação de scripts em NLP, utilizar textos originários da web ou mesmo de arquivos instalados localmente. Por exemplo, no caso de se utilizar livros, alguns deles já são instalados a partir da NLTK e disponibilizados pela própria biblioteca. É o caso de livros que fazem parte do Projeto Gutenberg. No entanto, pode-se analisar outros textos do Projeto Gutenberg, que apresenta um catálogo de 25.000 livros online gratuitos e assim obter uma URL para um arquivo de texto ASCII. Embora 90% dos textos do Projeto Gutenberg sejam em inglês, ele inclui material em mais de 50 outros idiomas, incluindo catalão, chinês, holandês, finlandês, francês, alemão, italiano, português e espanhol, com mais de 100 textos cada (Bird; Klein; Loper, 2019).

Por exemplo, caso se queira baixar o livro “Memórias Póstumas de Brás Cubas” de Machado de Assis, basta executar o seguinte script, o qual associa à variável “url” o link específico para este livro:

```
from urllib import request
url = "https://www.gutenberg.org/cache/epub/54829/pg54829.txt"
response = request.urlopen(url)
raw = response.read().decode('utf8')
type(raw)
```

```
str
```

Note que é necessário importar o módulo “request” a partir da biblioteca “urllib”. O método “urlopen” é utilizado para abrir o arquivo e depois o comando “read” em conjunto com “decode”, que disponibilizam na variável “raw” o conteúdo do livro.

Verificando o conteúdo da variável “raw”, tem-se:

```
'\ufeffProject Gutenberg\'s Memorias Postumas de Braz Cubas, by Machado de Assis\r\n\r\nThis eBook is for the use of anyone a
nywhere in the United States and most\r\nother parts of the world at no cost and with almost no restrictions\r\nwhatsoever.
You may copy it, give it away or re-use it under the terms of\r\nthe Project Gutenberg License included with this eBook or on
line at\r\nwww.gutenberg.org.  If you are not located in the United States, you\'ll have\r\nto check the laws of the country
where you are located before using this ebook.\r\n\r\nTitle: Memorias Postumas de Braz Cubas\r\n\r\nAuthor: Machado de Assis
\r\n\r\nRelease Date: June 2, 2017 [EBook #54829]\r\n\r\nLanguage: Portuguese\r\n\r\n\r\n*** START OF THIS PROJECT GUTENBERG
EBOOK MEMORIAS POSTUMAS DE BRAZ CUBAS ***\r\n\r\n\r\n\r\n\r\n\r\nProduced by Laura Natal Rodriguez & Marc D\'Hooghe at Free\r\nLi
terature (online soon in an extended version, also linking\r\nto free sources for education worldwide ... MOOC\'s,\r\neducati
onal materials,...)\r\n\r\n\r\n\r\n\r\n\r\n\r\nMEMÓRIAS PÓSTHUMAS\r\n\r\nDE\r\n\r\nBRAZ CUBAS\r\n\r\nPOR\r\n\r\nMACHADO DE ASSIS
\r\n\r\nRIO DE JANEIRO\r\n\r\nTYPOGRAPHIA NACIONAL\r\n\r\n1881\r\n\r\n\r\n\r\n\r\nOBRAS DO AUTOR\r\n\r\n    Memórias Pósthuma
s de Braz Cubas\r\n    Helena, romance\r\n    Yayá Garcia, romance\r\n    Resurreição, romance\r\n    A mão e a luva, romance
\r\n    Historias da meia noite\r\n    Contos Fluminenses\r\n    Americanas, poesias\r\n    Phalenas, poesias\r\n    Chrysali
das, poesias\r\n    Tu só, tu, puro amor, comédia\r\n    Os deuses de casaca, comédia\r\n    Desencantos, comédia\r\n    Thea
tro\r\n\r\n\r\n\r\n\r\n\r\nAO LEITOR\r\n\r\n\r\nQue, no alto do principal de seus livros, confessasse Stendhal havel-o\r\nescript
o para cem leitores, cousa é que admira e consterna. O que não\r\nadmira, nem provavelmente consternará é se este outro livro
não tiver\r\nos cem leitores de Stendhal, nem cincoenta, nem vinte, e quando muito,\r\ndez. Dez? Talvez cinco. Trata-se, na v
erdade, de uma obra diffusa,\r\nna qual eu, Braz Cubas, se adoptei a fórma livre de um Sterne, de\r\num Lamb, ou de um de Mai
stre, não sei se lhe metti algumas rabugens\r\nde pessimismo. Póde ser. Obra de finado. Escrevi-a com a penna da\r\ngalhofa e
a tinta da melancholia; e não é difficil antever o que poderá\r\nsair desse connubio. Accresce que a gente grave achará no li
```

Para a leitura de um arquivo local, utiliza-se as próprias funções embutidas na linguagem Python para operar com arquivos. O seguinte script é equivalente à leitura feita diretamente na Web do script anterior:

```
f = open('pg54829.txt')
```

```
raw = f.read()
```

```
raw
```

```
'Project Gutenberg\'s Memorias Postumas de Braz Cubas, by Machado de Assis\n\nThis eBook is for the use of anyone anywhere in
the United States and most\nother parts of the world at no cost and with almost no restrictions\nwhatsoever.  You may copy i
t, give it away or re-use it under the terms of\nthe Project Gutenberg License included with this eBook or online at\nwww.gut
enberg.org.  If you are not located in the United States, you\'ll have\nto check the laws of the country where you are locate
d before using this ebook.\n\nTitle: Memorias Postumas de Braz Cubas\n\nAuthor: Machado de Assis\n\nRelease Date: June 2, 201
7 [EBook #54829]\n\nLanguage: Portuguese\n\n\n*** START OF THIS PROJECT GUTENBERG EBOOK MEMORIAS POSTUMAS DE BRAZ CUBAS ***\n
\n\n\n\nProduced by Laura Natal Rodriguez & Marc D\'Hooghe at Free\nLiterature (online soon in an extended version, also link
ing\nto free sources for education worldwide ... MOOC\'s,\neducational materials,...)\n\n\n\n\n\nMEMÓRIAS PÓSTHUMAS\n\nDE\n\n
BRAZ CUBAS\n\nPOR\n\nMACHADO DE ASSIS\n\nRIO DE JANEIRO\n\nTYPOGRAPHIA NACIONAL\n\n1881\n\n\n\n\nOBRAS DO AUTOR\n\n    Memóri
as Pósthumas de Braz Cubas\n    Helena, romance\n    Yayá Garcia, romance\n    Resurreição, romance\n    A mão e a luva, roma
nce\n    Historias da meia noite\n    Contos Fluminenses\n    Americanas, poesias\n    Phalenas, poesias\n    Chrysalidas, po
esias\n    Tu só, tu, puro amor, comédia\n    Os deuses de casaca, comédia\n    Desencantos, comédia\n    Theatro\n\n\n\n\nAO
```

3.1 Tokenização

Como sequência do exemplo visto anteriormente, pode-se notar que “raw” é uma variável do tipo *string*, de comprimento 388.217:

```
type(raw)
```

```
str
```

```
len(raw)
```

```
388217
```

Esse é o conteúdo bruto do livro, incluindo muitos detalhes sem interesse como espaços em branco, quebras de linha e linhas em branco. Observe o “\r” e o “\n” na linha de abertura do arquivo, que é como o Python exibe os caracteres especiais de retorno de carro e alimentação de linha (o arquivo foi criado em uma máquina Windows exibindo página de código de caracteres ANSI). Para o processamento de linguagem, deseja-se dividir a *string* em palavras e pontuações, como vimos anteriormente: essa etapa é a **tokenização** e produz uma estrutura contendo uma lista de palavras e pontuação. Isso é feito pelo uso do comando “word_tokenize”:

```
tokens = nltk.word_tokenize(raw)
```

```
len(tokens)
```

```
78647
```

```
print(tokens[:20])
```

```
['Project', 'Gutenberg', "'s", 'Memorias', 'Postumas', 'de', 'Braz', 'Cubas', ',', 'by', 'Machado', 'de', 'Assis', 'This', 'eBo
ok', 'is', 'for', 'the', 'use', 'of']
```

Observe que o NLTK foi necessário para tokenização, mas não para nenhuma das tarefas anteriores de abrir uma URL e lê-la em uma *string*. Em vez de lidar com string, pode-se trabalhar diretamente com uma variável do tipo “texto NLTK”, e assim realizar outros processamentos linguísticos. No texto a seguir, pode-se identificar o uso do método “collocations”, que identifica bigramas (combinações de palavras duas a duas):

Figura 6 – Métodos de *string* em Python

```
text = nltk.Text(tokens)
```

```
type(text)
```

```
nltk.text.Text
```

```
print(text[500:530])
```

```
['processo', 'extraordinario', 'que', 'empreguei', 'na', 'composição', 'destas', '_Memórias_', ',',
'trabalhadas', 'cá', 'no', 'outro', 'seculo', '.', 'Seria', 'curioso', ',', 'mas', 'nimiamente', 'ext
enso', ',', 'e', 'aliás', 'desnecessario', 'ao', 'entendimento', 'da', 'obra', '.']
```

```
text.collocations()
```

```
Quincas Borba; Lobo Neves; Project Gutenberg-tm; alguma cousa; Project
Gutenberg; meu pae; Braz Cubas; United States; não sei; dia seguinte;
Literary Archive; que não; Gutenberg-tm electronic; electronic works;
Archive Foundation; Com effeito; Gutenberg Literary; BRAZ CUBAS; Meu
pae; que era
```

O uso de *strings* permite que se disponha de uma série de métodos embutidos em Python com várias funcionalidades.

Tabela 2 – Métodos e funcionalidades

Método	Funcionalidade
`s.find(t)`	índice da primeira instância da string t dentro de s (-1 se não for encontrado)
`s.rfind(t)`	índice da última instância da string t dentro de s (-1 se não for encontrado)
`s.index(t)`	como `s.find(t)`, porém mostra a exceção `ValueError` caso não encontre
`s.rindex(t)`	como `s.rfind(t)`, porém mostra a exceção `ValueError` caso não encontre
`s.join(text)`	combine as palavras do texto em uma string usando s como cola
`s.split(t)`	divide s em uma lista onde quer que um t seja encontrado (espaço em branco por padrão)
`s.splitlines()`	divide s em uma lista de strings, uma por linha
`s.lower()`	uma versão em letras minúsculas da string s
`s.upper()`	uma versão em letras maiúsculas da string s
`s.title()`	uma versão com título da string s
`s.strip()`	uma cópia de s sem espaços em branco à esquerda ou à direita

Método	Funcionalidade
`s.replace(t, u)`	substitui instâncias de t por u dentro de s

Fonte: Elaborado com base em Bird *et al*, 2019.

3.2 Derivação (Stemming)

Quando se usa um mecanismo de pesquisa na web, geralmente não é importante se as palavras no documento diferem dos termos de pesquisa por possuírem terminações diferentes. Uma consulta sobre “carros” localiza documentos contendo “carro” e “carros”. De fato, “carro” e “carros” são apenas duas formas da mesma palavra de dicionário (ou ainda denominado de “lema”). Para algumas tarefas de processamento de linguagem, geralmente se quer ignorar as terminações das palavras e lidar apenas com as raízes das palavras. Esse é o processo de *stemming* ou **derivação**.

NLTK inclui vários “*stemmers*”, ou seja, funções com módulos específicos para o processo de derivação. No caso da língua portuguesa, existe o *RSLP Portuguese stemmer*. O script a seguir mostra como utilizar o *stemmer* específico para a língua portuguesa e depois fazer o processamento diretamente sobre os *tokens* identificados (apenas 50 *tokens* a partir do *token* de número 500, indicado por “tokens[500:550]”):

```
stemmer = nltk.stem.RSLPStemmer()
```

```
print([stemmer.stem(t) for t in tokens[500:550]])
```

```
['process', 'extraordinari', 'que', 'empreg', 'na', 'compos', 'dest', '_memórias_', ',', 'trabalh',
'cá', 'no', 'outr', 'secul', '.', 'ser', 'curi', ',', 'mas', 'nim', 'extens', ',', 'e', 'aliás', 'des
necessari', 'ao', 'entend', 'da', 'obr', '.', 'a', 'obr', 'em', 'si', 'mesm', 'é', 'tud', ':', 'se',
'te', 'agrad', ',', 'fin', 'leit', ',', 'pago-m', 'da', 'taref', ';', 'se']
```

O processo de derivação também pode ser conseguido diretamente com o uso de expressões regulares (já estudadas anteriormente).

Um processo semelhante à derivação é a **lematização**. A lematização geralmente se refere à análise morfológica das palavras, que visa remover as terminações flexionais. Ajuda a retornar a forma base ou de dicionário de uma palavra conhecida como lema. Por exemplo, o lematizador WordNet só remove os afixos se a palavra resultante estiver em seu dicionário. Esse processo de verificação adicional tende a tornar o algoritmo do lematizador mais lento. A seguir, utilizando-se um conjunto de tokens obtido do livro “Crime and

Punishment” de Fyodor Dostoevsky, gera-se um conjunto de lemas a partir de um excerto do texto:

```
wnl = nltk.WordNetLemmatizer()
```

```
print(tokens[400:450])
```

```
['words', 'over', 'our', 'heads,', 'and', 'they', 'made', 'us', 'put', 'on', 'the', 'white', 'shirts', 'worn', 'by', 'persons',
'condemned', 'to', 'death.', 'Thereupon', 'we', 'were', 'bound', 'in', 'threes', 'to', 'stakes,', 'to', 'suffer', 'execution.',
'Being', 'the', 'third', 'in', 'the', 'row,', 'I', 'concluded', 'I', 'had', 'only', 'a', 'few', 'minutes', 'of', 'life', 'befor
e', 'me.', 'I', 'thought']
```

```
print([wnl.lemmatize(t) for t in tokens[400:450]])
```

```
['word', 'over', 'our', 'heads,', 'and', 'they', 'made', 'u', 'put', 'on', 'the', 'white', 'shirt', 'worn', 'by', 'person', 'co
ndemned', 'to', 'death.', 'Thereupon', 'we', 'were', 'bound', 'in', 'three', 'to', 'stakes,', 'to', 'suffer', 'execution.', 'Be
ing', 'the', 'third', 'in', 'the', 'row,', 'I', 'concluded', 'I', 'had', 'only', 'a', 'few', 'minute', 'of', 'life', 'before',
'me.', 'I', 'thought']
```

Pode ser observado que as palavras “shirts” e “persons” foram lematizadas; entretanto, “concluded” não foi reduzida (do tempo passado para o radical “conclude”). Ainda não existem lematizadores na NLTK para a língua portuguesa.

Dessa forma, a derivação geralmente se refere a um processo heurístico grosseiro, que corta as extremidades das palavras esperando atingir o objetivo corretamente na maioria das vezes e geralmente inclui a remoção de afixos derivacionais. A lematização geralmente se refere a fazer as coisas corretamente com o uso de um vocabulário e análise morfológica das palavras, normalmente com o objetivo de remover apenas terminações flexionais e retornar à forma base ou dicionário de uma palavra, conhecida como **lema**.

3.3 Segmentação

A tokenização é vista como uma instância de um problema mais geral de **segmentação**. A segmentação pode ser por sentenças ou mesmo por palavras. Manipular textos no nível de palavras individuais muitas vezes pressupõe a capacidade de dividir um texto em frases individuais. Alguns corpora, por exemplo, já fornecem acesso em nível de sentença (Bird; Klein; Loper, 2019).

No script a seguir mostra-se um exemplo de segmentação por sentenças com o conteúdo do livro *Memórias Póstumas de Brás Cubas* novamente. O comando utilizado é “sent_tokenize”. A variável gerada contém as sentenças geradas pelo caractere de nova linha dentro do texto base:

```
sents = nltk.sent_tokenize(raw)
```

```
sents[79:90]
```

```
['Veiu modernamente um professor e achou\nmeio de demonstrar que ambos esses conceitos eram erroneos e abstrusos,\ne que dos do
us cesares, o delicioso, o verdadeiramente delicioso, foi\no «abobora» de Seneca.',
 'E tu, madama Lucrecia, flor dos Borgias, se um\npoeta te pintou como a Messalina catholica, appareceu um Gregorovius\nincredu
lo que te apagou muito essa qualidade, e, se não vieste a\nlyrio, tambem não ficaste pantano.',
 'Eu deixo-me estar entre o poeta e o\nsabio.',
 'Viva pois a historia, a voluvel historia que dá para tudo; e, tornando\ná idéa fixa, direi que é ella a que faz os varões for
tes e os doudos;\na idéa mobil, vaga ou furta-cor é a que faz os Claudios,--formula\nSuetonio.',
 'Era fixa a minha idéa, fixa como... Não me occorre nada que seja assaz\nfixo nesse mundo: talvez a lua, talvez as pyramides d
o Egypto, talvez\na finada dieta germanica.',
 'Veja o leitor a comparação que melhor lhe\nquadrar, veja-a e não esteja dahi a torcer-me o nariz, só porque ainda\nnão chegám
os á parte narrativa destas memorias.',
 'Lá iremos.',
```

No caso da segmentação de palavras, essa tarefa é útil, por exemplo, quando se executa o processamento de linguagem falada, no sentido de separar as palavras a partir de um texto de fala capturada (Bird; Klein; Loper, 2019).

TEMA 4 – CLASSIFICAÇÃO E AVALIAÇÃO DE DOCUMENTOS

A detecção de padrões é uma parte crucial da Inteligência Artificial e, por consequência, do processamento de linguagem natural. Em língua portuguesa, os verbos regulares são formados pelo radical mais o sufixo, que tende a ser padronizado de acordo com o tempo de conjugação do verbo. Por exemplo, na conjugação do verbo “falar”, o radical “fal-” é adicionado aos sufixos de acordo com o tempo para conjugar; no tempo presente, adiciona-se os sufixos {-o, -as, -a, -amos, -ais, -am}, conforme a pessoa do verbo. Esses padrões observáveis – estrutura de palavras e frequência de palavras – se correlacionam com aspectos particulares do significado, como tempo verbal e o conteúdo. O desafio para detectar padrões se encontra em quais características devem ser consideradas com aspectos relacionados ao significado. Dentre várias tarefas, a **classificação** é um tipo de tarefa que permite identificar perfis de características com alguma categoria em particular.

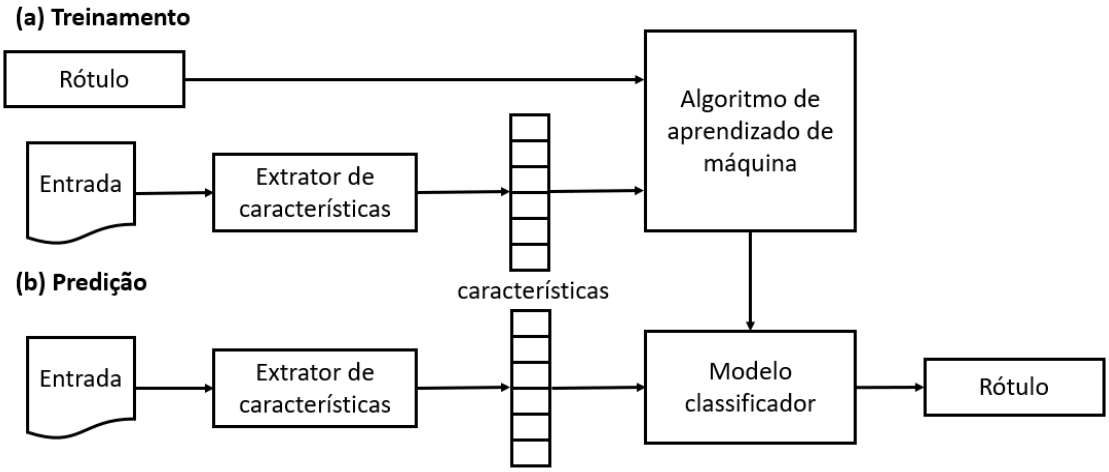
Conforme Bird *et al*. (2019), a **classificação** é a tarefa de escolher o rótulo de classe correto para uma determinada entrada de dados. Em tarefas básicas de classificação, cada entrada é considerada isoladamente de todas as outras entradas, e o conjunto de rótulos é definido antecipadamente. Alguns exemplos de tarefas de classificação são:

- decidir se um e-mail é considerado spam ou não;
- decidir qual é a categoria de um artigo de notícias, a partir de uma lista fixa de áreas temáticas como "esportes", "tecnologia" e "política"; e

- decidir se uma determinada ocorrência da palavra “banco” é usada para se referir a uma instituição financeira, a um assento de praça ou ao ato de bancar uma determinada situação.

Um classificador é dito **supervisionado** se for construído com base em corpora de treinamento contendo o rótulo correto para cada entrada. De maneira geral, tanto em machine learning quanto em NLP, um banco de dados com exemplos é dividido em dois conjuntos: um que ficará com o professor que irá **ensinar** efetivamente o modelo e outro com um professor que irá apenas **avaliar** o modelo: o seu conhecimento não será usado para a fase de aprendizagem. O primeiro conjunto é dito **conjunto de treinamento** e o segundo o **conjunto de teste**. Um processo de treinamento pode utilizar diferentes combinações aleatórias dos conjuntos, de acordo também com critérios de divisão do banco de dados e das quantidades de exemplos.

Figura 7 – Esquema de classificação supervisionada



Fonte: Elaborado com base em Bird *et al*., 2019.

Na figura anterior está descrito o esquema da classificação supervisionada. Em (a) durante o **treinamento**, um extrator de características é utilizado para converter cada valor de entrada em um conjunto de características (também denominado de “vetor de características”). Esses conjuntos capturam as informações básicas sobre cada entrada que devem ser suficientes para classificá-las. Pares de conjuntos de características e rótulos são alimentados no algoritmo de aprendizado de máquina para a geração de um modelo. Em (b), durante a **predição** ou teste, o mesmo extrator de características é usado para

converter para o conjunto de características as entradas que não estavam incluídas. Esses conjuntos de características são então inseridos com base do que foi aprendido pelo modelo, o que gerará então os rótulos preditos.

Em processamento de linguagem natural, várias tarefas podem ser enquadradas na classificação supervisionada.

- **identificação de gênero em discurso** – nomes que terminam em “a” provavelmente são do sexo feminino, enquanto os nomes que terminam em “o” provavelmente são do sexo masculino.
- **classificação de documentos** – a partir do uso de corpora, pode-se construir classificadores que marcarão automaticamente novos documentos com rótulos de categoria apropriados.
- **anotação ou marcação (tagging) de parte do discurso (part-of-speech)** – pode-se treinar um classificador para descobrir quais sufixos de palavras podem indicar alguma informação relevante; por exemplo, o tempo e a pessoa de um verbo.
- **exploração de contexto** – ao aumentar a função de extração de características, poderíamos modificar o anotador de parte do discurso para alavancar uma variedade de outros recursos internos da palavra, como o comprimento da palavra, o número de sílabas que ela contém ou seu prefixo. No entanto, desde que o extrator de características olhe somente para a palavra-alvo, não há como adicionar características que dependam do contexto em que a palavra aparece. Assim, as características contextuais podem fornecer dicas valiosas sobre a anotação (tag) correta. Por exemplo, ao marcar a palavra "partir", sabendo-se que a palavra anterior é "a", isso permitirá determinar que ela atua como uma locução adverbial, não como um verbo.
- **classificação de sequências** – na captura de dependências entre tarefas de classificação de entradas que são relacionadas de forma sequencial, pode-se utilizar modelos de classificadores conjuntos, que escolhem uma rotulagem apropriada para uma coleção de entradas relacionadas. No caso de anotação de parte do discurso, uma variedade de diferentes modelos de classificador de sequência pode ser usada para escolher conjuntamente os rótulos de parte da fala para todas as palavras em uma determinada frase.

Como exemplos de algoritmos para classificadores estão o Naive-Bayes, classificadores de entropia máxima e redes neurais artificiais.

TEMA 5 – EXEMPLO: CLASSIFICAÇÃO DE DOCUMENTOS

Um exemplo de classificação de documentos será desenvolvido a seguir. Esse modelo fará a classificação de documentos relacionados ao Corpus de Revisão de Filmes (Movie Review Corpus) que categoriza as revisões como positivas ou negativas. O modelo a ser criado irá, portanto, permitir que a partir de uma palavra presente no texto, seja possível correlacionar essa palavra com o rótulo positivo ou negativo (Bird; Klein; Loper, 2019).

Em primeiro lugar, é feita a importação da NLTK e do corpus de revisão de filmes:

```
import nltk
from nltk.corpus import movie_reviews
```

Em seguida, gera-se uma lista dupla no Python (“documents”) que relacione as palavras das revisões com o rótulo especificamente.

```
documents = [(list(movie_reviews.words(fileid)), category)
              for category in movie_reviews.categories()
                 for fileid in movie_reviews.fileids(category)]
```

Para identificar, por exemplo, uma revisão específica, basta colocar o índice para verificar (note, no final da lista, a palavra “neg”, indicando que a revisão de índice 100 é uma revisão com rótulo negativo):

```
print(documents[:100])
```

```
e', 'off', 'doing', 'the', 'classic', "'", 'sex', ',', 'drugs', ',', 'and', 'rock', 'n', "'", 'roll', "'", 'thing', ',', 'lea
ving', 'poor', 'strawberry', 'fields', 'without', 'her', 'true', 'love', ',', 'billy', '.', 'she', 'sets', 'out', 'to', 'fin
d', 'him', ',', 'leaving', 'her', 'poor', 'dad', 'and', 'mom', '(', 'mrs', '.', 'fields', '!', 'hey', ',', 'wanna', 'cookie',
'?', ')', 'all', 'alone', 'to', 'sing', 'a', 'horrible', 'rendition', 'of', "'", 'she', "'", 's', 'leaving', 'home', "'",
',', 'accompanied', 'by', 'those', 'same', 'creepy', 'robots', '.', 'but', 'is', 'billy', 'getting', 'it', 'on', 'with', 'luc
y', '(', 'dianne', 'steinberg', ')', '?', 'ya', 'know', ',', 'the', 'diamond', 'possessing', 'girl', 'who', 'hangs', 'out',
'in', 'the', 'sky', '?', 'well', ',', 'there', "'", 's', 'some', 'crazy', 'chick', 'singing', 'soprano', 'on', 'the', 'roof',
'of', 'a', 'building', 'next', 'to', 'a', 'bus', 'stop', ',', 'and', 'i', 'assume', 'that', "'", 's', 'who', 'it', 'was',
'.', 'it', "'", 's', 'movies', 'like', 'this', 'that', 'make', 'ya', 'sit', 'back', 'and', 'ask', 'the', 'unanswerable', 'que
stion', ',', '"', 'what', 'the', 'hell', 'were', 'they', 'thinking', '?', '?', '?', '"', 'nobody', 'will', 'ever', 'know',
',', 'but', 'as', 'a', 'novelty', ',', 'sgt', '.', 'pepper', "'", 's', 'is', 'one', 'to', 'examine', '.', 'carol', 'channin
g', ',', 'robert', 'palmer', ',', 'keith', 'carradine', '-', 'they', "'", 're', 'all', 'here', '.', 'but', 'why', '?', '?',
'?', '?', 'who', 'knows', '.', 'it', "'", 's', 'irrelevant', '.', 'there', "'", 's', 'as', 'much', 'meaning', 'to', 'be', 'fo
und', 'here', 'as', 'there', 'is', 'to', 'be', 'found', 'in', 'your', 'belly', '-', 'button', 'lint', ',', 'although', 'the',
'latter', 'may', 'be', 'more', 'interesting', '.', 'with', 'the', 'recent', 'onslaught', 'of', '70', "'", 's', 'nostalgia',
'in', 'the', 'movie', 'world', '(', 'the', 'ice', 'storm', ',', 'boogie', 'nights', ',', 'reissues', 'of', 'the', 'star', 'wa
rs', 'trilogy', ',', 'etc', '.', ')', ',', 'let', "'", 's', 'pray', 'this', 'doesn', "'", 't', 'get', 'a', 'special', '20th',
'anniversary', ',', 'second', '-', 'chance', 'in', 'theaters', '.', 'in', 'the', 'words', 'of', 'paul', 'mccartney', ',', 'li
ve', 'and', 'let', 'die', '.', 'in', 'fact', ',', 'bury', 'this', 'one', 'while', 'you', 'still', 'can', '.'], 'neg')]
```

É desejável que a lista de documentos esteja em uma ordem aleatória, o que pode ser conseguido a partir do comando “shuffle” do módulo “random”:

```
import random
random.shuffle(documents)
```

A seguir, efetua-se uma contagem de palavras dentro do corpus, por meio do módulo “FreqDist”:

```
all_words = nltk.FreqDist(w.lower() for w in movie_reviews.words())
```

```
all_words
```

```
FreqDist({',': 77717, 'the': 76529, '.': 65876, 'a': 38106, 'and': 35576, 'of': 34123, 'to': 31937, "'": 30585, 'is': 25195, 'i
n': 21822, ...})
```

Separa-se da variável "all_words" apenas uma lista das primeiras 2000 palavras, sem a contagem:

```
word_features = list(all_words)[:2000]
```

A próxima célula detalha a criação de uma função definida por usuário denominada “document_features”, que fará a extração das características, identificando a presença de cada palavra (ou não) contida em cada documento.

```
def document_features(document):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['contains({})'.format(word)] = (word in document_words)
    return features
```

Por exemplo, a chamada a essa função irá retornar, para um documento especificamente:

```
print(document_features(movie_reviews.words('pos/cv957_8737.txt')))
```

```
{'contains(,)': True, 'contains(the)': True, 'contains(.)': True, 'contains(a)': True, 'contains(and)': True, 'contains(of)':
True, 'contains(to)': True, "contains(')": True, 'contains(is)': True, 'contains(in)': True, 'contains(s)': True, 'contains
(")': True, 'contains(it)': True, 'contains(that)': True, 'contains(-)': True, 'contains())': True, 'contains((): True, 'con
tains(as)': True, 'contains(with)': True, 'contains(for)': True, 'contains(his)': True, 'contains(this)': True, 'contains(fil
m)': False, 'contains(i)': False, 'contains(he)': True, 'contains(but)': True, 'contains(on)': True, 'contains(are)': True,
'contains(t)': False, 'contains(by)': True, 'contains(be)': True, 'contains(one)': True, 'contains(movie)': True, 'contains(a
n)': True, 'contains(who)': True, 'contains(not)': True, 'contains(you)': True, 'contains(from)': True, 'contains(at)': Fals
e, 'contains(was)': False, 'contains(have)': True, 'contains(they)': True, 'contains(has)': True, 'contains(her)': False, 'co
ntains(all)': True, 'contains(?)': False, 'contains(there)': True, 'contains(like)': True, 'contains(so)': False, 'contains(o
ut)': True, 'contains(about)': True, 'contains(up)': False, 'contains(more)': False, 'contains(what)': True, 'contains(whe
n)': True, 'contains(which)': True, 'contains(or)': False, 'contains(she)': True, 'contains(their)': False, 'contains(:)': Tr
```

Note que se a palavra está contida no documento de revisão que foi detalhado no argumento (“pos/cv957_8737.txt”), ela irá indicar “True” se estiver presente ou “False” em caso contrário.

A partir de agora, cria-se o modelo para fazer o treinamento propriamente dito. Dos três comandos, o primeiro é o que relaciona a extração das características (presença ou não de palavras na revisão) com o rótulo nos documentos. O segundo faz a divisão do conjunto de treinamento e do conjunto de teste ou predição. O terceiro comando é o classificador Naive-Bayes, que usa o conjunto de treinamento como argumento.

```
featuresets = [(document_features(d), c) for (d,c) in documents]
train_set, test_set = featuresets[100:], featuresets[:100]
classifier = nltk.NaiveBayesClassifier.train(train_set)
```

Para avaliar se o modelo conseguiu aproximar das revisões fornecidas pelo corpus, utiliza-se a métrica da acurácia para a verificação do conjunto de teste (que não fez parte do treinamento) que retorna um valor de 0.81:

```
print(nltk.classify.accuracy(classifier, test_set))
```

```
0.81
```

É possível ainda verificar em detalhe as características mais relevantes encontradas no processo de avaliação do classificador (a partir do comando “shoe_most_informative_features”):

```
classifier.show_most_informative_features(5)
```

```
Most Informative Features
   contains(outstanding) = True              pos : neg    =     10.9 : 1.0
         contains(mulan) = True              pos : neg    =      8.6 : 1.0
        contains(seagal) = True              neg : pos    =      7.2 : 1.0
   contains(wonderfully) = True              pos : neg    =      6.4 : 1.0
         contains(damon) = True              pos : neg    =      6.1 : 1.0
```

É possível identificar que a palavra “seagal” possui mais avaliações negativas que positivas (em torno de 7.2 para 1), enquanto a palavra “mulan” obteve alto índice de avaliações positivas sobre as negativas (em torno de 8.6 para 1).

FINALIZANDO

Esta aula apresentou de maneira preliminar a linguagem Python e uma das bibliotecas mais utilizadas para processamento de linguagem natural, a NLTK . Com ela, é possível desenvolver scripts para execução de forma rápida e bastante interativa. Também foram explicadas algumas tarefas de NLP, como a tokenização, a derivação (*stemming*), a segmentação e a lematização. Em foco, a classificação e a avaliação de documentos foram abordadas, sendo explanado como um modelo genérico de classificação supervisionada é estruturado. O exemplo de classificação de documentos utilizando o Movies Review Corpus foi explicado, mostrando a prática de classificação no ambiente de programação Jupyter Notebook com a linguagem Python.

REFERÊNCIAS

BIRD, S.; KLEIN, E.; LOPER, E. **Natural Language Processing with Python**. [*S. l.*], 2019. Disponível em: <http://www.nltk.org/book/>. Acesso em: 10 ago. 2021.

KOPEC, D. **Classic Computer Science Problems in Python**. Shelter Island-NY: Manning Publications Co., 2019.

MÜLLER, A. C.; GUIDO, S. **Introduction to Machine Learning with Python and Scikit-Learn**. [*S. l.: s. n.*], 2015.