POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Extrator de características de letras para reconhecimento de emoção em música [LMER-FE]

Licenciatura em Engenharia Informática

João Pedro Nascimento Ferreira nº 2191241

Miguel Alexandre Inácio Mendonça nº 2200652

Leiria, setembro de 2023

POLITÉCNICO DE LEIRIA | ESCOLA SUPERIOR DE TECNOLOGIA E GESTÃO

Extrator de características de letras para reconhecimento de emoção em música

[LMER-FE]

Licenciatura em Engenharia Informática

João Pedro Nascimento Ferreira nº 2191241

Miguel Alexandre Inácio Mendonça nº 2200652

Trabalho de Projeto da unidade curricular de Projeto Informático realizado sob a orientação do Professor Doutor Ricardo Manuel da Silva Malheiro.

Leiria, setembro de 2023

Agradecimentos

Expressamos os nossos sinceros agradecimentos aos professores e à escola pelo apoio e contribuição ao longo do nosso percurso académico. Agradecemos especialmente ao nosso orientador pelo seu empenho em nos orientar durante este projeto. Também expressamos gratidão aos restantes professores, à instituição de ensino, aos nossos colegas e aos nossos familiares e amigos. Reconhecemos a importância de cada um deles na nossa formação académica e agradecemos pelo seu encorajamento e apoio incondicional. Finalizamos expressando a nossa profunda gratidão por termos tido a oportunidade de aprender e fazer parte dessa instituição exemplar.

Resumo

O reconhecimento de emoções a partir da música (Music Emotion Recognition – MER) está cada vez mais a ser um alvo de pesquisa mais significativa por parte da comunidade científica que se dedica à Recuperação de Informação Musical (Music Information Retrieval).

De facto, a pesquisa de música através de emoções é um dos tipos de pesquisa mais efetuados atualmente pelos utilizadores. Bases de dados musicais de sites como o AllMusic[1], Spotify[2], ou o Last.fim[3] crescem gradualmente todos os dias, o que requer uma enorme atenção e quantidade de trabalho para manter atualizado todo o processo de catalogação. Infelizmente, a classificação manual com etiquetas emocionais é normalmente um processo muito subjetivo e demorado.

Este problema pode ser ultrapassado com a utilização de sistemas de reconhecimento automático. Além de classificação automática de música, o MER tem várias outras aplicações como recomendação de música, criação automática de playlists e desenvolvimento de jogos, cinema, publicidade e saúde.

A maioria dos primeiros sistemas automáticos de MER eram baseados apenas na análise do áudio, posteriormente estudos vieram a revelar que ao combinar as duas dimensões (áudio e letra da música) melhorava significativamente a eficácia dos sistemas MER.

Este projeto foca-se num projeto MERGE (Music Emotion Recognition – Next Generation) que é o seguimento de um outro projeto chamado MOODetector, ambos financiados pela FCT e do qual o nosso orientador participa/ou.

No âmbito destes projetos foram construídos recursos nomeadamente um extrator de features, anteriormente implementado em JAVA. Como cada vez mais é utilizado a linguagem Python decidiu-se migrar a aplicação de forma a atualizar a aplicação para uma linguagem mais atual.

Este projeto centra-se principalmente no papel das letras musicais no processo de MER. A extração de características (feature extraction) é uma das etapas mais importantes no

[1] https://www.allmusic.com
[2] https://www.spotify.com/pt-pt/
[3] https://www.last.fm

processo de implementação de sistemas de extração de features de letras musicais em Python utiliza grande parte das características utilizadas no estado de arte complementadas por novas características estilísticas, estruturais, semânticas e baseadas em conteúdo.

Abstract

Emotion recognition from music (Music Emotion Recognition - MER) is becoming an increasingly significant research target within the scientific community dedicated to Music Information Retrieval. In fact, searching for music by emotions is one of the most commonly performed types of queries by users nowadays. Music databases from websites like AllMusic, Spotify, or Last.fm are growing steadily every day, requiring immense attention and a substantial amount of work to keep the entire cataloging process up to date. Unfortunately, manual classification with emotional labels is often a very subjective and time-consuming process.

This issue can be overcome through the use of automatic recognition systems. Apart from automatic music classification, MER has several other applications, such as music recommendation, automatic playlist generation, and development in fields like gaming, cinema, advertising, and healthcare.

The majority of early automatic MER systems were based solely on audio analysis. Subsequently, studies revealed that combining both dimensions (audio and lyrics) significantly improved the effectiveness of MER systems.

This project focuses on a MERGE (Music Emotion Recognition - Next Generation) project, which is a continuation of another project called MOODetector, both funded by FCT and involving our advisor. As part of these projects, resources have been constructed, including a feature extractor previously implemented in JAVA. As Python is increasingly being utilized, the decision was made to migrate the application in order to update it to a more current language.

This project primarily focuses on the role of music lyrics in the process of MER. Feature extraction is one of the most crucial stages in implementing systems for extracting features from music lyrics in Python. It incorporates a significant portion of the features used in the state of the art, supplemented by new stylistic, structural, semantic, and content-based characteristics.

Índice

Lista de Figuras

Lista de tabelas

Lista de siglas e acrónimos

Tabela 1 Lista de siglas e acrónimos

ANEW	affective norms for english words
ACL	all capital letters
BOW	bag of words
CBF	content based features
DAL	dictionary of affective language
FCL	First capital letter
FCT	Fundacao para a Ciencia e a Tecnologia
GI	general inquirer
GUI	Graphical user interface
IDF	Inverse document frequency
LIWC	linguistic inquiry and word count
MER	music emotion recognition
MERGE	Music emotion recognition – Next Generation
MIR	music information retrieval
NLP	natural language processing
NLTK	Natural Language Toolki
NRC	National Research Council
POS Tags	part-of-speech tags
SemBF	semantic based features
StruBF	structure based features
StyBF	stylistic based features
STP	Standard pos tagger
TFIDF	term frequency–inverse document frequency
TF	Term frequency

1. Introdução

Este relatório apresenta o desenvolvimento de uma aplicação em Python voltada para a extração de features (características) a partir de letras de músicas. O objetivo é a partir do output desta aplicação poder-se construir um sistema de reconhecimento de emoções em letras de música. A música é uma forma poderosa de expressão artística que tem a capacidade de evocar emoções intensas e transmitir mensagens profundas. Compreender as emoções subjacentes em músicas é de grande interesse tanto para a indústria musical (e.g., pesquisa por emoções em aplicações como spotify ou allmusic) quanto para a pesquisa em áreas como psicologia, ciência cognitiva e análise de dados. É também importante para outras áreas como saúde (e.g., Sistemas que permitem melhorar determinadas patologias cognitivas como a alzheimer através da sugestão de determinados tipos de música), entretenimento (e.g., no cinema e jogos de computador a música ser recomendada de acordo com o momento do filme ou jogo) e publicidade (e.g., sugestão de música adequada aos propósitos da publicidade).

Esta aplicação é construída no âmbito de um projeto do Centro de Informática e Sistemas da Universidade de Coimbra (CISUC) chamado MERGE (Music Emotion Recognition New Generation) financiado pela FCT, ao qual o nosso orientador pertence e que pretende melhorar o estado de arte no que toca ao reconhecimento de emoções em música (áudio + letra) utilizando tanto abordagens clássicas de *machine learning* que inclui os processos de extração de features seguido de classificação/regressão e avaliação como abordagens de *deep learning* e *transfer learning*. O projeto incluí tanto a construção de sistemas de reconhecimento de emoções em música (MER) só para o áudio da música, só para a letra da música e bimodal incluindo o áudio e a letra.

No âmbito do projeto o nosso orientador já tinha construído no âmbito do seu doutoramento, com várias features criadas por si, uma aplicação para extração de features a partir de letras de música em linguagem java. A sugestão para o nosso projeto foi o de migrar esta aplicação em java para a linguagem Python por esta ser a mais usada atualmente.

O objetivo deste projeto de licenciatura foi, portanto, criar uma ferramenta capaz de extrair features a partir de letras de música com o objetivo de poder ser usada no processo de MER. Para alcançar esse objetivo, foram exploradas essencialmente técnicas de processamento de linguagem natural (NLP[4]), utilizando bibliotecas e recursos disponíveis em Python.

A aplicação desenvolvida utiliza um conjunto diversificado de recursos linguísticos e algoritmos para extrair informações relevantes das letras de músicas. Foram utilizados recursos de análise sintática, tais como etiquetagem de palavras (*POS tagging*), e recursos semânticos, como dicionários de emoções. Além disso, foram exploradas técnicas de representação de texto, como a modelação de BOW (*bag-of-words*) e a utilização de índices de frequência de termos e outras medidas como TFIDF (*term frequency–inverse document frequency*).

Espera-se que a aplicação desenvolvida possa ser importante para o processo de MER, já que permite automatizar o processo de extração de features. Além disso, o projeto ofereceu uma oportunidade de aprofundamento no estudo de processamento de linguagem natural, *machine learning* e desenvolvimento de aplicações em Python.

No contexto do entretenimento, a análise de emoções em letras de músicas tem sido cada vez mais utilizada em ferramentas como o Spotify, que utilizam algoritmos avançados para recomendar músicas baseado no estado de espírito do utilizador. Essas recomendações personalizadas levam em consideração as emoções transmitidas pelas letras das músicas, proporcionando uma experiência musical mais envolvente e cativante.

Além disso, a compreensão das emoções nas letras de músicas também possui aplicações práticas em outras áreas, como saúde. A música tem sido explorada como uma terapia complementar, ajudando no tratamento de distúrbios emocionais, como ansiedade e depressão. Ao identificar as emoções presentes nas letras, é possível selecionar músicas adequadas para promover o relaxamento, motivação ou expressão emocional.

No decorrer deste relatório, serão apresentados os detalhes do processo de desenvolvimento da aplicação, as técnicas utilizadas, os resultados obtidos e as conclusões relevantes.

[4] Natural language processing

Assim como nesta introdução, no resto do relatório serão utilizados vários estrangeirismos relacionados essencialmente com termos técnicos da área de informática por habitualmente os conhecermos na sua representação original estando estes representados pelo tipo de letra itálico.

O relatório está dividido da seguinte forma:

- Capítulo 2: Estado de arte
    - Começamos por descrever o que é a tecnologia MER e a sua crescente atenção que tem recebido para a comunidade científica. Destacamos que esta avança no que toca a análise tradicional das características como timbre, ritmo, melodia e visa identificar emoções a partir de letras musicais utilizando abordagens mais abrangentes.
    - De seguida, exploramos a parte teórica do nosso projeto, está dividida em 4 categorias: Content-based features, Stylistic-based features, Structural-Based features, Semantic-Based features. Resumimos cada uma destas features inicialmente e de seguida exploramos uma a uma detalhadamente, explicando as suas origens, que funcionalidades contêm e exemplos do seu funcionamento.
- Capítulo 3: Trabalho realizado
    - Neste capítulo começamos por descrever as janelas que criámos na interface do utilizador e a funcionalidade de cada uma.
    - Explicamos que bibliotecas e funções utilizamos para o desenvolvimento da aplicação.
    - De seguida, é feita uma explicação detalhada para cada feature implementada das funções implementadas, desde o código, o seu funcionamento passo-a-passo e por fim um exemplo de um possível output.
- Capítulo 4: Conclusão
    - Neste capítulo, resumimos o projeto, realçando a sua importância para a comunidade científica e as aprendizagens que adquirimos. Exploramos as complexidades emocionais nas letras musicais, adquirindo competências em Processamento de Linguagem Natural e análise textual. Enaltecemos também que a abordagem interdisciplinar uniu música e tecnologia, enriquecendo a nossa

compreensão e terminamos com a ideia de que esta aplicação tem potencial para impactar a comunidade científica.

2. Estado de Arte

2.1.Music Emotion Recognition

Desde há algum tempo o reconhecimento de emoções em música (MER) está a atrair o foco da comunidade científica de recuperação de informação em música (MIR). Atualmente, investigadores têm trabalhado para desenvolver métodos eficazes de MER que vão além da mera análise das características musicais, como por exemplo, o timbre, ritmo, dinâmica, tonalidade, melodia, harmonia, tempo e expressividade vocal, estas podem variar dependendo do contexto específico e preferências dos investigadores. Este trabalho tem imenso impacto no melhoramento da interação do utilizador com sistemas de recomendação musical e personalização de playlists, como é o caso do Spotify, pois as preferências e estados de espírito individuais são altamente influenciados pelas emoções evocados pela música (Yang, 2012).

Além disso, técnicas de *machine learning* têm sido aplicadas para classificar as emoções com base nessas features extraídas (Koelsch, Stefan ,2010). Com o crescente interesse em pesquisas interdisciplinares entre música e *computer science*, espera-se que o MER continue a avançar e melhorar a forma como interagimos e desfrutamos da música em diversos contextos (Li et al., 2017).

As bases de dados de música no mundo real, provenientes de sites como AllMusic ou Last.fm estão a crescer gradualmente, o que requer uma quantidade enorme de trabalho manual para mantê-las atualizadas. Infelizmente, a anotação manual de música com etiquetas de emoção é normalmente um processo subjetivo e uma tarefa dispendiosa e demorada.

Este problema pode ser superado com o uso de sistemas automáticos de reconhecimento (Hu, Downie,2010).

A maioria dos sistemas automáticos de MER em fases iniciais baseava-se na análise do conteúdo áudio por exemplo, (Lu et al., 2006). Mais tarde, os investigadores começaram a combinar áudios e letras, levando a sistemas bimodais de MER com uma precisão melhorada (Hu and Downie,2010), (Hu et al,2009), (Laurier et al,2008).

Por exemplo na música dança, muitas vezes, o áudio é a dimensão mais relevante no ponto de vista da perceção de emoções (e.g., happy de Pharrell Williams (Malheiro, 2017)),

enquanto na música poética é normal as letras se destacarem mais do ponto de vista da perceção das emoções (e.g., Ne me quitte pas de Jacques Brel (Malheiro, 2017)).

Para a implementação de um sistema automático de classificação de música (áudio e letra) em emoções temos antes de mais de levar em consideração com o modelo de emoções a utilizar. De acordo com (Malheiro, 2017) o modelo de emoções de Russell (Russel, 2000) é um dos modelos mais utilizados pela comunidade científica e por aplicações comerciais.

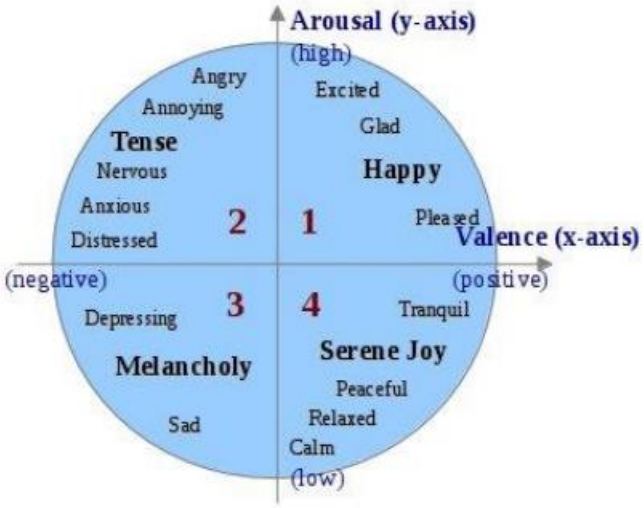


Figura 1 Russell's circumplex model (Malheiro, 2017))

O modelo de Russell propõe que as emoções humanas podem ser representadas por duas dimensões correspondentes aos 2 eixos cartesianos: valência e *arousal* (Russell, 1980). Essas duas dimensões ajudam a entender como diferentes emoções se relacionam entre si e como podem ser agrupadas.

A dimensão de valência descreve o grau de positividade ou negatividade associado a uma emoção. Por outras palavras, ela mede se uma emoção é interpretada como agradável ou desagradável.

A dimensão de *arousal* refere-se ao nível de excitação ou intensidade da emoção. Esta mede o quão energética e estimulante é a emoção em particular.

Com essas duas dimensões, podemos posicionar diferentes emoções num espaço bidimensional. Por exemplo, a felicidade e alegria são emoções com valência positiva e *arousal* positivo (1º quadrante), raiva e angústia são emoções com valência negativa e *arousal* positivo (2º quadrante), tristeza e depressão são emoções com valência e *arousal* negativos (3º quadrante) e a alegria serena e relaxamento são emoções com valência positiva e *arousal* negativo (4º quadrante).

A representação das emoções nestas duas dimensões pode ser útil em diversos contextos, incluindo a pesquisa de emoções na música, em que diferentes trechos musicais podem invocar emoções com variações de valência e *arousal*.

Este projeto que implementamos em Python está direcionado para extração de *features* a partir de letras musicais.

2.2. Features Extraction

As features consideradas neste projeto estão divididas em 4 categorias e são as categorias de features criadas no âmbito do projeto MERGE pelo nosso orientador (Malheiro et al, 2016):

- Content-Based features (CBF): atributos ou características específicos extraídos a partir do conteúdo da letra e que inclui n-gramas (unigramas, bigramas e trigramas) aos quais podem ser aplicadas operações como remoção de *stopwords* e/ou *stemming* das palavras.
- Stylistic-based features (StyBF): atributos ou características que refletem o estilo de escrita do texto da letra. São utilizadas para analisar e distinguir as diferenças estilísticas entre diferentes músicas.

  Algumas das Stylistic-based features comuns incluem: contagem de palavras classificadas como vocabulário "calão" (*slang words*), a utilização e contagem de palavras todas as maiúsculas (*capital letters*) numa letra musical, a frequência de palavras das várias classes gramaticais (*Part-of-Speech Tags* – POS Tags) presentes no texto como por exemplo nomes e adjectivos.
- Structured-based features (StruBF): atributos ou características que se referem à organização e à forma da própria letra musical. Estão relacionadas à estruturação das palavras, frases, versos na composição lírica.

  Algumas das Structured-based features comuns incluem: contagem de frequência do refrão, contagem de vezes que o título da música é referido na letra.
- Semantic features (SemBF): atributos ou características de uma letra musical que se referem aos atributos e informações relacionados ao significado e ao conteúdo semântico das palavras, frases e versos presente na letra da música.

  Algumas das Semantic features comuns incluem: cálculo da média de valência e *arousal* das palavras com conteúdo emocional (a partir de dicionários de palavras com contexto emocional) presentes na letra, utilização de bibliotecas e/ou

frameworks (LIWC[5], NRCLex[6], General Inquirer [7]para extrair informação de cada palavra presente na letra com base nestas *frameworks*.

2.2.1. Content-Based Features (CBF)

As características mais comuns usadas na análise de texto, assim como na análise de letras, são características baseadas no conteúdo (CBF), nomeadamente de modelo *bag-of-words* (BOW) (Sebastiani,2002).

Neste projeto, utilizamos apenas as *bag-of-words* unigramas, bigramas e trigramas, na maioria dos casos estes são os mais relevantes.

O BOW é uma técnica amplamente utilizada no processamento de linguagem natural e na análise de texto. Essa abordagem consiste em representar um texto como um conjunto não ordenado de palavras, ignorando a estrutura gramatical e a sequência de palavras.

O processo BOW está definido pelas seguintes etapas

1. Tokenização: O texto é dividido em unidades linguísticas chamadas *tokens*, podem ser conjuntos de unigramas, bigramas, trigramas
2. Pré-processamento: O texto é normalizado para remover a pontuação, converter todas as palavras para letras minúsculas e eliminar palavras irrelevantes, como:
    a. Artigos e preposições (*stopwords*)
    Por exemplo, “a”, “the”, “is”, “are”, como são palavras muitos comuns acabam por trazer pouco conhecimento de modo que se torna útil eliminá-las para obtermos modelos mais precisos.
    b. Aplicar operação de *stemming* isto é, reduzir a palavra à sua raíz.
    Por exemplo a palavra “programming”, “programmer”, e “programs” podem ser todas reduzidas à palavra “program”.
    c. Se pretendermos ser ainda mais específicos é ainda possível efetuar a combinação das duas operações, ou seja, aplicar as *stemming* e remoção de *stopwords*.
3. Construção da BOW: As palavras são contadas e formam um vetor de características que representam o texto.

[5] https://www.liwc.app
[6] https://pypi.org/project/NRCLex/
[7] Um sistema informático para análise e recuperação de conteúdo com base na frase como unidade de informação..

Por exemplo na frase: “Learning is a good practice.”
O BOW de unigramas seria: {“Learning”, “is”, “a”, “good”, “pratice”}.
O BOW de bigramas seria: {“learning is,”is a”,”a good”,”good practice”}.
O BOW de trigrams seria: {“learning is a”,” is a good”,” a good practice”}.

As features podem ser representadas neste projeto de 3 formas distintas:

1. Representação por frequência: Esta representação permite que ao percorrer todos os documentos (letras) de uma pasta, calcular a frequência de ocorrência por letra em todas as letras de cada feature.
2. Representação booleana: Esta representação permite nos saber ao percorrer todos os documentos (letras) de uma pasta, verificar se a feature existe (1) ou não existe (0) em cada uma das letras.
3. Representação TF-IDF: esta representação permite a partir da representação por frequência calcular a medida TF-IDF que avalia a importância de uma feature, dentro de um conjunto de documentos. Este funciona calculando dois fatores principais:
    a. TF (Term Frequency): mede com que frequência um n-grama aparece num ficheiro. Quanto mais vezes este aparecer, maior será o seu valor dentro do documento.
    b. IDF (Inverse Document Frequency): mede a raridade de um n-grama em relação a todos os documentos. Se o n-grama aparecer em vários documentos, o seu valor IDF será menor. Por outro lado, se este tiver poucas presenças nos documentos, o seu valor IDF será maior.

Na nossa aplicação, optámos por utilizar apenas combinações de unigramas, bigramas e trigramas com as transformações mencionadas anteriormente.

2.2.2. Stylistic-Based Features (StyBF)

Estas features estão relacionadas com aspetos estilísticos da linguagem, ou seja, estão relacionadas com o estilo do texto da letra, bem como a escolha do tipo de palavras para transmitir uma determinada ideia (ou emoção, no nosso caso). No caso da música, essas questões podem estar relacionadas ao estilo do compositor, ao género musical ou às emoções que pretendemos transmitir.

Utilizámos duas features relacionadas com o uso de letras maiúsculas: *All Capital Letters* (ACL), que devolve o número de palavras escritas inteiramente em maiúsculas e *First Capital Letters* (FCL), que representa o número de palavras iniciadas por uma letra maiúscula.

O uso de letras maiúsculas numa letra musical pode ter várias influências e impactos na perceção e interpretação da música, tais como:

1. Ênfase e Intensidade: pode ser empregue para destacar palavras ou orações importantes e aumentar a ênfase e a intensidade do conteúdo lírico.
   Por exemplo: WE WILL ROCK YOU, Queen[8] é uma música que após a análise detalhada da letra, na nossa opinião transmite intensidade e enfatiza a mensagem da união.
2. Transmissão de Emoção: podem ser usadas para transmitir emoções específicas, como raiva, gritos ou excitação. Por exemplo: ROAR" - Katy Perry [9] segundo a nossa opinião após análise da letra é utilizado para transmitir um rugido poderoso.
3. Organização e Estrutura: pode ser empregue para organizar e estruturar a letra da música. Por exemplo: I WILL ALWAYS LOVE YOU - Whitney Houston[10] está escrito inteiramente em letras maiúsculas, enfatizando a mensagem central da canção: uma promessa eterna de amor.
4. Criação de contraste: Ao alternar entre letras maiúsculas e minúsculas, os compositores podem criar contrastes entre diferentes partes de música. Por exemplo: "SAY my name" - Destiny's Child[11] a música tem um tema de relacionamento e comunicação, e que para nós transmite a mensagem onde uma pessoa está a pedir ao parceiro para pronunciar o seu nome em vez de evitá-lo ou não o reconhecer. O uso de letras maiúsculas no título, especificamente em "SAY", cria um contraste com o restante do título em minúsculas ("my name").

[8] https://www.youtube.com/watch?v=-tJYN-eG1zk&pp=ygUQd2Ugd2lsbCByb2NrIHlvdQ%3D%3D

[9] https://www.youtube.com/watch?v=CevxZvSJLk8&pp=ygUEcm9hcg%3D%3D

[10] https://www.youtube.com/watch?v=3JWTaaS7LdU&pp=ygUWaSB3aWxsIGFsd2F5cyBsb3ZlIHlvdQ%3D%3D

[11] https://www.youtube.com/watch?v=sQgd6MccwZc&pp=ygULc2F5IG15IG5hbWU%3D

No entanto, é importante notar que o uso de letras maiúsculas deve ser cuidadosamente considerado, pois um uso excessivo ou inadequado pode afetar negativamente a perceção da música.

Implementamos também a feature de POS tagging: é uma ferramenta ou algoritmo utilizado em NLP para atribuir as partes do discurso a cada palavra num texto. As partes do discurso referem-se às categorias gramaticais às quais as palavras pertencem, como substantivos, verbos, adjetivos, advérbios, preposições, entre outros.

Quando se trata de letras musicais, o POS tagging pode ser utilizado como parte de uma análise de texto para extrair informações relevantes sobre as palavras e a sua estrutura gramatical. Isto pode ser útil para entender a organização do texto, identificar os tipos de palavras mais comuns na música (substantivos, verbos, adjetivos, etc.), e até mesmo analisar o sentimento e a emoção expressos por meio do uso de certas partes do discurso.

Por exemplo, a marcação gramatical da seguinte frase "The student read the book" seria "The/DT student/NN read/VBZ the/DT book/NN", onde DT, NN e VBZ significam, respectivamente, determinante, substantivo e verbo na terceira pessoa do singular no tempo presente. Esta análise é efetuada de acordo com o contexto da palavra na frase.

Por fim, implementamos uma nova característica: o número de ocorrências de palavras de gíria (conhecidas por *slangwords*). Essas palavras são retiradas do Dicionário de Gírias Online[12] (gírias americanas, inglesas e urbanas) fornecido pelo nosso orientador. Implementamos esta feature porque, em aspetos específicos como o Hip-Hop ou Rap as ideias são expressas normalmente com muitas destas palavras, o que torna esta feature importante para descrever emoções especificas associadas a géneros musicais específicos e verificou-se que é importante para a identificação de músicas do 2º quadrante (Malheiro, 2017).

2.2.3. Structural-Based Features

Estas características estão relacionadas a atributos extraídos da estrutura ou organização de uma letra musical, em vez de se concentrarem apenas no conteúdo morfológico, sintático ou semântico das palavras e frases.

[12] https://www.dictionary.com/e/slang/

As características que foram implementadas neste projeto foram:

1. Número de repetições do título: é uma técnica comum na música e pode ser utilizada de várias maneiras para transmitir emoções, enfatizar a mensagem da música e criar um efeito memorável.
   Por exemplo: "Hey Jude" - The Beatles[13]: Nesta música icónica dos Beatles, o título "Hey Jude" é repetido várias vezes no refrão. A repetição ajuda a criar um senso de camaradagem e conexão entre a banda e o público. A música é como uma mensagem de encorajamento, onde "Jude" pode ser interpretado como uma pessoa genérica que está a ser consolada e incentivada. A repetição do título na nossa opinião, reforça a mensagem de apoio e empatia. Valores altos desta feature estão muito associados a letras com emoções predominantes do 1º quadrante (Malheiro, 2017).

2. Deteção do refrão: o número de repetições do refrão de uma música pode influenciar diversos aspetos da composição e da experiência do ouvinte. O refrão é uma parte da música que é repetida várias vezes ao longo da composição e geralmente contém a mensagem central ou o tema principal da música. A quantidade de repetições do refrão pode afetar a forma como a música é percebida pelo público. Algumas formas como o número de repetições do refrão pode influenciar uma música incluem:
   a. Ênfase na Mensagem Central: Quanto mais vezes o refrão é repetido, maior a ênfase na mensagem central da música. A repetição enfatiza a ideia ou sentimento expresso no refrão, tornando-o mais memorável e impactante.
   b. Criação de *engagement* [14]do Público: Quando o refrão é repetido várias vezes, os ouvintes têm mais oportunidades de memorizar e posteriormente a cantá-la. Aumentando o envolvimento emocional do público com a música.
   c. Criar um ponto alto da música: A repetição do refrão pode ser usada para criar um ponto alto ou clímax na música.

[13] https://www.youtube.com/watch?v=mQER0A0ej0M&pp=ygUIaGV5IGp1ZGU%3D

[14] Compromisso com o público

d. Unir o público em concertos: Em shows ao vivo, a repetição do refrão pode ser usada para unir o público, incentivando-os a cantar. Isso cria uma atmosfera de camaradagem e conexão entre o artista e o público.

Valores altos desta feature estão muito associados a músicas do 1º quadrante (Malheiro, 2017).

De notar que esta implementação apenas deteta o refrão em letras de música (que estejam anotadas em sites como o genius[15] com metadados como [chorus], não faz a contagem do número de vezes que o refrão é repetido na letra. Este processo já não estava funcional na versão java que nos foi fornecida, porque no projeto MERGE para efeitos de investigação esta feature (contagem do refrão na letra) foi extraída de uma forma semi-automática. Segundo o nosso orientador no projeto está-se a trabalhar no processo de extração automática da feature e portante deverá ser implementada numa próxima iteração do nosso projeto.

2.2.4. Semantic-Based Features (SemBF)

Estas features estão relacionadas aos atributos e informações relacionados ao significado e ao conteúdo semântico das palavras, frases e versos presentes nas letras da música.

Utilizámos *framewords* e bibliotecas para a implementação desta feature, tais como:

1. NRCLex: é uma biblioteca para análise de sentimentos e emoções em texto, desenvolvida em Python pela NRC (National Research Council) Canadá. Utiliza técnicas de NLP para atribuir polaridades emocionais a palavras e textos. Esta base de dados está divida em 10 emoções/sentimentos: ‘Fear’, 'Anger', 'Anticip', 'Trust', 'Surprise', 'Positive', 'Negative', 'Sadness', 'Disgust', 'Joy'.

   Ao analisar um texto, a ferramenta identifica as palavras presentes e verifica as suas correspondentes emoções e polaridades na base de dados. Esta feature não existia na implementação em java e, portanto, após aprovação do orientador esta foi inserida na nossa aplicação.

[15] https://genius.com

2. Gazetteers: listas de palavras ou termos específicos associados a conceitos semânticos particulares. Podem incluir palavras-chave que representam tópicos específicos, entidades ou qualquer outro significado semântico. Cada gazetteer é, portanto, um dicionário de palavras classificadas com as dimensões de valência e *arousal*.
3. No nosso caso foi nos fornecido 5 dicionários (Gazetteers), um que incluí palavras num contexto emocional independentemente do seu quadrante) e os outros 4 que são específicos de cada um dos 4 quadrantes e que contém palavras com contexto emocional desses quadrantes e que estão classificadas com os valores respetivos de valência e *arousal*. A utilização destes 5 gazetteers vai, portanto, permitir criar 10 novas features para cada letra que correspondem á media dos valores de valência e arousal das palavras da letra presentes em cada um dos gazetteers.
4. General Inquirer (GI): é um sistema de análise de texto desenvolvido (Philip et al, 1966) usada para análise de textos e extração de features semânticas. Contém uma base de dados com uma lista extensa de palavras, rotulados por 182 rótulos semânticos, permitindo identificar e categorizar diferentes tipos de palavras em relação ao seu significado.
   Cada palavra da lista está rotulada por um ou mais rótulos. Por exemplo a palavra "alegria" pode ser considerada à categorização de emoção positiva. Já a palavra "abandono" está rotulada como uma palavra "negativ", "weak" e "fail".

5. DAL_ANEW: é uma base de dados léxico-semântica "DAL (Dictionary of Affect in Language) (Whissel, 1989) e ANEW (Affective Norms For English Words) (Bradley e Lang, 1999). Contém informações sobre palavras em inglês e os seus valores de valência, *arousal* e *dominance*.
   O valor *dominance* refere-se ao grau de controlo ou influência que um estímulo exerce sobre a pessoa que o percebe. Por exemplo um valor de baixa *dominance* está associado a sentimentos de submissão e passividade, já um valor de alta *dominance* está associado a sentimentos como poder ou controlo.

Através desta base de dados conseguimos calcular a média da valência, *arousal* e *dominance* das palavras das letras presentes nessa base de dados.

6. LIWC: "Linguistic Inquiry and Word Count" é umaferramenta de análise de texto desenvolvida por James W. Pennebaker e colegas na Universidade do Texas em Austin.

   O LIWC é uma abordagem computacional que utiliza dicionários léxico-semânticos para analisar e quantificar diferentes aspetos da linguagem e do conteúdo emocional em textos escritos.

   Através desta ferramenta conseguimos classificar cada palavra com uma das features que esta ferramenta oferece, que no total contém 62 tipos de features diferentes.

7. Warriner: refere-se ao conjunto de palavras conhecidos como “Warriner’s English Word Ratings” (Victoria A. Warriner, David A. Kuperman e Michael J. Brysbaert, 2013). Estas listas são uma compilação de palavras da língua inglesa acompanhada das suas classificações nas dimensões de valência, *arousal* e *dominance*.

   Os dados do "Warriner" são usados em análises de texto, estudos sobre emoções e sentimentos em linguagem natural e para compreender como as palavras influenciam as perceções e experiências dos indivíduos. Este dicionário é composto por 13000 palavras da língua inglesa.

Em todos os casos, exceto quando fazemos uso de bibliotecas externas, foi-nos fornecido documentos com informação necessária para fazer o processamento de cada feature.

3. Trabalho realizado

3.1. LMER-FE – Main Window

É responsável por criar a janela principal do programa usando a biblioteca PyQt5[16]. A janela contém vários botões que permitem aceder diferentes funcionalidades do programa.

O botão "Content-Based Features" abre uma nova janela que permite escolher duas funcionalidades ao nível do conteúdo lírico da letra musical.

O botão "Features Stylistics" abre uma nova janela que permite escolher três funcionalidades ao nível estilístico da letra musical.

O botão "Structural Based Features" abre uma nova janela que permite escolher duas funcionalidades ao nível estrutural da letra musical.

O botão "Features Semantics" abre uma nova janela que permite escolher sete funcionalidades ao nível semântico da letra musical.

Ao clicar no botão "Choose a folder" é exibida uma caixa de diálogo de seleção de pasta para o utilizador. Essa caixa de diálogo é fornecida pela classe QFileDialog [17] da biblioteca PyQt5. A caixa de diálogo apresenta uma janela na qual o utilizador pode navegar pelas pastas do sistema de ficheiros e escolher a pasta desejada.

Quando o utilizador seleciona uma pasta, o caminho absoluto dessa pasta é retornado como resultado. Esse caminho é armazenado na variável pasta_selecionada.

Em seguida, a função salvar_string é chamada para guardar o caminho da pasta selecionada num ficheiro de texto chamado "caminho_pasta.txt". Essa função abre o ficheiro em modo de escrita ("w") e escreve o caminho da pasta no ficheiro.

Desta forma, a função "Escolher Pasta" permite que o utilizador selecione uma pasta específica e guarde o caminho dessa pasta para utilização posterior no programa. Isso é útil,

[16] biblioteca Python que permite utilizar o conjunto de ferramentas de interface gráfica (GUI) multiplataforma.

[17] permite que o utilizador navegue pelo sistema de ficheiros para selecionar um ou vários ficheiros ou um diretório.

por exemplo, quando é necessário aceder a ficheiros ou realizar análises nos ficheiros contidos nessa pasta selecionada.

Ao iniciar o programa pela primeira vez, este deve ser o primeiro botão a ser clicado de modo a garantir que trabalhamos com a pasta desejada.

Essa interface intuitiva e os recursos fornecidos pelos botões permitem que os utilizadores realizem uma análise detalhada e abrangente de texto por meio do programa. Cada funcionalidade é implementada numa janela separada, garantindo uma experiência organizada e eficiente.

3.2.Bibliotecas utilizadas:

Para o desenvolvimento deste projeto utilizámos as seguintes bibliotecas:

3.2.1. PyQt5

A biblioteca PyQt5 contém um conjunto de ferramentas que possibilita a criação de interfaces gráficas para aplicações Python. Desenvolvida por Riverbank Computing em 2016, é uma versão Python da biblioteca original Qt, que é escrita em C++. A combinação da flexibilidade do Python com as capacidades gráficas poderosas do Qt torna a PyQt5 uma escolha popular para programadores que procuram criar aplicações visualmente atraentes e altamente funcionais.

Para o nosso projeto utilizámos esta biblioteca para criar toda a interface gráfica para o utilizador desde as janelas da aplicação, como botões, menus, etc.

3.2.2. Sys

A biblioteca Sys é um modulo integrado da linguagem de programação Python que oferece acesso a funcionalidades e informações específicas do sistema. Permite que os programadores interajam com o código que está a ser executado, permite obter informações sobre o sistema e lida com recursos do sistema operativo.

Esta faz parte da biblioteca padrão do Python, logo não é necessário instalá-la separadamente.

3.2.3. Csv

É um modulo integrado da linguagem python que fornece funcionalidades para a leitura e escrita de ficheiros CSV (Comma-Separated Values) que é um formato utilizado para armazenar dados em tabelas de texto simples.

Utilizámos esta biblioteca porque prima pela sua simplicidade e facilidade de ser lida e manipulada consoante as nossas necessidades.

3.2.4. Nltk

É uma biblioteca de código aberto em Python que fornece ferramentas e recursos para o processamento de linguagem natural (NLP). Foi desenvolvida para ajudar os programadores a trabalhar com texto e linguagem de maneira eficaz e eficiente.

Esta biblioteca necessita de instalação: pip install nltk

Para o nosso projeto utilizámos as seguintes funcionalidades da biblioteca:

- Word_tokenize: é usada para dividir um texto em palavras individuais, denominadas de tokens. Cada *token* é geralmente corresponde a uma palavra. No nosso caso esta função foi fundamental para desenvolver todas as features.
- Pos_tag: é um etiquetador que utiliza o algoritmo do percepton médio para realizar a marcação de partes de discurso (Part-of-Speech) em textos. Atribui rótulos a cada palavra, identificando a sua classe gramatical, como substantivo, verbo, adjetivo, etc.
- *Stopwords*: é uma funcionalidade que permite lidar com palavras consideradas irrelevantes para a análise de texto devido à sua alta frequência e baixo valor informativo.
- Ngrams: é uma funcionalidade que permite efetuar a divisão do texto em conjuntos de uma, duas, três ou mais palavras.
- *Porter Stemmer*: é uma funcionalidade que permite efetuar o *stemming* a uma palavra reduzindo-a à sua raiz.

3.2.5. Os

É uma biblioteca incorporada na linguagem Python que fornece ferramentas para interagir com funcionalidades do sistema operativo. Permite que haja acesso e gestão de ficheiros e diretórios e manipulação de caminhos de ficheiros.

3.2.6. LIWC

É uma ferramenta utilizada para analisar e quantificar aspetos linguísticos e emocionais presentes num texto.

3.2.7. NRCLex

É uma ferramenta que se baseia no dicionário "NRC Emotion Lexicon" para analisar textos e identificar palavras associadas a diferentes emoções

3.2.8. Re

A biblioteca "re" permite realizar operações de pesquisa, correspondência e manipulação de *strings* com base em padrões específicos definidos por expressões regulares.

3.2.9. Counter

A biblioteca “Counter“ é um módulo que faz parte da biblioteca padrão do Python e oferece uma estrutura de dados chamada “Counter". Esta é uma subclasse de um dicionário (dict) que é usada para contar elementos numa coleção, como uma lista ou uma *string*. Permite a contagem da frequência de ocorrência de elementos individuais.

3.2.10. Lyricsgenius

A biblioteca "lyricsgenius" é uma ferramenta que permite aceder e obter letras de músicas a partir do site Genius. Esta é uma plataforma online onde os utilizadores podem contribuir com letras de músicas e também adicionar notas e explicações sobre o significado das letras.

3.3. Content-Based Features - Window

Para a implementação desta feature recorremos a algumas bibliotecas como NLTK[18] para o processamento de texto, como por exemplo a tokenização de texto, remoção de *stopwords* e de *stemming*.

Foram criadas 2 funções externas à função principal:

[18] Natural Language Toolkit

3.3.1. removeStopWords(text)

Que recebe um texto como argumento, efetua a tokenização do texto e através de funcionalidades da biblioteca NLTK já implementadas por outras entidades este remove as stopwords que encontra.

Código:

```
def removeStopWords(self, word_list): # Remove as stop words
    stop_words = set(stopwords.words('english')) # Obtém as stop words
    filtered_words = [word for word in word_list if word.lower() not in stop_words] # Remove as stop words
    return filtered_words # Retorna as palavras sem stop words
```

Figura 2 Função removeStopWords()

Output:

Trata-se de uma função auxiliar, logo não retorna nenhum *output*.

3.3.2. removeStemming(text)

Que recebe um texto como argumento e recorrendo a função da biblioteca NLTK, efetuamos a tokenização e com a utilização da função PorterStemmer reduzimos as palavras à sua forma radical.

Código:

```
def removeStemming(self, word_list):
    stemmer = PorterStemmer() # Cria o stemmer
    stemmed_words = [stemmer.stem(word) for word in word_list] # Remove o stemming
    return stemmed_words # Retorna as palavras sem stemming
```

Figura 3 Função removeStemming()

Output:

Trata-se de uma função auxiliar, logo não retorna nenhum *output*.

3.3.3. confirm():

Esta função é onde se faz todo o tratamento desta feature, processa ficheiros de texto usando as várias opções de n-gramas, processamento de palavras e métodos de contagem e cálculo.

Código:

```python
def confirm(self):
    all_words = []
    files_processed_words = []  # Armazena as palavras processadas de cada arquivo

    n_gram_option = self.group1.checkedButton().property("value")
    count_option = self.group3.checkedButton().property("value")
    processing_option = self.group2.checkedButton().property("value")

    csv_filename = self.create_csv_filename(n_gram_option, processing_option, count_option)


    print(csv_filename)
    print(n_gram_option)
    print(count_option)
    print(processing_option)

    for filename in os.listdir(origem):
        with open(os.path.join(origem, filename), 'r') as f: # abre o ficheiro
            alltext = f.read() # le o ficheiro
            wordFiletoFile = word_tokenize(alltext) # tokeniza o ficheiro
            wordsAllFile = word_tokenize(alltext) # tokeniza o ficheiro
            #convert to lower case
            wordsAllFile = [word.lower() for word in wordsAllFile]
            wordFiletoFile = [word.lower() for word in wordFiletoFile]

            # Processamento das palavras
            processed_words_AllFiles = self.process_words(wordsAllFile, processing_option)
            processed_words_FiletoFile = self.process_words(wordFiletoFile, processing_option)

            files_processed_words.append(processed_words_FiletoFile)  # Armazena palavras processadas do arquivo
            all_words.extend(processed_words_AllFiles)  # Adiciona palavras processadas à lista consolidada

    if n_gram_option == "1":
        ngramsAllFiles = list(ngrams(all_words, 1)) # Cria os n-grams
        ngram_docs = [" ".join(file_words) for file_words in files_processed_words] # Cria os n-grams
        print(ngramsAllFiles)

    elif n_gram_option == "2":
        ngramsAllFiles = list(ngrams(all_words, 2)) # Cria os n-grams
        ngram_docs = [" ".join(file_words) for file_words in files_processed_words] # Cria os n-grams

        print(ngramsAllFiles)

    elif n_gram_option == "3": # Se a opção for 3-gram
        ngramsAllFiles = list(ngrams(all_words, 3)) # Cria os 3-gramas
        ngram_docs = [" ".join(file_words) for file_words in files_processed_words] # Cria os n-grams

        print(ngramsAllFiles)

    else:
        print("Invalid Option")

    if count_option == "freq":
        with open(csv_filename, 'w', newline='', encoding='utf-8') as csvfile: # Cria o ficheiro CSV
            csvwriter = csv.writer(csvfile) # Cria o escritor CSV
            header = ['File'] + ngramsAllFiles # Cria o cabeçalho do CSV
            csvwriter.writerow(header) # Escreve o cabeçalho no CSV

            for filename, file_words in zip(os.listdir(origem), files_processed_words): # Percorre os ficheiros
                if n_gram_option == "1": # Se a opção for 1-gram
                    ngramsFileToFile = list(ngrams(file_words, 1)) # Cria os 1-gramas
                elif n_gram_option == "2": # Se a opção for 2-gram
                    ngramsFileToFile = list(ngrams(file_words, 2)) # Cria os 2-gramas
                elif n_gram_option == "3": # Se a opção for 3-gram
                    ngramsFileToFile = list(ngrams(file_words, 3)) # Cria os 3-gramas
                else:
                    continue

                ngram_counts = Counter(ngramsFileToFile) # Conta os n-gramas
                row_values = [filename] + [ngram_counts[ngram] if ngram in ngram_counts else 0 for ngram in
                                           ngramsAllFiles] # Cria a linha do CSV
                csvwriter.writerow(row_values) # Escreve a linha no CSV

    elif count_option == "tfidf": # Se a opção for TF-IDF
        tfidf_vectorizer = TfidfVectorizer(ngram_range=(1, int(n_gram_option))) # Cria o vetorizador TF-IDF
        tfidf_matrix = tfidf_vectorizer.fit_transform(ngram_docs) # Cria a matriz TF-IDF
        feature_names = tfidf_vectorizer.get_feature_names_out() # Obtém os nomes das features

        with open(csv_filename, 'w', newline='', encoding='utf-8') as csvfile: # Cria o ficheiro CSV
            csvwriter = csv.writer(csvfile) # Cria o escritor CSV
            header = ['File'] + ngramsAllFiles # Cria o cabeçalho do CSV
            csvwriter.writerow(header) # Escreve o cabeçalho no CSV

            for filename, tfidf_row in zip(os.listdir(origem), tfidf_matrix.toarray()): # Percorre os ficheiros
                row_values = [filename] + list(tfidf_row) # Cria a linha do CSV
                csvwriter.writerow(row_values) # Escreve a linha no CSV

    elif count_option == "bool": # Se a opção for booleana
        ngram_presence_data = [] # Armazena a presença dos n-gramas

        for filename, file_words in zip(os.listdir(origem), files_processed_words): # Percorre os ficheiros
            if n_gram_option == "1": # Se a opção for 1-gram
                ngramsFileToFile = list(ngrams(file_words, 1)) # Cria os 1-gramas
            elif n_gram_option == "2": # Se a opção for 2-gram
                ngramsFileToFile = list(ngrams(file_words, 2)) # Cria os 2-gramas
            elif n_gram_option == "3": # Se a opção for 3-gram
                ngramsFileToFile = list(ngrams(file_words, 3)) # Cria os 3-gramas
            else:
                continue

            ngram_presence = [1 if ngram in ngramsFileToFile else 0 for ngram in ngramsAllFiles] # Verifica a presença dos n-gramas
            ngram_presence_data.append([filename] + ngram_presence) # Adiciona a presença dos n-gramas à lista

        with open(csv_filename, 'w', newline='', encoding='utf-8') as csvfile: # Cria o ficheiro CSV
            csvwriter = csv.writer(csvfile) # Cria o escritor CSV
            header = ['File'] + ngramsAllFiles # Cria o cabeçalho do CSV
            csvwriter.writerow(header) # Escreve o cabeçalho no CSV

            for row in ngram_presence_data: # Percorre as linhas
                csvwriter.writerow(row) # Escreve a linha no CSV
    else:
        print("Invalid Option")
```

Figura 4 Código Confirm

Funcionamento:

1. O código começa por inicializar várias variáveis e extrair valores de grupos de *radiobuttons* (group1, group2 e group3) relacionados com opções de n-grams, opções de processamento de palavras e opções de contagem.

2. A função então constrói o nome do ficheiro CSV com base nas opções selecionadas. (eg. 1_st_bool, unigramas, stemming, boolean, respetivamente).

3. O código itera sobre ficheiros num diretório (assumindo que a variável origem contém o caminho para o diretório). Para cada ficheiro:

    a. Lê o conteúdo do ficheiro e divide-o em palavras.

    b. Converte as palavras para minúsculas.

    c. Aplica uma função de processamento de palavras (self.process_words) para processar as palavras.

    d. Armazena as palavras processadas para futura utilização.

4. Dependendo da opção de n-grams selecionada, o código gera n-grams (unigramas, bigramas ou trigramas) para todas as palavras processadas em todos os ficheiros.

5. O código então lida com diferentes opções de contagem (freq, tfidf ou bool):

    a. Para a opção freq, ele conta as ocorrências de n-grams em cada ficheiro e escreve as contagens no ficheiro CSV.

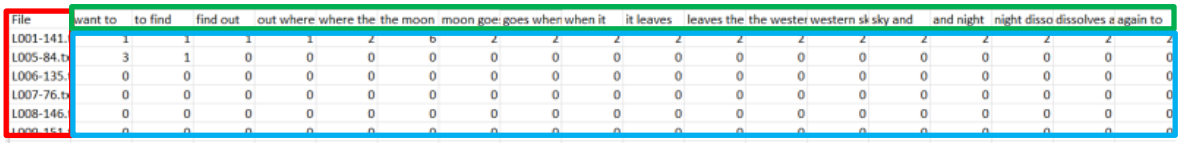
    b. Para a opção tfidf, é utilizado o vetorizador TF-IDF para calcular pontuações TF-IDF para os n-grams e escreve-os no ficheiro CSV.

    c. Para a opção bool, ele verifica a presença de n-grams em cada ficheiro e escreve valores binários de presença num ficheiro CSV. (1 caso esteja presente, 0 caso contrário).

6. O código repete passos semelhantes para cada ficheiro, gerando n-grams apropriados ou valores de contagem e escrevendo-os no ficheiro CSV.

7. Se for escolhida uma opção inválida (quer para n-grams quer para contagem), é impresso "Opção Inválida".

Output por exemplo para bigramas:



File	want to	to find	find out	out where	where the	the moon	moon goe	goes when	when it	it leaves	leaves the	the wester	western sk	sky and	and night	night disso	dissolves a	again to
L001-141.	1	1	1	1	2	6	2	2	2	2	2	2	2	2	2	2	2	2
L005-84.tx	3	1	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L006-135.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L007-76.tx	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L008-146.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
L009-151.	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

[] → Ficheiros a serem analisados

[] → Features extraídas

[] → Valores das features

3.3.4. process_words(self, words, processing_option)

1. A função recebe dois argumentos:
    a. self: uma referência à própria instância da classe que contém essa função.
    b. words: uma lista de palavras para processar.
    c. processing_option: uma string que indica qual tipo de processamento que irá ser aplicado.
2. A função começa por inicializar a lista processed_words com a lista de palavras original palavras.
3. A função verifica o valor de processing_option para determinar qual o tipo de processamento que será aplicado:
    a. Se processing_option for "st": A função chama o método self.removeStemming(words).
    b. Se processing_option for "sw": A função chama o método self.removeStopWords(palavras).
    c. Se processing_option for "st_sw": A função executa primeiro o stemming utilizando self.removeStemming(palavras) e, em seguida, aplica a remoção de stop words usando self.removeStopWords().

4. Depois de realizar os passos de processamento, a função remove a pontuação de cada palavra.
5. Finalmente, a função retorna a lista de processed_words, que é a lista resultante após o processamento das palavras.

Output:

Trata-se de uma função auxiliar, logo não retorna nenhum output.

3.4. Stylistic Features

3.4.1. capitalLetters:

Esta feature foi implementada para conseguirmos efetuar a contagem de quantas letras maiúsculas existem em cada ficheiro de texto presente na pasta base_dir.

Código:

```
def capitalLetters(self):
        try:
            csv_filename = os.path.join(project_dir, "Output", "CapitalLetters.csv") #criar o ficheiro csv
            files_in_dir = listdir(base_dir) #listar todos os ficheiros na pasta
            capital_Letteres_counts = {} #criar um dicionario para guardar o nome do ficheiro e o numero de capital letters
            for file in files_in_dir: #percorrer todos os ficheiros
                filepath = nltk.data.find(base_dir + file) #encontrar o caminho do ficheiro
                textfile = open(filepath, 'r').read() #abrir o ficheiro
                count = sum(1 for c in textfile if c.isupper()) #contar o numero de capital letters
                capital_Letteres_counts[file] = count #guardar o nome do ficheiro e o numero de capital letters
                with open(csv_filename, 'w') as f: #guardar no ficheiro csv
                    f.write("%s,%s \n" % ("File", "Capital Letters")) #escrever o header do ficheiro csv
                    for key in capital_Letteres_counts.keys(): #percorrer o dicionario
                        f.write("%s,%s\n" % (key, capital_Letteres_counts[key])) #escrever no ficheiro csv
                print("File: ", file, " has ", count, " capital letters") #print o numero de capital letters
            QMessageBox.information(self, "Success", "CapitalLetters.csv successfully created in the Output folder.")
        except Exception as e:
            QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 5 Código Capital Letters

Funcionamento:

1. A função começa com um bloco "try-except", que é usado para lidar com exceções que podem ocorrer durante a execução do código.

2. A variável "csv_filename" é criada para armazenar o caminho completo e o nome do ficheiro CSV que será gerado após a contagem das letras maiúsculas. O ficheiro CSV irá conter as informações do nome do ficheiro e o número de letras maiúsculas.

3. A função lista todos os ficheiros presentes no diretório especificado pela variável "base_dir" usando a função "listdir" da biblioteca "os". Isto permite que o código percorra cada ficheiro para realizar a contagem.

4. É criado um dicionário chamado "capital_Letteres_counts" para armazenar o nome do ficheiro como chave e o número de letras maiúsculas como valor correspondente.

5. Para cada ficheiro presente na lista do diretório, o código realiza as seguintes ações:

    a) Obtém o caminho completo do ficheiro usando "nltk.data.find" e a variável "base_dir + file".

    b) Lê o conteúdo do ficheiro de texto e armazena-o na variável "textfile".

    c) Conta o número de letras maiúsculas no texto usando a função "sum" numa expressão que percorre cada caractere "c" no texto e verifica se ele é maiúsculo usando "c.isupper()".

    d) Armazena o nome do ficheiro e o número de letras maiúsculas no dicionário "capital_Letteres_counts".

    e) Abre o ficheiro CSV em modo de escrita ("w") usando "open" com a variável "csv_filename" e escreve o cabeçalho do ficheiro CSV, que contém as informações "File", "Capital Letters”.

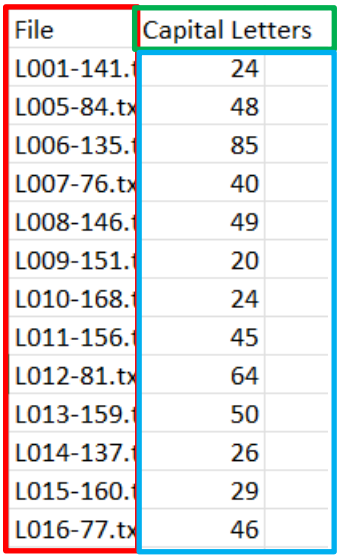
    f) Percorre o dicionário "capital_Letteres_counts" e escreve no ficheiro CSV o nome do ficheiro e o número de letras maiúsculas correspondente a cada chave e valor do dicionário.

6. Após a contagem das letras maiúsculas em todos os ficheiros, uma message box é exibida, indicando que o ficheiro CSV foi criado com sucesso na pasta "Output".

7. Caso ocorra algum erro durante a execução do código, a exceção será capturada pelo bloco "except", e uma caixa de diálogo de erro será exibida, mostrando a mensagem de

erro específica. E.g: caso o ficheiro não seja encontrado, ou caso haja algum erro na escrita do csv.

Output:



File	Capital Letters
L001-141.t	24
L005-84.tx	48
L006-135.t	85
L007-76.tx	40
L008-146.t	49
L009-151.t	20
L010-168.t	24
L011-156.t	45
L012-81.tx	64
L013-159.t	50
L014-137.t	26
L015-160.t	29
L016-77.tx	46

Figura 6 Output Capital Letters

[] → Ficheiros a serem analisados

[] → Features extraídas

[] → Valores das features

3.4.2. ACL:

Esta feature foi implementada para conseguirmos efetuar a contagem de quantas palavras inteiramente maiúsculas existem em cada ficheiro de texto presente na pasta base_dir.

Código:

```python
def ACL(self):
    try:
        csv_filename = os.path.join(project_dir, "Output", "ACL.csv") # criar o ficheiro csv
        files_in_dir = listdir(base_dir) # listar todos os ficheiros na pasta
        ACL_counts = {} # criar um dicionario para guardar o nome do ficheiro e o numero de capital letters
        words_acl = [] # criar uma lista para guardar as palavras em maiusculas

        for file in files_in_dir: # percorrer todos os ficheiros
            filepath = nltk.data.find(base_dir + file) # encontrar o caminho do ficheiro
            textfile = open(filepath, 'r').read() # abrir o ficheiro
            words = re.findall(r'\b[A-Z]+\b', textfile)
            words_acl.append(words) # guardar as palavras em maiusculas
            count = len(words) # contar o numero de palavras em maiusculas
            ACL_counts[file] = count # guardar o nome do ficheiro e o numero de palavras em maiusculas
            with open(csv_filename, 'w') as f:
                f.write("%s,%s \n" % ("File", "All Capital Letters")) # escrever o header do ficheiro csv
                for key in ACL_counts.keys(): # percorrer o dicionario
                    f.write("%s,%s\n" % (key, ACL_counts[key])) # escrever no ficheiro csv
            print("File:", file, "has", count, "words entirely in uppercase") # print o numero de palavras em maiusculas
        QMessageBox.information(self, "Success", "ACL.csv successfully created in the Output folder.")
    except Exception as e:
        QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 7 Código ACL

Funcionamento:

1. A variável csv_filename é criada para armazenar o caminho completo e o nome do ficheiro CSV que irá ser gerado após a contagem das palavras em maiúsculas. O ficheiro CSV irá conter as informações do nome do ficheiro e o número de palavras em maiúsculas de cada ficheiro.

2. A função lista todos os ficheiros presentes na pasta especificada pela variável base_dir usando a função listdir da biblioteca os. Isto permitirá que o código percorra cada ficheiro para realizar a contagem.

3. É criado um dicionário chamado ACL_counts para armazenar o nome do ficheiro como chave e o número de palavras em maiúsculas como valor correspondente.

4. É criada uma lista chamada words_acl para armazenar todas as palavras em maiúsculas encontradas em cada ficheiro. Essa lista é utilizada apenas para fins informativos e não é usada diretamente na contagem.

5. Para cada ficheiro presente na lista de ficheiros da pasta, o código realiza as seguintes ações:

a) Obtém o caminho completo do ficheiro usando nltk.data.find e a variável base_dir + file.

b) Lê o conteúdo do ficheiro de texto e armazena-o na variável textfile.

c) Utiliza a expressão regular re.findall(r'\b[A-Z]+\b', textfile) para encontrar todas as palavras em maiúsculas no texto. Essas palavras são armazenadas na lista words.

d) Conta o número de palavras em maiúsculas encontradas e armazena esse valor no dicionário ACL_counts com o nome do ficheiro correspondente.

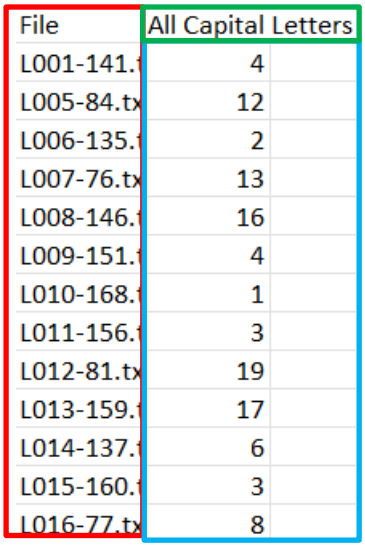
e) Abre o ficheiro CSV em modo de escrita ("w") usando open com a variável csv_filename e escreve o cabeçalho do ficheiro CSV, que contém as informações "File" e "All Capital Letters".

f) Percorre o dicionário ACL_counts e escreve no ficheiro CSV o nome do ficheiro e o número de palavras em maiúsculas correspondente a cada chave e valor do dicionário.

6. Após a contagem das palavras em maiúsculas de todos os ficheiros, uma mensagem de informação é exibida, indicando que o ficheiro CSV foi criado com sucesso na pasta "Output".

7. Caso ocorra algum erro durante a execução do código, a exceção será capturada pelo bloco "except", e uma caixa de diálogo de erro será exibida, mostrando a mensagem de erro específica. E.g: caso o ficheiro não seja encontrado, ou a lista estiver vazia.

Output:



File	All Capital Letters
L001-141.t	4
L005-84.tx	12
L006-135.t	2
L007-76.tx	13
L008-146.t	16
L009-151.t	4
L010-168.t	1
L011-156.t	3
L012-81.tx	19
L013-159.t	17
L014-137.t	6
L015-160.t	3
L016-77.tx	8

Figura 8 Output ACL

[] → Ficheiros a serem analisados

[] → Features extraídas

[] → Valores das features

3.4.3. slangWords:

Essa função é responsável por contar quantas palavras de gíria (slang) existem em cada ficheiro de texto presente na pasta base_dir.

Código:

```python
def slangWords(self):
    try:
        slang_File = os.path.join(project_dir, "src", "auxiliarFiles", "slang.txt")
        csv_filename = os.path.join(project_dir, "Output", "SlangWords.csv") #criar o ficheiro csv
        files_in_dir = listdir(base_dir) #listar todos os ficheiros na pasta
        slangWords = [] #criar uma lista para guardar as palavras em gíria
        with open(slang_File, 'r') as slangFile: #abrir o ficheiro com as palavras em gíria
            for line in slangFile: #percorrer o ficheiro
                slangWords.append(line.strip()) #guardar as palavras em gíria na lista
        with open(csv_filename, 'w', newline='') as csvfile: #criar o ficheiro csv
            csvwriter = csv.writer(csvfile)  # criar o writer
            csvwriter.writerow(['File', 'Slang Words'])

            for file in files_in_dir:   #percorrer todos os ficheiros
                filepath = nltk.data.find(base_dir + file) #encontrar o caminho do ficheiro
                textfile = open(filepath, 'r').read() #abrir o ficheiro
                count = 0 #inicializar o contador

                for word in textfile.split(): #percorrer todas as palavras do ficheiro
                    if word in slangWords: #verificar se a palavra está na lista de palavras em gíria
                        count += 1 #incrementar o contador

                csvwriter.writerow([file, count]) #escrever no ficheiro csv
        QMessageBox.information(self, "Success", "SlangWords.csv successfully created in the Output folder.")
    except Exception as e:
        QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 9 Código slangWords

Funcionamento:

1. A variável slang_File é criada para armazenar o caminho completo do ficheiro de texto que contém as palavras em gíria (slang).

2. A variável csv_filename é criada para armazenar o caminho completo e o nome do ficheiro CSV que será gerado após a contagem das palavras em gíria. O ficheiro CSV armazena as informações do nome do ficheiro e o número de palavras em gíria de cada ficheiro.

3. A função lista todos os ficheiros presentes na pasta especificada pela variável base_dir utilizando a função listdir da biblioteca os. Isso permitirá que o código percorra cada ficheiro para realizar a contagem.

4. É criada uma lista chamada slangWords para armazenar todas as palavras em gíria presentes no ficheiro de texto especificado pelo variável slang_File. As palavras em gíria são lidas do ficheiro de texto e adicionadas à lista após retirar os espaços em branco usando line.strip().

5. O código cria o ficheiro CSV "SlangWords.csv" na pasta "Output" e escreve o cabeçalho do ficheiro com as informações "File" e "Slang Words".

6. Para cada ficheiro presente na lista de ficheiros da pasta, o código realiza as seguintes ações:

    a) Obtém o caminho completo do ficheiro usando nltk.data.find e a variável base_dir + file.

    b) Lê o conteúdo do ficheiro de texto e armazena-o na variável textfile.

    c) Inicializa um contador chamado count a zero.

    d) Percorre todas as palavras do ficheiro usando textfile.split().

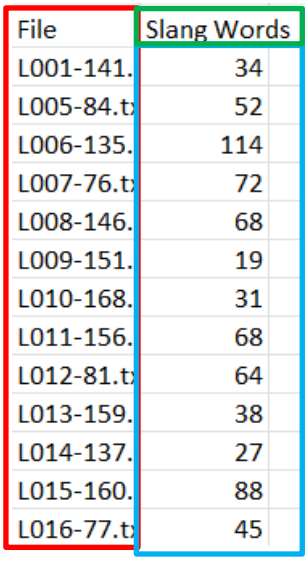
    e) Verifica se a palavra está presente na lista de palavras em gíria slangWords. Se estiver, incrementa o contador count.

    f) Escreve o nome do ficheiro e o valor do contador no ficheiro CSV usando csvwriter.writerow([file, count]).

7. Após a contagem das palavras em gíria em todos os ficheiros, uma mensagem de informação é exibida, indicando que o ficheiro CSV foi criado com sucesso na pasta "Output".

8. Caso ocorra algum erro durante a execução do código, a exceção será capturada pelo bloco "except", e uma caixa de diálogo de erro será exibida, mostrando a mensagem de erro específica. E.g: caso o ficheiro não seja encontrado, ou a lista estiver vazia.

Output:



File	Slang Words
L001-141.	34
L005-84.t	52
L006-135.	114
L007-76.t	72
L008-146.	68
L009-151.	19
L010-168.	31
L011-156.	68
L012-81.t	64
L013-159.	38
L014-137.	27
L015-160.	88
L016-77.t	45

Figura 10 Output SlangWords

[] → Ficheiros a serem analisados

[] → Features extraídas

[] → Valores das features

3.5. Structural Features

3.5.1. contarRepeticoesDoTitulo:

Esta função tem como objetivo contar quantas vezes um título de música se repete em todas as letras musicais presentes na pasta base_dir.

Código:

```python
def contarRepeticoesDoTitulo(self):
    try:
        files_in_dir = listdir(base_dir)  # listar os ficheiros da pasta
        csv_filename = os.path.join(project_dir, "Output", "RepTitles.csv")  # criar o ficheiro csv
        with open(csv_filename, 'w', newline='') as csvfile:  # abrir o ficheiro csv
            csvwriter = csv.writer(csvfile)  # criar o writer
            csvwriter.writerow(["Name of file", "Number of repetitions"])  # escrever o cabeçalho
            for file in files_in_dir:  # percorrer os ficheiros da pasta
                titulo_encontrado = False  # inicializar a variável
                counter = 0  # inicializar o contador
                with open(base_dir + file, "r") as arquivo_musica:  # abrir o ficheiro de música
                    for linha_musica in arquivo_musica:  # percorrer as linhas do ficheiro de música
                        for linha in open(titulos, "r"):  # percorrer as linhas do ficheiro de títulos
                            titulo_partes = linha.strip().split("--")  # separar o título do artista
                            tituloficheiro = titulo_partes[0].strip()  # obter o título do ficheiro
                            tituloMusica = titulo_partes[1].strip()  # obter o título da música

                            if file.strip().lower() == tituloficheiro.strip().lower():  # comparar o título do ficheiro com o título do ficheiro de títulos
                                if tituloMusica in linha_musica.strip():  # verificar se o título da música está na linha da música
                                    counter += 1  # incrementar o contador
                                    titulo_encontrado = True  # atualizar a variável
                                    break  # sair do ciclo

                # Criar uma lista com os valores a serem escritos na linha do ficheiro CSV
                row_data = [file, counter]

                csvwriter.writerow(row_data)  # escrever a lista no ficheiro CSV

        QMessageBox.information(self, "Success", "RepTitles.csv successfully created in the Output folder.")
    except Exception as e:
        QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 11 Código contarRepetiçõesDoTitulo

Funcionamento:

1. A função começa por listar todos os ficheiros presentes na pasta especificada pela variável base_dir usando a função listdir da biblioteca os. Isto permite que o código percorra cada ficheiro para realizar a contagem.

2. Em seguida, é criado o nome completo do ficheiro CSV "RepTitles.csv" na pasta "Output" do projeto. O ficheiro CSV irá conter as informações do nome do ficheiro de música e o número de repetições do título desse ficheiro.

3. O código abre o ficheiro CSV para escrita e escreve o cabeçalho do ficheiro com as informações "Name of file" (Nome do ficheiro) e "Number of repetitions" (Número de repetições).

4. Para cada ficheiro presente na lista de ficheiros da pasta, o código realiza as seguintes ações:

    a) Inicializa uma variável booleana chamada titulo_encontrado com o valor "False". Essa variável será usada para verificar se o título do ficheiro foi encontrado no ficheiro de títulos.

b) Inicializa uma variável counter com o valor "0". Essa variável será usada para contar o número de repetições do título.

c) O código abre o ficheiro de música usando o caminho completo do ficheiro e lê as linhas.

d) Em seguida, o código percorre as linhas do ficheiro de títulos usando for linha in open(titulos, "r").

e) Para cada linha do ficheiro de títulos, o código separa o título do artista usando titulo_partes = linha.strip().split("--").

f) Obtém o título do ficheiro e o título da música usando tituloficheiro = titulo_partes[0].strip() e tituloMusica = titulo_partes[1].strip().

g) O código compara o título do ficheiro de música com o título do ficheiro de títulos, ignorando maiúsculas e minúsculas, usando if file.strip().lower() == tituloficheiro.strip().lower().

h) Se os títulos coincidirem, o código verifica se o título da música está presente em alguma linha do ficheiro. Se estiver, incrementa o contador counter, atualiza a variável titulo_encontrado para "True" e sai do ciclo usando break.

i) Após percorrer todas as linhas do ficheiro de títulos, o código verifica se o título foi encontrado (ou seja, se titulo_encontrado é verdadeiro). Se o título foi encontrado, escreve o nome do ficheiro e o valor do contador no ficheiro CSV usando csvwriter.writerow([file, counter]). Caso contrário, escreve o nome do ficheiro e o valor "0" no ficheiro CSV, indicando que não houve repetição do título.

5. Após contar as repetições do título de todos os ficheiros de música, uma mensagem de informação é exibida, indicando que o ficheiro CSV foi criado com sucesso na pasta "Output".

6. Caso ocorra algum erro durante a execução do código, a exceção será capturada pelo bloco "except", e uma caixa de diálogo de erro será exibida, mostrando a mensagem de erro específica. E.g: caso o ficheiro não seja encontrado, ou a lista estiver vazia.

Output:

Name of fi	Number of repetitions
L001-141.t	0
L005-84.tx	5
L006-135.t	1

Figura 12 Output RepeticoesDoTitulo

[] → Ficheiros a serem analisados

[] → Features extraídas

[] → Valores das features

```
L001-141.txt -- Moon Song
L005-84.txt --  New York, New York
L006-135.txt -- Nightmare
```

Figura 13 Ficheiro titulos.txt

3.5.2. extract_chorus:

Esta função tem como objetivo servir como função auxiliar para a extração do refrão:

Código:

```python
def extract_chorus(self, lyrics):
    if isinstance(lyrics, str): # verificar se o parâmetro é uma string
        lines = lyrics.split("\n") # separar as linhas da letra
        chorus_lines = [] # inicializar a lista
        in_chorus = False # inicializar a variável

        for line in lines: # percorrer as linhas da letra
            line = line.strip() # remover espaços em branco do início e fim da linha

            if line.startswith("[Chorus"): # verificar se a linha começa com [Chorus
                in_chorus = True # atualizar a variável
                continue
            elif line.startswith("["): # verificar se a linha começa com [
                in_chorus = False # atualizar a variável
                continue  # passar para a próxima linha

            if in_chorus and line: # verificar se a variável é verdadeira e se a linha não está vazia
                chorus_lines.append(line) # adicionar a linha à lista

        chorus = "\n".join(chorus_lines) # juntar as linhas da lista numa string
        return chorus # retornar a string
    else:
        return None
```

Figura 14 Código extract_chorus

Funcionamento:

1. Primeiro, verifica se o input lyrics é uma string usando isinstance(lyrics, str). Se não for uma string, a função retorna None.
2. Se lyrics for uma string, ela divide a letra da música em linhas usando lyrics.split("\n") e armazena-as na lista lines.
3. Inicializa uma lista vazia chorus_lines para armazenar as linhas que pertencem ao refrão.
4. Também inicializa uma variável booleana in_chorus para acompanhar se está atualmente a processar a parte do refrão ou não. É definida como false inicialmente.
5. A função itera por cada linha na lista lines:
    a) Se a linha começar com "[Chorus", define in_chorus como True para indicar que a parte do refrão começou e continua para a próxima linha.

b) Se a linha começar com "[" (parêntese quadrado) mas não "[Chorus", define in_chorus como False para indicar que a parte do refrão terminou e continua para a próxima linha.

c) Se in_chorus for True (ou seja, está atualmente a processar a parte do refrão) e a linha não estiver vazia (line), ela adiciona a linha à lista chorus_lines.

6. Após processar todas as linhas, a função junta as linhas na lista chorus_lines usando "\n" como separador para formar o refrão completo como uma string e o retorna.

7. Se o input lyrics não for uma string, a função retorna None.

Output:

Esta função não contém um output porque apenas serve como função auxiliar para a extração do refrão.

3.5.3. extract_chorus_from_song:

A função tem como objetivo extrair o refrão de músicas utilizando informações de um ficheiro chamado “refrao”, que contém os títulos e artistas das músicas. O refrão de cada música é procurado na plataforma Genius [19]através da biblioteca genius, que deve ser importada e estar disponível neste contexto.

Genius.com é uma plataforma online que fornece letras de músicas, anotações e informações sobre músicas e artistas. Possuem uma API pública que utilizámos que nos permite aceder a dados relacionados a letras de músicas.

Código:

[19] https://genius.com

```python
def extract_chorus_from_song(self):
    try:
        csv_filename = os.path.join(project_dir, "Output", "Chorus.csv")  # criar o ficheiro csv
        with open(refrao, "r") as arquivo, open(csv_filename, 'w',
                                                newline='') as csvfile:  # abrir o ficheiro de títulos e o ficheiro csv
            csvwriter = csv.writer(csvfile)  # criar o writer
            csvwriter.writerow(['Nome ficheiro', 'artista', 'nome'])  # escrever o cabeçalho

            for linha in arquivo:  # percorrer as linhas do ficheiro de títulos
                titulo_original = linha.strip()  # Remove espaços em branco do início e fim da linha
                titulo_partes = linha.split("-")  # separar o título do artista
                if len(titulo_partes) >= 4:  # verificar se o título tem 4 partes
                    lyrics_depois_do_hifen2 = titulo_partes[2]  # obter o artista
                    lyrics_depois_do_hifen3 = titulo_partes[3]  # obter o nome da música
                    artista = lyrics_depois_do_hifen2.strip().lower()  # remover espaços em branco do início e fim da linha e converter para minúsculas
                    nome_musica = lyrics_depois_do_hifen3.strip().lower()  # remover espaços em branco do início e fim da linha e converter para minúsculas
                    nomeFicheiro = titulo_partes[0] + "-" + titulo_partes[1]  # obter o nome do ficheiro
                    song = genius.search_song(artista, nome_musica)  # pesquisar a música no Genius

                    if song is not None:  # verificar se a música foi encontrada
                        lyrics = song.lyrics  # obter a letra da música
                        if lyrics:  # verificar se a letra da música foi encontrada
                            chorus = self.extract_chorus(lyrics)  # obter o refrão da música
                            csvwriter.writerow([nomeFicheiro, artista, nome_musica,
                                                ""])  # escrever o nome do ficheiro, o artista e o nome da música no ficheiro csv
                            csvwriter.writerow(['Chorus'])  # escrever o cabeçalho do refrão
                            for line in chorus.split('\n'):  # Escrever cada linha do refrão separadamente
                                csvwriter.writerow([line.strip()])
                            print("Chorus extracted for", nomeFicheiro)
                        else:
                            print("Lyrics not found for the song.")
                    else:
                        print("Song not found.")
                else:
                    print("Song not found.")
        QMessageBox.information(self, "Success", "Chorus.csv successfully created in the Output folder.")
    except Exception as e:
        QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 15 Código extract_chorus_from_song

Funcionamento:

1. Define o nome do ficheiro CSV como "Chorus.csv" na pasta de saída especificada em project_dir.
2. Abre o ficheiro refrao e o ficheiro CSV csv_filename usando o contexto de gestão de ficheiros com open.
3. Cria um objeto csvwriter para escrever no ficheiro CSV e escreve o cabeçalho contendo as colunas "Nome ficheiro", "artista" e "nome".
4. Itera por cada linha do ficheiro refrão.
5. Separa cada linha em partes usando o caracter "-" como separador, assumindo que cada linha contém informações do título e artista da música.
6. Verifica se a linha tem pelo menos 4 partes (nome do ficheiro, título, artista e nome da música). Se não tiver, imprime na consola "Música não encontrada." e continua para a próxima linha.

7. Obtém o artista e o nome da música a partir das partes correspondentes, removendo espaços em branco no início e no fim e convertendo para minúsculas.
8. Combina o nome do ficheiro a partir das duas primeiras partes.
9. Usa a função genius.search_song(artista, nome_musica) para pesquisar a música no Genius e armazena o resultado em song.
10. Verifica se a música foi encontrada no Genius. Se não, imprime "Letras não encontradas para a música." e continua para a próxima linha.
11. Se a música for encontrada, obtém a letra da música usando song.lyrics e armazena-a em lyrics.
12. Verifica se a letra da música foi encontrada. Se não, imprime "Refrão não encontrado para a música." e continua para a próxima linha.
13. Se a letra da música for encontrada, chama a função extract_chorus(lyrics) para extrair o refrão da música.
14. Escreve o nome do ficheiro, artista e nome da música no ficheiro CSV usando csvwriter.writerow.
15. Escreve o cabeçalho "Chorus" no ficheiro CSV para indicar o início da secção de refrão.
16. Escreve o refrão da música no ficheiro CSV.
17. Repete o processo para cada linha no ficheiro refrão.
18. Fecha o ficheiro CSV e mostra uma mensagem de sucesso em caso de sucesso ou uma mensagem de erro em caso de exceção.

Output:

```
Searching for "linkin park" by numb...
Done.
```

Figura 16 Output na Cmd do refrão

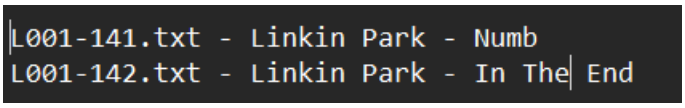


Figura 17 Pasta onde está armazenado o ficheiro "refrão"

Ficheiro “refrão”, este ficheiro deve ter sempre o mesmo formato (Nome ficheiro – Nome artista – Nome música) caso contrário a API não nos conseguirá fornecer a informação.

Ficheiro “Chorus.csv”

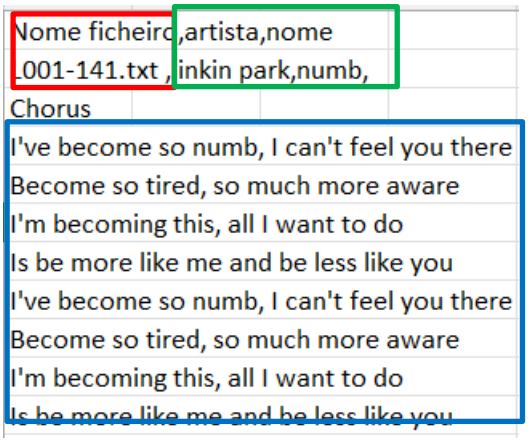


Figura 18 Output Chorus.csv

[] → Ficheiros a serem analisados

[] → Dados utilizados pela API

[] → Refrão da música

3.6. Semantic Features

3.6.1. NRCLex:

A feature NRCLex é uma biblioteca de NLP projetada para fornecer recursos semânticos, como a valência afetiva e a polaridade de palavras em textos.

O processo é realizado para cada ficheiro de letra de música presente na pasta especificada.

O texto de cada ficheiro é analisado utilizando funções da biblioteca, que permite identificar as principais emoções presentes no texto.

Os resultados são guardados num ficheiro CSV chamado "NRCLex.csv", contendo o nome do ficheiro de letra de música e as principais emoções identificadas ('Fear', 'Anger', 'Anticip', 'Trust', 'Surprise', 'Positive', 'Negative', 'Sadness', 'Disgust', 'Joy') e a frequência das palavras de cada emoção presente na letra.

Código:

```
def featuresNRCLex(self):
    try:
        print("Features NRCLex")
        files_in_dir = listdir(base_dir) # Lista de arquivos no diretório
        csv_filename = os.path.join(project_dir, "Output", "NRCLex.csv") # Nome do arquivo .csv
        with open(csv_filename, 'w', newline='') as csvfile:
            csvwriter = csv.writer(csvfile) # Cria um objeto csvwriter
            csvwriter.writerow(['Ficheiro', 'Fear', 'Anger', 'Anticip', 'Trust', 'Surprise', 'Positive', 'Negative', 'Sadness', 'Disgust', 'Joy']) # Escreve o cabeçalho do arquivo .csv
            for file in files_in_dir:  # Percorre todos os arquivos no diretório
                filepath = os.path.join(base_dir, file) # Caminho do arquivo
                textfile = open(filepath, 'r').read() # Lê o arquivo

                # Criar um objeto NRCLex para analisar o texto
                sentiment_obj = NRCLex(textfile) # Cria um objeto NRCLex

                emotions = sentiment_obj.affect_frequencies # Obter as emoções
                # Aceder diretamente aos valores das emoções
                fear_value = emotions['fear']
                anger_value = emotions['anger']
                anticip_value = emotions['anticip']
                trust_value = emotions['trust']
                surprise_value = emotions['surprise']
                positive_value = emotions['positive']
                negative_value = emotions['negative']
                sadness_value = emotions['sadness']
                disgust_value = emotions['disgust']
                joy_value = emotions['joy']

                # Escrever no arquivo CSV
                csvwriter.writerow(
                    [file, fear_value, anger_value, anticip_value, trust_value, surprise_value, positive_value,
                     negative_value, sadness_value, disgust_value, joy_value])
        QMessageBox.information(self, "Success", "NRCLex.csv successfully created in the Output folder.")
    except Exception as e:
        QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 19 Código NRCLex

Funcionamento:

1. A função começa com um bloco "try-except", que é usado para lidar com exceções que podem ocorrer durante a execução do código.
2. É exibida uma mensagem na consola (print) informando que a análise das features NRCLex vai começar.
3. De seguida, a função lista todos os ficheiros presentes no diretório especificado pela variável "base_dir" usando a função "listdir" da biblioteca "os". Isso é feito para que o código possa percorrer cada ficheiro e realizar a análise.
4. A variável "csv_filename" é criada para armazenar o caminho completo e nome do ficheiro CSV que será gerado após a análise dos textos. O ficheiro CSV irá conter as informações emocionais extraídas de cada ficheiro de texto.
5. A função abre o ficheiro CSV no modo de escrita ('w') usando a biblioteca "csv" e cria um objeto "csvwriter" para escrever os dados.
6. Uma linha de cabeçalho é escrita no ficheiro CSV, indicando as colunas que conterão as informações emocionais de cada ficheiro ("Ficheiro", "Fear", "Anger", "Anticip", "Trust", "Surprise", "Positive", "Negative", "Sadness", "Disgust" e "Joy").
7. Em seguida, um loop é utilizado para percorrer cada ficheiro presente na lista de ficheiros "files_in_dir". Para cada ficheiro, os seguintes passos são realizados:

a. É criado o caminho completo para o ficheiro de texto usando o diretório base e o nome do ficheiro atual.
b. O conteúdo do ficheiro de texto é lido e armazenado na variável "textfile".
c. É criado um objeto NRCLex chamado "sentiment_obj" para analisar o conteúdo emocional do texto.
d. O método "affect_frequencies" do objeto NRCLex é usado para obter um dicionário contendo as frequências das emoções presentes no texto.
e. As frequências das diferentes emoções são extraídas do dicionário e armazenadas em variáveis separadas (exemplo: "fear_value" armazenará a frequência da emoção "fear" no texto).
f. As informações são escritas no ficheiro CSV usando o método "writerow" do objeto "csvwriter". Cada linha do ficheiro conterá o nome do ficheiro atual e as frequências das emoções encontradas.

8. Após percorrer todos os ficheiros e escrever as informações no ficheiro CSV, uma caixa de diálogo de informação é exibida, indicando que o ficheiro CSV foi criado com sucesso na pasta "Output".
9. Caso ocorra algum erro durante a execução do código, a exceção será capturada pelo bloco "except", e uma caixa de diálogo de erro será exibida, mostrando a mensagem de erro específica. E.g: caso o ficheiro não seja encontrado.

Output:

Ficheiro	Fear	Anger	Anticip	Trust	Surprise	Positive	Negative	Sadness	Disgust	Joy
L001-141.txt	0.214285714	0.0	0.0	0.0	0.0	0.285714285	0.142857	0.142857142	0.0	0.0
L005-84.txt	0.111111111	0.0	0.0	0.111111	0.0	0.166666666	0.166666	0.166666666	0.0	0.0
L006-135.txt	0.207692307	0.1384615	0.0	0.030769	0.0	0.069230769	0.269230	0.107692307	0.092307692	0.023076923076923078
L007-76.txt	0.078947368	0.0526315	0.0	0.026315	0.0	0.236842105	0.184210	0.105263157	0.105263157	0.15789473684210525
L008-146.txt	0.0	0.057692	0.0	0.153846	0.038461538	0.173076923	0.153846	0.076923076	0.038461538	0.15384615384615385
L009-151.txt	0.0	0.0	0.0	0.097560	0.048780487	0.292682926	0.0	0.097560975	0.0	0.24390243902439024
L010-168.txt	0.035714285	0.035714	0.0	0.178571	0.0	0.464285714	0.035714	0.035714285	0.0	0.14285714285714285
L011-156.txt	0.035714285	0.0	0.0	0.392857	0.035714285	0.25	0.0	0.107142857	0.0	0.14285714285714285
L012-81.txt	0.0	0.0	0.0	0.111111	0.111111111	0.333333333	0.166666	0.0	0.0	0.16666666666666666
L013-159.txt	0.055555555	0.055555	0.0	0.111111	0.166666666	0.166666666	0.133333	0.155555555	0.022222222	0.011111111111111112
L014-137.txt	0.046511627	0.046511	0.0	0.069767	0.023255813	0.255813953	0.186046	0.186046511	0.046511627	0.11627906976744186
L015-160.txt	0.15	0.0	0.0	0.0	0.05	0.3	0.2	0.2	0.0	0.05
L016-77.txt	0.102564102	0.153846	0.0	0.076923	0.051282051	0.102564102	0.256410	0.102564102	0.102564102	0.05128205128205128

Figura 20 Output NRCLex

[] → Ficheiros a serem analisados

[] → Features extraídas

[] → Valores das features

3.6.2. FeaturesGazetters:

Para a implementação destas features utilizamos bibliotecas, dicionários de palavras e respetivos valores fornecidas pelo nosso orientador.

O código extrai palavras-chave de gazeteers dos ficheiros presentes num diretório, calcula a valência e o arousal associadas a essas palavras e armazena os dados num ficheiro CSV.

Código:

```
def Gazetteers(self):
  try:
      gazetteer_files = ["Gazetteers.txt","GazQ1-dal.txt", "GazQ2-dal.txt", "GazQ3-dal.txt", "GazQ4-dal.txt"]
      for gazetteer_file in gazetteer_files:
          print(f"Processing {gazetteer_file}")
          gazetteers_file = os.path.join(project_dir, "src", "auxiliarFiles", gazetteer_file)
          files_in_dir = listdir(base_dir)
          csv_filename = os.path.join(project_dir, "Output", f"{gazetteer_file.split('.')[0]}.csv")

          with open(gazetteers_file, 'r') as f:
              wordslist_gazetteers = []
              valence = []
              arousal = []
              for line in f:
                  words = line.split()
                  if len(words) >= 3:
                      wordslist_gazetteers.append(words[0])
                      valence.append(float(words[1]))
                      arousal.append(float(words[2]))

          with open(csv_filename, 'w', newline='') as csvfile:
              csvwriter = csv.writer(csvfile)
              csvwriter.writerow(['File', 'AvgValence', 'AvgArousal'])

              for file in files_in_dir:
                  word_music_list = []
                  filepath = os.path.join(base_dir, file)
                  with open(filepath, 'r') as f:
                      for line in f:
                          wordlist = line.split()
                          if wordlist:
                              word_music_list.extend(wordlist)

                  common_words_count = 0
                  valenciatotal = 0
                  arousaltotal = 0

                  for word in word_music_list:
                      if word in wordslist_gazetteers:
                          index = wordslist_gazetteers.index(word)
                          valenciatotal += valence[index]
                          arousaltotal += arousal[index]
                          common_words_count += 1

                  if common_words_count > 0:
                      valenciatotal /= common_words_count
                      arousaltotal /= common_words_count
                  else:
                      # If there are no common words, set both averages to NaN
                      valenciatotal = np.nan
                      arousaltotal = np.nan

                  row_data = [file, valenciatotal, arousaltotal]
                  csvwriter.writerow(row_data)

      QMessageBox.information(self, "Success", "Gazetteer.csv, GazQ1-dal.csv, GazQ2-dal.csv, GazQ3-dal.csv and GazQ4-dal.csv successfully created in the Output folder.")
  except Exception as e:
      QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 21 Código Gazetteers

Funciomanento:

1. Em primeiro lugar é inicializado uma lista gazetteer_files que contém os nomes dos ficheiros que vamos processar.

2. O script percorre cada gazetteer_file na lista gazetteer_files.
3. É construído o caminho para o ficheiro de gazetteers atual usando os.path.join, com base na variável project_dir e no gazetteer_file atual.
4. Para cada linha no ficheiro de gazetteers atual, o script extrai a palavra, o valor de valência e o valor de arousal.
5. O script constrói o caminho para o ficheiro CSV de saída usando o gazetteer_file atual e escreve uma linha de cabeçalho.
6. Para cada ficheiro em files_in_dir, o script lê o conteúdo do ficheiro e processa cada linha, adicionando as palavras à lista word_music_list.
7. O script calcula então os valores da média de valência e arousal para as palavras em word_music_list. São também incrementados os valores e a contagem de palavras comuns.
8. Se existirem palavras comuns, os valores das médias de valência e arousal são calculadas. Caso contrário, são utilizados valores NaN.
9. O script escreve os resultados (nome do ficheiro, valência média e arousal médio) no ficheiro CSV.
10. Após processar todos os ficheiros de gazetteers e ficheiros presentes no diretório, o script exibe uma mensagem de informação indicando sucesso. Se ocorrer uma exceção em algum ponto, em vez disso é mostrada uma mensagem de erro.

Output:

File	AvgValence	AvgArousal
L001-141.txt	2.821111111111111	1.3955555555555557
L005-84.txt	2.6233333333333335	1.6066666666666667
L006-135.txt	1.5587499999999999	2.57875
L007-76.txt	2.4336363636363636	2.0354545454545456
L008-146.txt	2.84	2.56
L009-151.txt	2.856	2.4739999999999998

Figura 22 Output Gazetteers

[] → Ficheiros a serem analisados

[] → Features extraídas

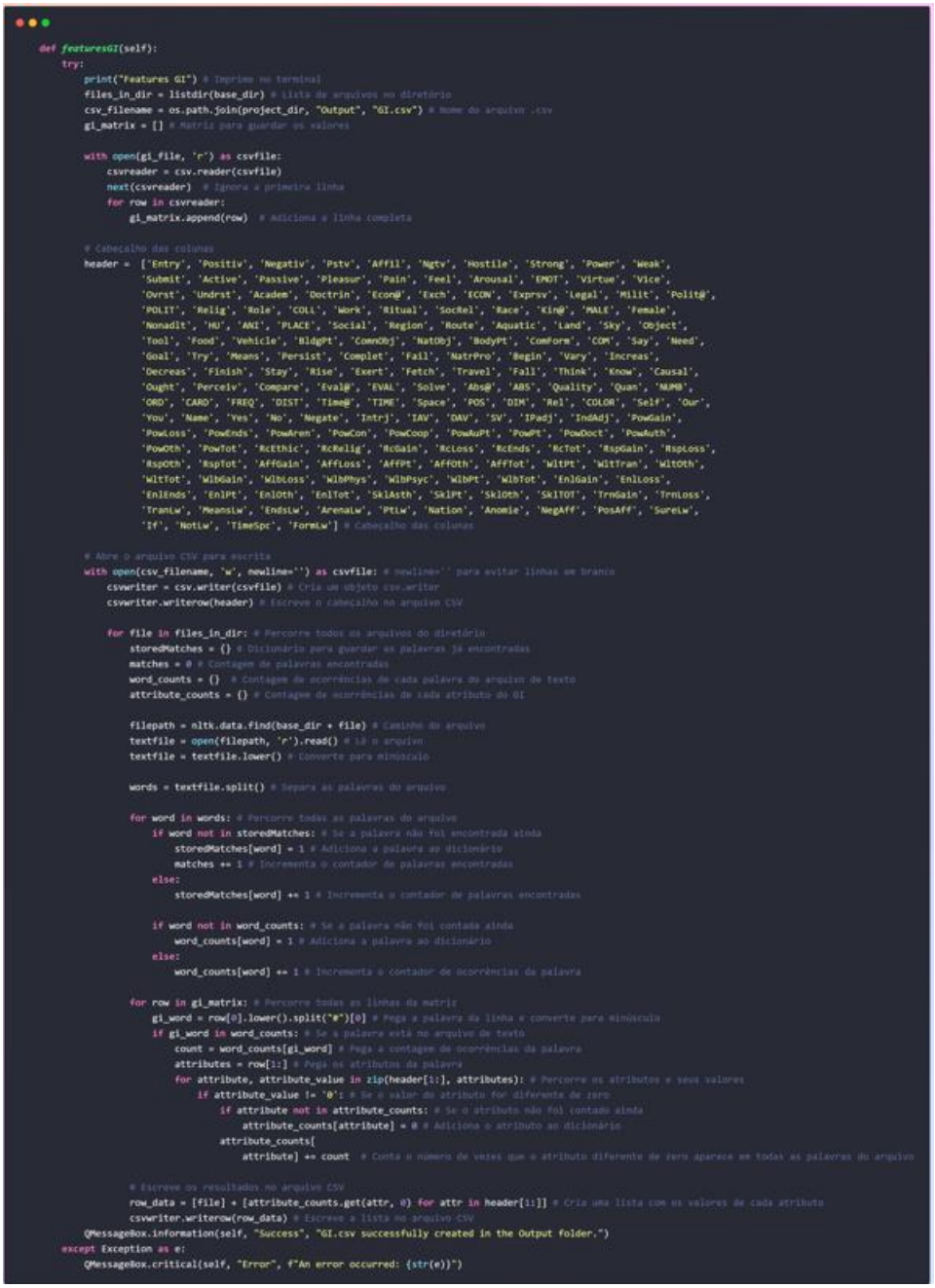
[] → Valores das features

3.6.3. General Inquirer (GI):

O "General Inquirer" é baseado num extenso dicionário léxico-semântico que contém uma grande quantidade de palavras e as suas correspondentes categorias semânticas. Essas categorias abrangem uma variedade de dimensões, incluindo afetividade (emoções positivas ou negativas), potência (poder ou controlo), atividade entre outras.

As categorias que extraímos foram-nos fornecidas e estão presentes no ficheiro “gi-11788.csv”.

Código:



```
def featuresGI(self):
    try:
        print("Features GI") # Imprime no terminal
        files_in_dir = listdir(base_dir) # Lista de arquivos no diretório
        csv_filename = os.path.join(project_dir, "Output", "GI.csv") # Nome do arquivo .csv
        gi_matrix = [] # Matriz para guardar os valores

        with open(gi_file, 'r') as csvfile:
            csvreader = csv.reader(csvfile)
            next(csvreader)  # Ignora a primeira linha
            for row in csvreader:
                gi_matrix.append(row)  # Adiciona a linha completa

        # Cabeçalho das colunas
        header =  ['Entry', 'Positiv', 'Negativ', 'Pstv', 'Affil', 'Ngtv', 'Hostile', 'Strong', 'Power', 'Weak',
                   'Submit', 'Active', 'Passive', 'Pleasur', 'Pain', 'Feel', 'Arousal', 'EMOT', 'Virtue', 'Vice',
                   'Ovrst', 'Undrst', 'Academ', 'Doctrin', 'Econ@', 'Exch', 'ECON', 'Exprsv', 'Legal', 'Milit', 'Polit@',
                   'POLIT', 'Relig', 'Role', 'COLL', 'Work', 'Ritual', 'SocRel', 'Race', 'Kin@', 'MALE', 'Female',
                   'Nonadlt', 'HU', 'ANI', 'PLACE', 'Social', 'Region', 'Route', 'Aquatic', 'Land', 'Sky', 'Object',
                   'Tool', 'Food', 'Vehicle', 'BldgPt', 'ComnObj', 'NatObj', 'BodyPt', 'ComForm', 'COM', 'Say', 'Need',
                   'Goal', 'Try', 'Means', 'Persist', 'Complet', 'Fail', 'NatrPro', 'Begin', 'Vary', 'Increas',
                   'Decreas', 'Finish', 'Stay', 'Rise', 'Exert', 'Fetch', 'Travel', 'Fall', 'Think', 'Know', 'Causal',
                   'Ought', 'Perceiv', 'Compare', 'Eval@', 'EVAL', 'Solve', 'Abs@', 'ABS', 'Quality', 'Quan', 'NUMB',
                   'ORD', 'CARD', 'FREQ', 'DIST', 'Time@', 'TIME', 'Space', 'POS', 'DIM', 'Rel', 'COLOR', 'Self', 'Our',
                   'You', 'Name', 'Yes', 'No', 'Negate', 'Intrj', 'IAV', 'DAV', 'SV', 'IPadj', 'IndAdj', 'PowGain',
                   'PowLoss', 'PowEnds', 'PowAren', 'PowCon', 'PowCoop', 'PowAuPt', 'PowPt', 'PowDoct', 'PowAuth',
                   'PowOth', 'PowTot', 'RcEthic', 'RcRelig', 'RcGain', 'RcLoss', 'RcEnds', 'RcTot', 'RspGain', 'RspLoss',
                   'RspOth', 'RspTot', 'AffGain', 'AffLoss', 'AffPt', 'AffOth', 'AffTot', 'WltPt', 'WltTran', 'WltOth',
                   'WltTot', 'WlbGain', 'WlbLoss', 'WlbPhys', 'WlbPsyc', 'WlbPt', 'WlbTot', 'EnlGain', 'EnlLoss',
                   'EnlEnds', 'EnlPt', 'EnlOth', 'EnlTot', 'SklAsth', 'SklPt', 'SklOth', 'SklTOT', 'TrnGain', 'TrnLoss',
                   'TranLw', 'MeansLw', 'EndsLw', 'ArenaLw', 'PtLw', 'Nation', 'Anomie', 'NegAff', 'PosAff', 'SureLw',
                   'If', 'NotLw', 'TimeSpc', 'FormLw'] # Cabeçalho das colunas

        # Abre o arquivo CSV para escrita
        with open(csv_filename, 'w', newline='') as csvfile: # newline='' para evitar linhas em branco
            csvwriter = csv.writer(csvfile) # Cria um objeto csv.writer
            csvwriter.writerow(header) # Escreve o cabeçalho no arquivo CSV

            for file in files_in_dir: # Percorre todos os arquivos do diretório
                storedMatches = {} # Dicionário para guardar as palavras já encontradas
                matches = 0 # Contagem de palavras encontradas
                word_counts = {}  # Contagem de ocorrências de cada palavra do arquivo de texto
                attribute_counts = {} # Contagem de ocorrências de cada atributo do GI

                filepath = nltk.data.find(base_dir + file) # Caminho do arquivo
                textfile = open(filepath, 'r').read() # Lê o arquivo
                textfile = textfile.lower() # Converte para minúsculo

                words = textfile.split() # Separa as palavras do arquivo

                for word in words: # Percorre todas as palavras do arquivo
                    if word not in storedMatches: # Se a palavra não foi encontrada ainda
                        storedMatches[word] = 1 # Adiciona a palavra ao dicionário
                        matches += 1 # Incrementa o contador de palavras encontradas
                    else:
                        storedMatches[word] += 1 # Incrementa o contador de palavras encontradas

                    if word not in word_counts: # Se a palavra não foi contada ainda
                        word_counts[word] = 1 # Adiciona a palavra ao dicionário
                    else:
                        word_counts[word] += 1 # Incrementa o contador de ocorrências da palavra

                for row in gi_matrix: # Percorre todas as linhas da matriz
                    gi_word = row[0].lower().split("#")[0] # Pega a palavra da linha e converte para minúscula
                    if gi_word in word_counts: # Se a palavra está no arquivo de texto
                        count = word_counts[gi_word] # Pega a contagem de ocorrências da palavra
                        attributes = row[1:] # Pega os atributos da palavra
                        for attribute, attribute_value in zip(header[1:], attributes): # Percorre os atributos e seus valores
                            if attribute_value != '0': # Se o valor do atributo for diferente de zero
                                if attribute not in attribute_counts: # Se o atributo não foi contado ainda
                                    attribute_counts[attribute] = 0 # Adiciona o atributo ao dicionário
                                attribute_counts[
                                    attribute] += count  # Conta o número de vezes que o atributo diferente de zero aparece em todas as palavras do arquivo

                # Escreve os resultados no arquivo CSV
                row_data = [file] + [attribute_counts.get(attr, 0) for attr in header[1:]] # Cria uma lista com os valores de cada atributo
                csvwriter.writerow(row_data) # Escreve a lista no arquivo CSV
        QMessageBox.information(self, "Success", "GI.csv successfully created in the Output folder.")
    except Exception as e:
        QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 23 Código GI

Funcionamento:

1. A função começa com um bloco "try-except", usado para lidar com exceções que podem ocorrer durante a execução do código. Caso ocorra algum erro, a mensagem de erro será exibida numa dialog box.

2. A função imprime uma mensagem na consola indicando que a análise das features GI vai começar.

3. Os ficheiros presentes no diretório especificado pela variável "base_dir" são listados usando a função "listdir" da biblioteca "os". Isto permite que o código percorra cada ficheiro para realizar a análise.

4. A variável "csv_filename" é criada para armazenar o caminho completo e nome do ficheiro CSV que irá ser gerado após a análise dos textos. O ficheiro CSV irá conter as informações de atributos das palavras encontradas nos textos.

5. A variável "gi_matrix" é criada para armazenar a matriz de atributos (features) que está presente no ficheiro CSV chamado "gi_file". O código abre esse ficheiro CSV usando a função "open" e lê as informações usando um objeto "csv.reader". Cada linha é adicionada à matriz "gi_matrix".

6. O cabeçalho das colunas é definido na lista "header". Esse cabeçalho contém os nomes dos atributos (features) que serão analisados.

7. O código então abre o ficheiro CSV para escrita no modo 'w' (modo escrita) usando a biblioteca "csv" e cria um objeto "csvwriter" para escrever os dados.

8. O cabeçalho das colunas é escrito no ficheiro CSV usando o método "writerow" do objeto "csvwriter".

9. Em seguida, o código percorre cada ficheiro presente na lista "files_in_dir" e realiza a análise do texto.

10. Um dicionário chamado "storedMatches" é criado para guardar as palavras já encontradas no ficheiro, para evitar que palavras repetidas sejam contadas várias vezes.

11. A variável "matches" é inicializada como 0 e será usada para contar o número total de palavras encontradas no ficheiro.

12. Dois dicionários são criados: "word_counts" para contar as ocorrências de cada palavra no ficheiro de texto, e "attribute_counts" para contar as ocorrências de cada atributo presente na matriz "gi_matrix" para as palavras encontradas no ficheiro de texto.

13. O código lê o conteúdo do ficheiro, converte todo o texto para letras minúsculas (para tornar a análise de palavras case-insensitive) e separa as palavras numa lista chamada "words".

14. O código, então, percorre cada palavra na lista "words" e realiza o seguinte:

    a) Verifica se a palavra ainda não foi encontrada e adiciona ao dicionário "storedMatches", incrementando o contador "matches".

    b) Se a palavra não foi contada ainda, é adicionada ao dicionário "word_counts" e a sua contagem é iniciada com 1. Caso contrário, a contagem da palavra é incrementada em 1.

15. O código, então, percorre cada linha na matriz "gi_matrix", que contém os atributos (features) para cada palavra.

    a) Para cada palavra na matriz, é verificado se ela está presente no dicionário "word_counts". Se a palavra estiver presente, sua contagem é recuperada da lista "word_counts".

    b) Os atributos da palavra são recuperados da linha atual da matriz "gi_matrix".

    c) O código, então, percorre cada atributo e o seu valor na lista "header[1:]", ignorando a primeira coluna ("Entry") do cabeçalho.

    d) Se o valor do atributo for diferente de '0', significa que o atributo está presente para a palavra e o seu valor tem de ser considerado na contagem.

    e) O atributo é adicionado ao dicionário "attribute_counts" e o seu valor é incrementado com a contagem da palavra multiplicada pelo valor do atributo na matriz "gi_matrix". Isso significa que o valor do atributo é multiplicado pelo número de vezes que a palavra foi encontrada no texto.

16. Após percorrer todas as palavras e calcular a contagem dos atributos, os resultados são escritos no ficheiro CSV.

17. Para cada ficheiro, uma lista chamada "row_data" é criada, que contém o nome do ficheiro seguido da contagem dos atributos (features). A contagem dos atributos é recuperada do dicionário "attribute_counts". Se algum atributo não estiver presente para a palavra, é atribuído o valor 0.

18. A lista "row_data" é escrita no ficheiro CSV usando o método "writerow" do objeto "csvwriter".

19. Após a análise de todos os ficheiros, uma dialog box de informação é mostrada, indicando que o ficheiro CSV foi criado com sucesso na pasta "Output".

20. Caso ocorra algum erro durante a execução do código, a exceção será capturada pelo bloco "except", e uma dialog box de erro será mostrada, contendo a mensagem de erro específica.

Output:

Entry	Positiv	Negativ	Pstv	Affil	Ngtv	Hostile	Strong	Power	Weak	Submit	Active	Passive	Pleasur	Pain	Feel	Arousal	EMOT	Virtue
L001-141	21	2	21	10	2	1	8	1	10	2	26	29	0	0	0	3	0	11
L005-84.t	15	8	17	14	6	9	95	10	9	0	55	18	0	0	0	2	0	5
L006-135	39	73	41	25	67	45	46	12	44	17	64	85	0	7	0	24	14	17
L007-76.t	22	14	22	30	15	10	31	9	24	5	56	32	8	3	0	6	8	6
L008-146	32	9	32	28	9	11	27	11	8	1	61	28	2	1	0	11	5	3
L009-151	8	2	8	3	2	1	4	2	1	0	17	2	1	1	0	1	2	3
L010-168	10	3	9	3	3	2	25	4	5	1	36	16	3	0	0	5	0	2
L011-156	18	5	18	7	4	7	20	10	1	1	28	5	1	3	0	0	1	5
L012-81.t	107	2	107	104	2	0	50	3	6	9	40	116	51	0	0	3	52	23
L013-159	21	11	21	10	11	11	15	14	5	0	40	56	3	1	0	5	8	11
L014-137	15	18	15	17	18	2	19	11	14	0	17	49	6	4	0	0	8	1
L015-160	5	2	5	9	2	6	28	0	5	5	48	24	0	0	0	10	5	3
L016-77.t	30	20	30	17	20	29	30	8	10	13	26	9	0	2	0	21	1	18

Figura 24 Output GI

Nota: Este output apenas contém parte dos atributos, para facilitar a compreensão.

[] → Ficheiros a serem analisados

[] → Features extraídas

[] → Valores das features

3.6.4. DAL_ANEW:

É um recurso lexical utilizado em estudos de NLP e análise de texto. É uma extensão do ANEW (Affective Norms for English Words), que é um conjunto de normas afetivas para palavras em inglês.

A feature DAL_ANEW tem como objetivo analisar as características emocionais das letras de músicas por meio da contagem de palavras presentes no ficheiro "DAL_ANEW.txt".

O processo é realizado para cada ficheiro de letra de música presente na pasta especificada.

Código:

```python
def featuresDAL_ANEW(self):
    try:
        print("Features DAL_ANEW") # Mensagem no console
        files_in_dir = listdir(base_dir) # Lista os arquivos do diretório
        csv_filename = os.path.join(project_dir, "Output", "DAL_ANEW.csv") # Caminho do arquivo CSV

        with open(csv_filename, 'w', newline='') as csvfile: # newline='' para evitar linhas em branco
            csvwriter = csv.writer(csvfile) # Cria um objeto csv.writer
            csvwriter.writerow(['File', 'AvgValence', 'AvgArousal', 'AvgDominance']) # Escreve o cabeçalho no arquivo CSV

            for file in files_in_dir: # Percorre todos os arquivos do diretório
                storedMatches = [] # Lista para guardar as palavras já encontradas
                valenciatotal = 0 # Contagem de valência
                arousaltotal = 0 # Contagem de arousal
                dominanceTotal = 0 # Contagem de dominance

                word_music_list = [] # Lista para guardar as palavras do arquivo de texto
                filepath = nltk.data.find(base_dir + file) # Caminho do arquivo

                with open(filepath, 'r') as f: # Abre o arquivo
                    for line in f: # Percorre todas as linhas do arquivo
                        wordlist = line.split() # Separa as palavras da linha
                        if wordlist: # Se a lista não estiver vazia
                            word_music_list.extend(wordlist) # Adiciona as palavras à lista

                with open(dal_anew_file, 'r') as f: # Abre o arquivo DAL_ANEW
                    wordslist_dal_anew = [] # Lista para guardar as palavras do arquivo
                    valence = [] # Lista para guardar os valores de valência
                    arousal = [] # Lista para guardar os valores de arousal
                    dominance = [] # Lista para guardar os valores de dominance
                    for line in f: # Percorre todas as linhas do arquivo
                        words = line.split() # Separa as palavras da linha
                        if words: # Se a lista não estiver vazia
                            wordslist_dal_anew.append(words[0]) # Adiciona as palavras à lista
                            valence.append(words[1]) # Adiciona os valores de valência à lista
                            arousal.append(words[2]) # Adiciona os valores de arousal à lista
                            dominance.append(words[3]) # Adiciona os valores de dominance à lista

                for words in word_music_list: # Percorre todas as palavras do arquivo de texto
                    if words in wordslist_dal_anew and words not in storedMatches: # Se a palavra está no arquivo DAL_ANEW e não foi encontrada ainda
                        storedMatches.append(words) # Adiciona a palavra à lista de palavras encontradas
                        matches = storedMatches.count(words) # Conta o número de vezes que a palavra aparece no arquivo de texto
                        valenciatotal += float(valence[wordslist_dal_anew.index(words)]) * matches # Soma os valores de valência
                        arousaltotal += float(arousal[wordslist_dal_anew.index(words)]) * matches # Soma os valores de arousal
                        dominanceTotal += float(dominance[wordslist_dal_anew.index(words)]) * matches # Soma os valores de dominance

                meanValence = valenciatotal / len(storedMatches) # Calcula a média de valência
                meanArousal = arousaltotal / len(storedMatches) # Calcula a média de arousal
                meanDominance = dominanceTotal / len(storedMatches) # Calcula a média de dominance
                csvwriter.writerow([file, meanValence, meanArousal, meanDominance]) # Escreve os resultados no arquivo CSV
        QMessageBox.information(self, "Success", "DAL_ANEW.csv successfully created in the Output folder.") # Mensagem de sucesso
    except Exception as e:
        QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 25 Código DAL_ANEW

Funcionamento:

1. A função começa com um bloco "try-except", que é usado para lidar com exceções que podem ocorrer durante a execução do código.

2. A função imprime uma mensagem no console, indicando que a análise das features DAL_ANEW vai começar.

3. Os ficheiros presentes no diretório especificado pela variável "base_dir" são listados usando a função "listdir" da biblioteca "os". Isto permite que o código percorra cada ficheiro para realizar a análise.

4. A variável "csv_filename" é criada para armazenar o caminho completo e nome do ficheiro CSV que irá ser gerado após a análise dos textos. O ficheiro CSV irá conter os valores de valência, arousal e dominance médios encontrados nos textos.

5. O código, então, abre o ficheiro CSV para escrita no modo 'w' (modo escrita) utilizando a biblioteca "csv" e cria um objeto "csvwriter" para escrever os dados no ficheiro.

6. O cabeçalho das colunas é escrito no ficheiro CSV, contendo as informações "File", "AvgValence", "AvgArousal" e "AvgDominance".

7. Para cada ficheiro presente na lista "files_in_dir", o código realiza as seguintes ações:

    a) Cria uma lista chamada "storedMatches" para armazenar as palavras já encontradas no ficheiro, evitando duplicações na contagem.
    b) Inicializa as variáveis "valenciatotal", "arousaltotal" e "dominanceTotal" como 0, que serão usadas para somar os valores de valência, arousal e dominance encontrados nos ficheiros.
    c) Lê o conteúdo do ficheiro de texto e separa as palavras numa lista chamada "word_music_list".
    d) Abre o ficheiro "DAL_ANEW" e cria listas para guardar as palavras e os seus respetivos valores de valência, arousal e dominance.
    e) Percorre cada palavra na lista "word_music_list" e realiza as seguintes ações:

        i) Verifica se a palavra está presente no ficheiro "DAL_ANEW" (representado pela lista "wordslist_dal_anew") e se esta ainda não foi encontrada anteriormente. Se ambos os critérios forem bem-sucedidos, a palavra é adicionada à lista "storedMatches" e o número de ocorrências dela no ficheiro de texto é contado.

        ii) Recupera os valores de valência, arousal e dominance associados à palavra na lista "wordslist_dal_anew" usando o método "index". De seguida, multiplica esses valores pelo número de ocorrências da palavra no ficheiro de texto e adiciona os resultados às variáveis "valenciatotal", "arousaltotal" e "dominanceTotal".

8. Calcula a média de valência, arousal e dominance dividindo as somas totais pelas quantidades de palavras encontradas (armazenadas em "len(storedMatches)").

9. Escreve os resultados de cada ficheiro no ficheiro CSV usando o método "writerow" do objeto "csvwriter".

10. Após a análise de todos os ficheiros, uma dialog box de informação é exibida, indicando que o ficheiro CSV foi criado com sucesso na pasta "Output".

11. Caso ocorra algum erro durante a execução do código, a exceção será capturada pelo bloco "except", e uma dialog box de erro é exibida, mostrando a mensagem de erro específica.

Output:

File	AvgValenc	AvgArousa	AvgDominance
L001-141.t	1.9413820	1.5918979	1.8760000000000003
L005-84.t	1.8099953	1.6775883	1.7581395348837208
L006-135.t	1.7806317	1.7774642	1.744715447154471
L007-76.t	1.8995326	1.7865532	1.7173913043478257
L008-146.t	1.9706196	1.7414232	1.617857142857143
L009-151.t	1.98855	1.7550192	1.6000000000000003
L010-168.t	1.8913730	1.6546365	1.7538461538461538
L011-156.t	1.8331382	1.6913044	1.870588235294117
L012-81.t	2.0030836	1.6980020	1.636734693877551
L013-159.t	1.8889147	1.7051411	1.5823529411764714
L014-137.t	1.9864708	1.7100812	1.637500000000001
L015-160.t	1.8815741	1.7579516	1.5290322580645161
L016-77.t	1.8338346	1.8292551	1.8081632653061226

Figura 26 Código DAL_ANEW

[] → Ficheiros a serem analisados

[] → Features extraídas

[] → Valores das features

3.6.5. LIWC:

O LIWC utiliza algoritmos de NLP para contar e categorizar palavras em diferentes categorias linguísticas e psicológicas. A ferramenta possui um dicionário léxico "LIWC2007_English100131.dic" (James Pennebaker, 2015).

Código:

```
def featuresLIWC(self):
    try:
        print("Features LIWC") # Mensagem no console
        files_in_dir = listdir(base_dir) # Lista os arquivos do diretório
        csv_filename = os.path.join(project_dir, "Output", "LIWC.csv") # Caminho do arquivo CSV

        with open(csv_filename, 'w', newline='') as csvfile: # newline='' para evitar linhas em branco
            csvwriter = csv.writer(csvfile) # Cria um objeto csv.writer
            header = ['File', 'funct', 'pronoun', 'ppron', 'you', 'social', 'verb', 'auxverb', 'past', 'number', 'conj',
                      'cogmech', 'tentat', 'excl', 'affect', 'posemo', 'achieve', 'ipron', 'certain', 'adverb', 'time',
                      'relativ', 'discrep', 'bio', 'body', 'present', 'negemo', 'health', 'percept', 'feel', 'preps',
                      'space', 'shehe', 'family', 'anx', 'article', 'see', 'incl', 'inhib', 'quant', 'motion', 'hear',
                      'cause', 'leisure'] # Cabeçalho do arquivo CSV
            csvwriter.writerow(header) # Escreve o cabeçalho no arquivo CSV

            for file in files_in_dir: # Percorre todos os arquivos do diretório
                filepath = nltk.data.find(base_dir + file) # Caminho do arquivo
                textfile = open(filepath, 'r').read() # Abre o arquivo
                tokens = tokenize(textfile) # Tokeniza o texto
                liwc = Counter(category for token in tokens for category in parse(token)) # Conta as categorias LIWC
                liwc_values = {} # Dicionário para guardar os valores de cada categoria
                for category in header[1:]: # Percorre todas as categorias
                    liwc_values[category] = liwc[category] # Adiciona os valores ao dicionário
                row_data = [file] + [liwc_values.get(category, 0) for category in header[1:]] # Lista com os valores de cada categoria
                csvwriter.writerow(row_data) # Escreve os resultados no arquivo CSV
        QMessageBox.information(self, "Success", "LIWC.csv successfully created in the Output folder.")
    except Exception as e:
        QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 27 Código LIWC

Funcionamento:

1) A função começa com um bloco "try-except", que é usado para lidar com exceções que podem ocorrer durante a execução do código.

2) A função imprime uma mensagem na consola, indicando que a análise das features LIWC vai começar.

3) Os ficheiros presentes no diretório especificado pela variável "base_dir" são listados usando a função "listdir" da biblioteca "os". Isto permite que o código percorra cada ficheiro para realizar a análise.

4) A variável "csv_filename" é criada para armazenar o caminho completo e nome do ficheiro CSV que será gerado após a análise dos textos. O ficheiro CSV conterá as informações das categorias LIWC encontradas nos textos.

5) O código, então, abre o ficheiro CSV para escrita no modo 'w' (modo escrita) usando a biblioteca "csv" e cria um objeto "csvwriter" para escrever os dados no ficheiro.

6) O cabeçalho das colunas é escrito no ficheiro CSV, contendo as informações "File" e as categorias LIWC, como "funct", "pronoun", "ppron", "you", "social", "verb", "auxverb", "past", "number", "conj", entre outras.

7) Para cada ficheiro presente na lista "files_in_dir", o código realiza as seguintes ações:

    a) Recupera o caminho do ficheiro "filepath" e lê o conteúdo do ficheiro "textfile".

    b) O texto é tokenizado usando a função "tokenize", que divide o texto em palavras (tokens).

    c) As palavras são analisadas pelo LIWC usando a função "parse", que retorna as categorias LIWC para cada palavra.

    d) Um objeto "Counter" é usado para contar as ocorrências de cada categoria LIWC nas palavras analisadas.

    e) Os valores das categorias LIWC são armazenados no dicionário "liwc_values", onde a chave é o nome da categoria e o valor é a contagem correspondente.

8) O código percorre todas as categorias do LIWC (exceto o "File", que já está presente no cabeçalho) e cria uma lista chamada "row_data", que contém os valores das categorias para cada ficheiro. A lista é criada concatenando o nome do ficheiro e os valores das categorias correspondentes.

9) A lista "row_data" é escrita no ficheiro CSV usando o método "writerow" do objeto "csvwriter".

10) Após a análise de todos os ficheiros, uma caixa de diálogo de informação é exibida, indicando que o ficheiro CSV foi criado com sucesso na pasta "Output".

11) Caso ocorra algum erro durante a execução do código, a exceção será capturada pelo bloco "except", e uma caixa de diálogo de erro será exibida, mostrando a mensagem de erro específica. E.g: caso o ficheiro não seja encontrado, ou a lista estiver vazia.

Output:

File	funct	pronoun	ppron	you	social	verb	auxverb	past	number	conj	cogmech	tentat	excl	affect	posemo	achieve
L001-141.	49	11	9	5	9	14	4	1	2	4	14	0	5	1	1	1
L005-84.t	67	12	2	2	2	20	9	0	1	0	15	2	0	2	0	3
L006-135.	163	51	38	29	50	38	19	4	2	6	41	3	1	37	7	4
L007-76.t	95	35	26	16	32	37	14	15	2	4	19	2	2	10	4	3
L008-146.	107	59	47	12	43	47	18	17	1	10	33	7	4	21	16	1
L009-151.	25	10	10	5	10	8	3	1	0	2	5	0	1	8	7	0
L010-168.	58	9	5	1	7	18	11	1	0	3	14	2	0	5	4	0
L011-156.	107	16	15	0	10	16	10	1	1	13	21	2	1	7	5	0
L012-81.t	126	23	8	7	9	44	27	1	0	14	40	13	13	5	3	0
L013-159.	100	32	12	7	17	31	13	14	4	19	44	13	6	8	3	5
L014-137.	55	13	10	4	9	16	8	8	0	7	18	0	0	13	5	2
L015-160.	92	39	30	25	26	50	23	1	1	9	28	1	0	10	8	1
L016-77.t	49	19	12	3	19	14	2	10	4	0	9	0	0	8	3	4

Figura 28 Output LIWC

[] → Ficheiros a serem analisados

[] → Features extraídas

[] → Valores das features

3.6.6. Warriner:

Este recurso contém uma ampla lista de palavras da língua inglesa, cada uma acompanhada de suas classificações em três dimensões afetivas: valência, arousal e dominance.

Estes valores foram extraídos foram nos fornecidos do ficheiro “Warriner.txt”.

Código:

```
def featuresWarriner(self):
    try:
        csv_filename = os.path.join(project_dir, "Output", "Warriner.csv") # Nome do arquivo CSV
        warriner_ratings = {}  # Dicionário que armazenará os dados
        word_emotion_data = {}  # Dicionário para armazenar as emoções das palavras no arquivo a ser testado

        # Lendo o arquivo Warriner e preenchendo o dicionário
        with open(warriner_file, 'r', encoding='utf-8') as txtfile:
            for line in txtfile:
                row = line.strip().split('\t')
                if len(row) == 4:
                    word, valence, arousal, dominance = row
                    warriner_ratings[word.lower()] = {
                        'valence': float(valence),
                        'arousal': float(arousal),
                        'dominance': float(dominance)
                    }

        for file in os.listdir(base_dir):  # Iterar sobre cada arquivo no diretório
            filepath = os.path.join(base_dir, file)  # Obter o caminho completo do arquivo
            lyric_text = open(filepath, 'r').read()  # Ler o arquivo

            lyric_words = lyric_text.lower().split()  # Separar as palavras do texto

            # Inicializar contadores para as emoções
            total_valence = 0
            total_arousal = 0
            total_dominance = 0
            total_matches = 0

            for word in lyric_words:
                if word in warriner_ratings:
                    total_matches += 1
                    word_data = warriner_ratings[word]
                    total_valence += word_data['valence']
                    total_arousal += word_data['arousal']
                    total_dominance += word_data['dominance']

            if total_matches > 0:
                average_valence = total_valence / total_matches
                average_arousal = total_arousal / total_matches
                average_dominance = total_dominance / total_matches
            else:
                average_valence = 0
                average_arousal = 0
                average_dominance = 0

            word_emotion_data[file] = {
                'average_valence': average_valence,
                'average_arousal': average_arousal,
                'average_dominance': average_dominance
            }

        with open(csv_filename, 'w', newline='') as csvfile:  # Abrir o arquivo CSV para escrita
            csvwriter = csv.writer(csvfile)  # Criar um objeto para escrever no arquivo CSV
            csvwriter.writerow(['File', 'Average Valence', 'Average Arousal',
                                'Average Dominance'])  # Escrever o cabeçalho do arquivo CSV

            for file, emotion_data in word_emotion_data.items():
                csvwriter.writerow([file, emotion_data['average_valence'], emotion_data['average_arousal'],
                                    emotion_data['average_dominance']])  # Escrever os dados no arquivo CSV

        print(f"Emotion scores saved to '{csv_filename}'")  # Imprimir mensagem de sucesso
        QMessageBox.information(self, "Success",
                                "Emotion scores successfully saved to Warriner.csv in the Output folder.")

    except Exception as e:
        print(f"An error occurred: {str(e)}")
        QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 29 Código Warriner

Funcionamento:

1) A função começa com um bloco "try-except", que é usado para lidar com exceções que podem ocorrer durante a execução do código.

2) A variável "csv_filename" é criada para armazenar o caminho completo e nome do ficheiro CSV que será gerado após a análise dos textos. O ficheiro CSV irá conter as informações de valência, arousal e dominance médios encontrados nos textos.

3) A variável "warriner_ratings" é criada como um dicionário vazio para armazenar os dados.

4) O código abre o ficheiro de texto da base de dados Warriner usando a função "open" com o parâmetro 'r' para leitura, e o encoding 'utf-8' é especificado para garantir que o ficheiro é lido corretamente.

5) O código percorre cada linha do ficheiro usando um loop "for". Cada linha é separada em colunas usando o método "split" com '\t' como separador. Se a linha tiver 4 colunas, os valores de palavra, valência, arousal e dominance são extraídos e armazenados no dicionário "warriner_ratings". A palavra é convertida para letras minúsculas antes de ser usada como chave no dicionário.

6) O código abre o ficheiro CSV para escrita no modo 'w' (modo escrita) usando a biblioteca "csv" e cria um objeto "csvwriter" para escrever os dados no ficheiro.

7) O cabeçalho das colunas é escrito, contendo as informações "File" e "Average Valence", "Average Arousal" e "Average Dominance".

8) Para cada ficheiro presente na lista de ficheiros do diretório (obtida usando "os.listdir"), o código realiza as seguintes ações:

    a) Obtém o caminho completo do ficheiro usando "os.path.join".

    b) Lê o conteúdo do ficheiro e converte todas as palavras em letras minúsculas antes de separá-las numa lista chamada "lyric_words".

    c) O código conta o número de palavras no texto e armazena esse valor na variável "word_count".

    d) Se o número de palavras for igual a zero (ou seja, não há palavras no texto), o código continua para o próximo ficheiro.

e) Para cada palavra em "lyric_words", o código verifica se a palavra está presente no dicionário "warriner_ratings" usando "get". Se a palavra estiver presente, o código obtém os valores de valência, arousal e dominance associados a ela. Caso contrário, atribui o valor 0.

f) É efetuado o cálculo da soma total dos valores de valência, arousal e dominance para todas as palavras no texto.

g) De seguida, calcula a média de valência, arousal e dominance, dividindo as somas totais pelo número de palavras (armazenado em "word_count").

h) Cria uma lista chamada "row_data" com o nome do ficheiro e os valores médios de valência, arousal e dominance.

i) A lista "row_data" é escrita no ficheiro CSV usando o método "writerow" do objeto "csvwriter".

9) Após a análise de todos os ficheiros, a função imprime uma mensagem na consola indicando que os valores de emoção foram guardados.

10) Uma dialog box é mostrada, indicando que o ficheiro CSV foi criado com sucesso na pasta "Output".

11) Caso ocorra algum erro durante a execução do código, a exceção será capturada pelo bloco "except", e uma dialog box é nos apresentada, mostrando a mensagem de erro específica.

Output:

File	Average V	Average A	Average Dominance
L001-141.t	1.9612698	1.1815873	1.78920634920635
L005-84.tx	2.0222222	1.3801754	1.7728070175438606
L006-135.t	1.4357323	1.3294191	1.49939393939394
L007-76.tx	1.5727272	1.0855023	1.4808133971291864
L008-146.t	1.5801357	1.0342986	1.4980542986425345
L009-151.t	1.2875000	0.8185526	1.1848684210526317
L010-168.t	2.1472262	1.3001459	2.0087591240875917
L011-156.t	1.4457021	1.0246808	1.4251489361702128
L012-81.tx	1.9574904	1.1567300	1.6527376425855522
L013-159.t	1.4240625	1.0119196	1.335625
L014-137.t	1.8328244	1.3124427	1.7090076335877857
L015-160.t	1.4670652	0.9341847	1.433369565217391
L016-77.tx	1.8520930	1.4939534	1.7939534883720931

Figura 30 Output Warriner

[] → Ficheiros a serem analisados

[] → Features extraídas

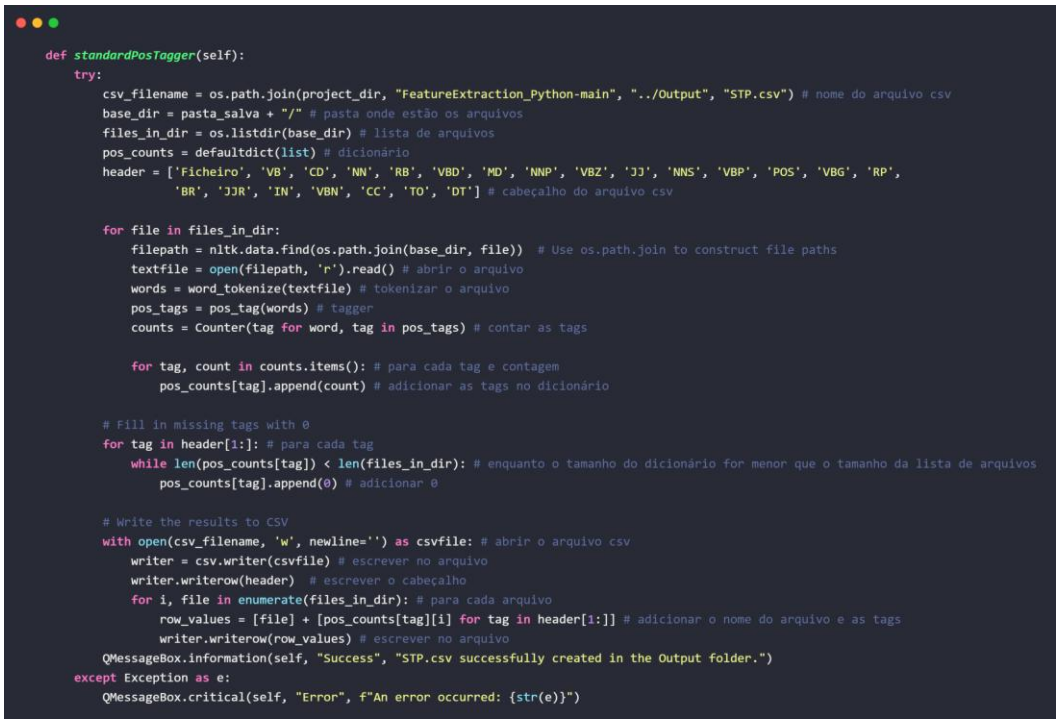
[] → Valores das features

3.7.Standard Pos Tagger

3.7.1. standardPosTagger:

Esta funcionalidade faz parte das features estilísticas, mas devido à sua complexidade decidimos separa-la num subcapítulo. Esta tem a função de etiquetar com “tags” as classes de gramaticais das palavras presentes nas letras musicais.

Código:



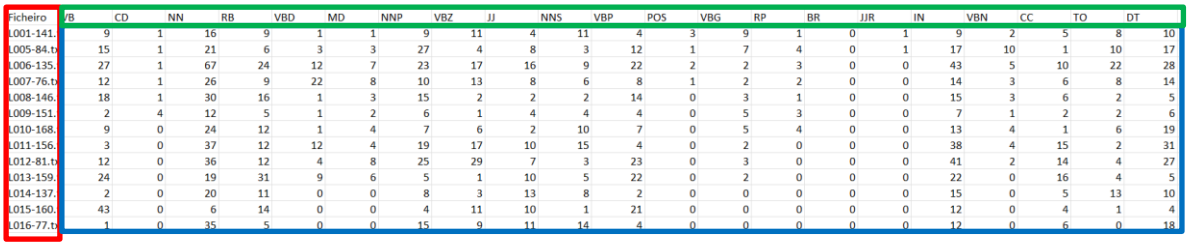
```
def standardPosTagger(self):
    try:
        csv_filename = os.path.join(project_dir, "FeatureExtraction_Python-main", "../Output", "STP.csv") # nome do arquivo csv
        base_dir = pasta_salva + "/" # pasta onde estão os arquivos
        files_in_dir = os.listdir(base_dir) # lista de arquivos
        pos_counts = defaultdict(list) # dicionário
        header = ['Ficheiro', 'VB', 'CD', 'NN', 'RB', 'VBD', 'MD', 'NNP', 'VBZ', 'JJ', 'NNS', 'VBP', 'POS', 'VBG', 'RP',
                  'BR', 'JJR', 'IN', 'VBN', 'CC', 'TO', 'DT'] # cabeçalho do arquivo csv

        for file in files_in_dir:
            filepath = nltk.data.find(os.path.join(base_dir, file))  # Use os.path.join to construct file paths
            textfile = open(filepath, 'r').read() # abrir o arquivo
            words = word_tokenize(textfile) # tokenizar o arquivo
            pos_tags = pos_tag(words) # tagger
            counts = Counter(tag for word, tag in pos_tags) # contar as tags

            for tag, count in counts.items(): # para cada tag e contagem
                pos_counts[tag].append(count) # adicionar as tags no dicionário

        # Fill in missing tags with 0
        for tag in header[1:]: # para cada tag
            while len(pos_counts[tag]) < len(files_in_dir): # enquanto o tamanho do dicionário for menor que o tamanho da lista de arquivos
                pos_counts[tag].append(0) # adicionar 0

        # Write the results to CSV
        with open(csv_filename, 'w', newline='') as csvfile: # abrir o arquivo csv
            writer = csv.writer(csvfile) # escrever no arquivo
            writer.writerow(header)  # escrever o cabeçalho
            for i, file in enumerate(files_in_dir): # para cada arquivo
                row_values = [file] + [pos_counts[tag][i] for tag in header[1:]] # adicionar o nome do arquivo e as tags
                writer.writerow(row_values) # escrever no arquivo
        QMessageBox.information(self, "Success", "STP.csv successfully created in the Output folder.")
    except Exception as e:
        QMessageBox.critical(self, "Error", f"An error occurred: {str(e)}")
```

Figura 31 Código STP

Funcionamento:

1. Define o nome do ficheiro CSV onde os resultados serão guardados (csv_filename).
2. Obtém a lista de ficheiros presentes na pasta especificada em base_dir.
3. Cria um dicionário vazio pos_counts com chaves para cada tag gramatical do cabeçalho e valores como listas para armazenar as contagens de cada tag para cada ficheiro.
4. Define o cabeçalho do ficheiro CSV com as tags gramaticais a serem extraídas.
5. Para cada ficheiro na lista, abre o ficheiro, tokeniza o texto e atribui tags gramaticais a cada palavra usando o tagger.
6. Conta as ocorrências de cada tag no ficheiro e armazena as contagens no dicionário pos_counts.
7. Preenche as tags ausentes com 0 no dicionário.
8. Escreve os resultados no ficheiro CSV com o nome do ficheiro e as contagens de cada tag.

Output:



Ficheiro	VB	CD	NN	RB	VBD	MD	NNP	VBZ	JJ	NNS	VBP	POS	VBG	RP	BR	JJR	IN	VBN	CC	TO	DT
L001-141.	9	1	16	9	1	1	9	11	4	11	4	3	9	1	0	1	9	2	5	8	10
L005-84.tx	15	1	21	6	3	3	27	4	8	3	12	1	7	4	0	1	17	10	1	10	17
L006-135.	27	1	67	24	12	7	23	17	16	9	22	2	2	3	0	0	43	5	10	22	28
L007-76.tx	12	1	26	9	22	8	10	13	8	6	8	1	2	2	0	0	14	3	6	8	14
L008-146.	18	1	30	16	1	3	15	2	2	2	14	0	3	1	0	0	15	3	6	2	5
L009-151.	2	4	12	5	1	2	6	1	4	4	4	0	5	3	0	0	7	1	2	2	6
L010-168.	9	0	24	12	1	4	7	6	2	10	7	0	5	4	0	0	13	4	1	6	19
L011-156.	3	0	37	12	12	4	19	17	10	15	4	0	2	0	0	0	38	4	15	2	31
L012-81.tx	12	0	36	12	4	8	25	29	7	3	23	0	3	0	0	0	41	2	14	4	27
L013-159.	24	0	19	31	9	6	5	1	10	5	22	0	2	0	0	0	22	0	16	4	5
L014-137.	2	0	20	11	0	0	8	3	13	8	2	0	0	0	0	0	15	0	5	13	10
L015-160.	43	0	6	14	0	0	4	11	10	1	21	0	0	0	0	0	12	0	4	1	4
L016-77.tx	1	0	35	5	0	0	15	9	11	14	4	0	0	0	0	0	12	0	6	0	18

Figura 32 Output STP

[] → Ficheiros a serem analisados

[] → Features extraídas

[] → Valores das features

4. Conclusão

Ao longo deste relatório, delineámos o desenvolvimento de uma aplicação em Python voltada para a extração de características de letras de músicas, contribuindo para o ambicioso projeto MERGE (Music Emotion Recognition New Generation) do Centro de Informática e Sistemas da Universidade de Coimbra (CISUC). Financiado pela FCT, o projeto visa elevar os padrões no reconhecimento de emoções na música, incorporando técnicas de *machine learning* e *deep learning*, considerando tanto o áudio quanto a letra.

O relatório apresenta minuciosamente os detalhes do processo de desenvolvimento e as técnicas empregadas, além de destacar o estado da arte, evidenciando o impacto e potencial dessa aplicação.

A criação desta aplicação representa um passo significativo no campo do Music Emotion Recognition (MER), automatizando a extração de features das letras musicais e viabilizando análises mais eficientes e detalhadas. Através da aplicação de técnicas de processamento de linguagem natural (NLP) e representação textual, esta ferramenta capacita o sistema MER a decifrar as complexas nuances emocionais presentes nas letras musicais.

Foram realizados testes com a presente aplicação em Python e esses resultados foram comparados com a aplicação em JAVA para que a partir de um conjunto de letras musicais garantirmos que os resultados se mantinham os mesmos.

Este projeto foi enriquecedor e desafiante, proporcionando uma aprendizagem abrangente em tecnologias como linguagem Python, processamento de linguagem natural, *machine learning* e *deep learning*. Reconhecemos a profundidade e complexidade da música, e como a tecnologia atua como um meio para explorar as suas dimensões emocionais.

Além do seu impacto prático, a aplicação oferece valiosas contribuições para a comunidade científica. Ao fornecer um método eficaz e coerente para extrair características das letras de músicas, esta ferramenta serve como base para pesquisas mais profundas em psicologia, ciência cognitiva e análise de dados. A disponibilidade dessa aplicação em Python aumenta a acessibilidade e a versatilidade, fomentando uma gama mais ampla de investigações académicas.

Com a abertura para colaborações interdisciplinares e a contínua evolução da música e da tecnologia, vislumbramos um futuro onde a análise emocional das letras de músicas continuará a explorar caminhos inovadores, aprofundando a nossa compreensão das complexidades da expressão humana por meio da harmonia das palavras. Por conseguinte, esta aplicação contribui para o crescimento do conhecimento científico, enriquecendo tanto o panorama tecnológico quanto a compreensão das emoções humanas na música.

Bibliografia ou Referências Bibliográficas

Malheiro, (2017). Emotion-based Analysis and Classification of Music Lyrics, Ricardo Malheiro. (2017).

Malheiro et al, (2016). Ricardo Malheiro, Renato Panda, Paulo Gomes and Rui Pedro Paiva. (2016). “Emotionally-Relevant Features for Classification and Regression of Music Lyrics”.

Yang, (2012). Yang, H. (Hui-Ling). (2012). “Exploring the relationships between music preferences and personality traits in the Taiwanese adolescent context”.

Koelsch, Stefan. (2010). “Towards a neural basis of music-evoked emotions”.

Bradley e Lang, (1999). Bradley, M. and Lang, P. (1999). “Affective Norms for English Words (ANEW): Stimuli, Instruction Manual and Affective Ratings”. Technical report C-1, The Center for Research in Psychophysiology, University of Florida.

Sebastiani, (2002). Sebastiani, Fabrizio. (2002). “Machine learning in automated text categorization”.

Communities," 2018 IEEE International Conference on Big Data (Big Data), Seattle, WA, 2018, pp. 4424-4430, doi: 10.1109/BigData.2018.8622040.

Li et al, (2007). Li, H., Pang, N. and Guo, S. (2007). “Research on Textual Emotion Recognition Incorporating Personality Factor”. In: International Conference on Robotics and Biomimetics, Sanya, China.

Hu, X. and Downie, J. (2007). “Exploring mood metadata: Relationships with genre, artist and usage metadata”. In: 8 th International Conference on Music Information Retrieval (ISMIR’07).

Hu, Downie, (2010). Hu, X. and Downie, J. (2010). “When Lyrics Outperform Audio For Music Mood Classification: A Feature Analysis”. 11th International Society for Music Information Retrieval Conference (ISMIR 2010).

Hu, Downie, (2010) Hu, X. and Downie, J. (2010). “Improving mood classification in music digital libraries by combining lyrics and audio”. In: 10th annual joint conference on Digital libraries, pp. 159-168.

Hu, Downie, (2010) Hu, X., Downie, J., Laurier, C., Bay, M. and Ehmann, A. (2008). “The 2007 MIREX Audio Music 144 Classification task: lessons learned”. In: 9 th International Conference on Music Information Retrieval (ISMIR’08), pp. 462-467.

Lu et al, (2006). Lu, C., Hong, J. and Lara, S. (2006). “Emotion Detection in Textual Information by Semantic Role Labeling and Web Mining Techniques”. 3rd Taiwanese-French Conference on Information Technology (TFIT 2006).

Lu et al, (2006). Lu, L., Liu, D. and Zhang, H. (2006). “Automatic mood detection and tracking of music audio signals“. IEEE Transactions on Audio, Speech, and Language Processing, Vol. 14, No. 1, pp. 5- 18.

Philip et al, (1966). Philip J. Stone, Dexter C. Dunphy e Marshall S. Smith. (1966). “The General Inquirer: A Computer Approach to Content Analysis”.

Laurier et al, (2008). Laurier, C., Grivolla, J. and Herrera, P. (2008). “Multimodal music mood classification using audio and lyrics”. In: International Conference on Machine Learning and Applications.

Russell, J. (1980). “A circumspect model of affect”. Journal of Psychology and Social Psychology, Vol. 39, No. 6, pp. 1161-1178.

Sebastiani, F. (2002). “Machine learning in automated text categorization”. ACM Computing Surveys, Vol. 34, No. 1, pp. 1–47.

Whissel, (1989). Whissell, C. (1989). “Dictionary of Affect in Language”. In: Plutchik and Kellerman (Eds.) Emotion: Theory, Research and Experience, Vol. 4, pp. 113–131, Academic Press, NY.

Biblioteca sys: https://docs.python.org/pt-br/3/library/sys.html (acedido 27 de agosto de 2023).

Pyqt5: https://acervolima.com/python-introducao-ao-pyqt5/ (acedido 27 de agosto de 2023).

Nltk: https://www.nltk.org/ (acedido 27 de agosto de 2023)..

PyQt5: https://www.geeksforgeeks.org/python-introduction-to-pyqt5/ (acedido 27 de agosto de 2023).

PyQt5: https://build-system.fman.io/pyqt5-tutorial (acedido 27 de agosto de 2023).

PyQt5: https://www.geeksforgeeks.org/pyqt5-qradiobutton/ (acedido 27 de agosto de 2023).

PyQt5: https://pythonbasics.org/pyqt-radiobutton/ (acedido 27 de agosto de 2023).

PyQt5: https://pythonprogramminglanguage.com/pyqt5-button/ (acedido 27 de agosto de 2023).

https://github.com/dwzhou/SentimentAnalysis (acedido 28 de agosto de 2023).

Genius: https://docs.genius.com/ (acedido 28 de agosto de 2023).

Genius: https://genius.com/developers (acedido 28 de agosto de 2023).

Working with CSV files: https://www.analyticsvidhya.com/blog/2021/08/python-tutorial-working-with-csv-file-for-data-science/ (acedido 28 de agosto de 2023).

CSV: https://docs.python.org/3/library/csv.html (acedido 28 de agosto de 2023).

String upper: https://www.w3schools.com/python/ref_string_upper.asp (acedido 21 de maio de 2023).

String upper: https://www.programiz.com/python-programming/methods/string/upper (acedido 28 de junho de 2023).

How to set an icon in PyQt5: https://www.geeksforgeeks.org/how-to-set-icon-to-a-window-in-pyqt5/ (acedido 28 de junho de 2023).

Split Python: https://blog.betrybe.com/python/python-split/ (acedido 28 de junho de 2023).

Dividir strings em python: https://www.freecodecamp.org/portuguese/news/como-dividir-uma-string-em-substrings-em-python/ (acedido 28 de junho de 2023).

Remove Stemming: https://stackoverflow.com/questions/71566021/how-to-stop-stemming-from-removing-the-letters-that-change-the-meaning-of-the-wo (acedido 28 de julho de 2023).

Stemming words: https://stackoverflow.com/questions/30458511/stemming-words-in-python?rq=3 (acedido 29 de julho de 2023).

Stemming words with nltk: https://www.geeksforgeeks.org/python-stemming-words-with-nltk/ (acedido 29 de julho de 2023).

Remove stopwords: https://www.analyticsvidhya.com/blog/2019/08/how-to-remove-stopwords-text-normalization-nltk-spacy-gensim-python/ (acedido 29 de julho de 2023)

Remove stopwords: https://stackabuse.com/removing-stop-words-from-strings-in-python/ (acedido 29 de julho de 2023).

Remove stopwords: https://www.geeksforgeeks.org/removing-stop-words-nltk-python/ (acedido 29 de julho de 2023)

Remove Stopwords using nltk: https://stackoverflow.com/questions/5486337/how-to-remove-stop-words-using-nltk-or-python (acedido 29 de julho de 2023)

NLP: https://towardsdatascience.com/natural-language-processing-feature-engineering-using-tf-idf-e8b9d00e7e76 (acedido 15 de maio de 2023)

TF-IDF: https://www.geeksforgeeks.org/understanding-tf-idf-term-frequency-inverse-document-frequency/ (acedido 12 de junho de 2023)

TF-IDF: https://towardsdatascience.com/tf-term-frequency-idf-inverse-document-frequency-from-scratch-in-python-6c2b61b78558 (acedido 12 de junho de 2023)

TF-IDF: https://stackoverflow.com/questions/49277926/python-tf-idf-algorithm (acedido 12 de junho de 2023)

Count frequency elements in list: https://stackoverflow.com/questions/2161752/how-to-count-the-frequency-of-the-elements-in-an-unordered-list (acedido 15 de agosto de 2023)

Word frequency analysis: https://codereview.stackexchange.com/questions/229916/word-frequency-analysis-python (acedido 15 de agosto de 2023)

Frequency or count of a list: https://stackoverflow.com/questions/66809690/frequency-or-count-of-a-list-of-sets-in-python (acedido 15 de agosto de 2023)

PyQt5: https://doc.qt.io/qtforpython-5/PySide2/QtWidgets/QDesktopWidget.html (acedido 26 de agosto de 2023)

Resize screen: https://stackoverflow.com/questions/35887237/current-screen-size-in-python3-with-pyqt5 (acedido 26 de agosto de 2023)

Pyqt5: https://pythonpyqt.com/qbuttongroup/ (acedido 26 de agosto de 2023)

NLTK: https://www.nltk.org/api/nltk.tag.html (acedido 6 de agosto de 2023)

POS_Tag: https://www.nltk.org/api/nltk.tag.pos_tag.html (acedido 22 de agosto de 2023)

POS_Tag with NLTK: https://www.guru99.com/pos-tagging-chunking-nltk.html (acedido 22 de agosto de 2023)

NLTK: https://www.nltk.org/_modules/nltk/tag.html (acedido 22 de agosto de 2023)

LIWC: https://pypi.org/project/liwc-text-analysis/ (acedido 14 de agosto de 2023)

LIWC: https://pypi.org/project/liwc/ (acedido 14 de agosto de 2023)

LIWC: https://github.com/chbrown/liwc-python (acedido 14 de agosto de 2023)

Os list_dir:https://www.geeksforgeeks.org/python-os-listdir-method/ (acedido 20 de agosto de 2023)

OS:https://www.tutorialspoint.com/python/os_listdir.htm (acedido 20 de agosto de 2023)

NRCLex:https://pypi.org/project/NRCLex/ (acedido 22 de agosto de 2023)

NRCLex:https://www.tutorialspoint.com/emotion-classification-using-nrc-lexicon-in-python (acedido 22 de agosto de 2023)

NRCLex:https://www.geeksforgeeks.org/emotion-classification-using-nrc-lexicon-in-python/ (acedido 22 de agosto de 2023)

Emotion classification: https://medium.com/geekculture/simple-emotion-classification-in-python-40fb24692541 (acedido 19 de agosto de 2023)

Gazetteer:https://gatenlp.github.io/python-gatenlp/gazetteers.html (acedido 23 de agosto de 2023)

Gazetteer:https://github.com/topics/gazetteer?l=python (acedido 23 de agosto de 2023)

Glossário

Arousal: Refere-se ao nível de excitação ou ativação emocional que um estímulo provoca. Pode variar de estados calmos e relaxados a estados altamente excitados e alertas. O arousal está relacionado à intensidade das emoções que um estímulo pode criar.

Valence: Refere-se à avaliação emocional positiva, negativa ou neutra de um estímulo. Por outras palavras, está relacionada a se algo é percebido como agradável, desagradável ou neutro em termos emocionais.

Dominance: Refere-se à medida em que uma palavra, expressão ou estímulo influencia ou controla a resposta emocional ou a atenção de uma pessoa. No contexto da análise emocional, a dominância avalia o grau em que algo é percebido como tendo autoridade, controle ou influência sobre a situação ou emoção.

Stemming: Stemming é um processo linguístico utilizado no processamento de linguagem natural (NLP) para reduzir uma palavra à sua forma raiz ou base, conhecida como "stem". O objetivo do stemming é normalizar as palavras, agrupando diferentes variações morfológicas da mesma raiz em uma única forma.

Slang Words: (palavra de calão) é uma expressão ou termo informal que faz parte da linguagem coloquial ou gíria de um determinado grupo, comunidade ou subcultura.

N-grams: N-gram é uma sequência contígua de n itens em uma sequência de texto. Os itens podem ser palavras, caracteres, fonemas ou outros elementos, dependendo do contexto de análise. Os n-grams são utilizados no processamento de linguagem natural (PLN) para capturar o contexto e as relações entre os elementos de um texto.

Stopwords: São consideradas palavras vazias, palavras que não transmitem informações especificas sobre o conteúdo ou o contexto do texto, por exemplo: “a”, “e”, “ou”, “o”.

PorterStemmer: O PorterStemmer é um algoritmo de stemming desenvolvido por Martin Porter em 1979. Ele é amplamente utilizado no processamento de linguagem natural (NLP) para reduzir palavras à sua forma raiz ou base, conhecida como "stem". O objetivo do PorterStemmer é normalizar palavras, agrupando diferentes variações morfológicas da mesma raiz em uma única forma.

Synesketch: foi o primeiro software gratuito e de open source da Web para reconhecimento de emoções textuais e visualização artística. Um trabalho pioneiro na humanização da inteligência artificial e na estética generativa, o Synesketch foi reconhecido por espaços artísticos internacionais, revistas e conferências científicas, assim como por várias organizações e indivíduos em toda a Web."

Machine learning: é um subcampo da Engenharia e da ciência da computação que evoluiu do estudo de reconhecimento de padrões e da teoria da aprendizagem computacional em inteligência artificial.

Deep learning; é uma técnica de machine learning que ensina os computadores a fazer o que é natural para os humanos: aprender pelo exemplo.

Transfer learining: é uma técnica de machine learning na qual o conhecimento aprendido numa tarefa é reutilizado para aumentar o desempenho para uma tarefa relacionada.

Framework: é um conjunto de ferramentas, bibliotecas, convenções e padrões que são organizados de forma coesa para auxiliar os investigadores de software na construção de aplicações, sistemas ou projetos específicos. Fornece uma estrutura geral que pode ser personalizada para atender às necessidades de um projeto em particular, acelerando o processo de desenvolvimento e facilitando a manutenção do código.

Tokens: é uma unidade de segmentação de texto, onde um texto é dividido em partes, que podem ser palavras individuais, partes de palavras ou até mesmo caracteres individuais, dependendo do nível desejado.

String: uma sequência de caracteres, ou seja, um conjunto ordenado de símbolos que podem incluir letras, números, espaços em branco e outros caracteres especiais. As strings são usadas para representar texto em muitos contextos de programação.

Output: refere-se à saída ou resultado produzido por um programa de computador após processar dados de entrada ou realizar cálculos.