William Kraemer Aliaga

DESENVOLVIMENTO DE UM SISTEMA DE RECOMENDAÇÃO MUSICAL SENSÍVEL AO CONTEXTO

Monografia submetida ao Programa de Graduação em Ciências da Computação para a obtenção do Grau de Bacharel em Ciências da Computação.
Orientador
Universidade Federal de Santa Catarina: Prof. Dr. Elder Rizzon Santos

Florianópolis

2018

Ficha de identificação da obra elaborada pelo autor,
através do Programa de Geração Automática da Biblioteca Universitária da UFSC.

Aliaga, William
Desenvolvimento de um Sistema de Recomendação Musical Sensível ao Contexto / William Aliaga ; orientador, Elder Rizzon Santos, 2019.
120 p.

Trabalho de Conclusão de Curso (graduação) - Universidade Federal de Santa Catarina, Centro Tecnológico, Graduação em Ciências da Computação, Florianópolis, 2019.

Inclui referências.

1. Ciências da Computação. 2. Sistemas de Recomendação. 3. Sistemas de Recomendação Sensiveis ao Contexto. 4. Sistemas de Recomendaçãos Musciais. I. Rizzon Santos, Elder. II. Universidade Federal de Santa Catarina. Graduação em Ciências da Computação. III. Título.

William Kraemer Aliaga

DESENVOLVIMENTO DE UM SISTEMA DE RECOMENDAÇÃO MUSICAL SENSÍVEL AO CONTEXTO

Esta Monografia foi julgada aprovada para a obtenção do Título de "Bacharel em Ciências da Computação", e aprovada em sua forma final pelo Programa de Graduação em Ciências da Computação.

Florianópolis, 12 de janeiro 2018.

Prof. Dr. Cislaghi
Coordenador
Universidade Federal de Santa Catarina

Banca Examinadora:

Prof. Dr. Elder Rizzon Santos
Orientador
Universidade Federal de Santa Catarina

Prof. Dr. Ricardo Azambuja Silveira
Universidade Federal de Santa Catarina

Prof. Me. Thiago Angelo Gelaim
Universidade Federal de Santa Catarina

Este trabalho é dedicado minha querida família que sempre acredita em mim.

AGRADECIMENTOS

Agradeço antes de tudo ao meu orientador, o Prof. Dr. Elder Rizzon que mesmo frustrado com minhas falhas em inúmeras oportunidades, sempre me deu uma oportunidade para me reerguer e continuar este trabalho até sua conclusão. Agradeço em seguida aos meus pais e irmãos, por me dar sempre o suporte necessário para estudar em um país diferente da minha origem e sempre ter me encorajado a continuar a pesar de todas as dificuldades encontradas no caminho. Agradeço também a minha namorada Monique, que sempre esteve disponível para me prover de carinho e cuidados quando mais precisava. Agradeço a todos os colegas de curso que de uma ou outra maneira contribuíram na minha formação seja por meio de entidades estudantis, seja por me acompanhar nas horas de estudos ou seja por compartilhar seu tempo comigo.

RESUMO

Os sistemas de recomendação tem ganhado imensa relevância ao impulsionar diversos mercados com sua capacidade de alavancar vendas, por tal motivo têm sido amplamente estudados pela comunidade acadêmica desde os anos noventa. Sistemas de recomendação não são sistemas triviais, requerem conhecimentos multidisciplinares e conhecimentos específicos da área. O presente trabalho tem por objetivo propor e desenvolver um modelo de recomendador musical baseando-se na teoria de recomendação, buscando se valer de técnicas clássicas de recomendação e aplicando uma camada extra de filtragem contextual, adicionando informações contextuais ao processo com o intuito de testar os benefícios em potencial para a recomendação.

Palavras-chave: Sistema de Recomendação, Filtragem por Conteúdo, Filtragem Colaborativa, Sensível ao Contexto

ABSTRACT

Recommendation Systems have gained huge relevance for boosting several markets with their capacity to leverage sales, for that reason they have been widely studied by the academic communitysinde the ninety's. Recommendation systems are not trivial systems, they require multidisciplinary knowledge and specific field knowledge. The present work aims to propose and develop a musical recommendation model based on recommendation theory, seeking to use classic recommendation techniques and applying an extra layer of contextual filtering, adding contextual information to the process in order to test the benefits for the recommendation.

Keywords: Recomendation Systems. Collaborative Filterning. Content Based Filtering, Context-Awareness.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

SR	Sistema de Recomendação
FC	Filtragem Colaborativa
BC	Baseado em Conteudo
AM	Aprendizado de Máquina
KNN	K Vizinhos Mais Próximos
MAP	Mean Average Precision
VP	Verdadeiro Positivo
FN	Falso Negativo
VN	Verdadeiro Negativo
FP	Falso Positivo

SUMÁRIO

1 INTRODUÇÃO

A industria musical é um dos maiores ramos do entretenimento e um dos mais relevantes da atualidade. A internet, por muito tempo temida pelos produtores de conteúdo por causa da pirataria, agora se mostra como uma forte aliada ao revolucionar o mercado totalmente apresentando novos meios de distribuição como música por download e serviços de streaming. Um relatório da Federação Internacional da Indústria Fonográfica (IFPI, na sigla em inglês) apresenta dados sobre o mercado de música digital, que em 2018 representou US$ 19,8 bilhões a nível global e obteve um crescimento de 9,7% em relação a 2017, sendo o quarto ano seguido de crescimento. Sem dúvidas os serviços de streaming de músicas tem um papel importante impulsionando o crescimento desse mercado, pois representam um total de 46.8% do total com aproximadamente 255 milhões de assinantes pagantes(IFPI, 2019). Tamanha popularidade dos serviços de streaming se deve em boa parte ao tamanho e variedade do acervo de músicas que apresentam. Um exemplo disso é o acervo do Spotify[1], maior empresa do mercado com 100 milhões de assinantes ao redor do mundo, que em 2019 atingiu a marca de 50 milhões de faixas, e que cresce em torno de 40 mil músicas por dia(TIMES, 2019). Com acervos de um tamanho tão grande, descobrir quais músicas são as mais apropriadas para serem apresentadas a um usuário se torna um desafio. Este problema não é enfrentado somente pela industria da música, mas por diversas outras áreas como por exemplo sugestão de produtos em e-commerce, recomendação de livros, recomendação filmes/séries e recomendação de amizades em redes social.

É neste cenário que sistemas de recomendação se tornam ferramentas relevantes, pois sistemas de recomendação foram desenvolvidos para ajudar nesse problema, criando uma seleção de itens que seriam de interesse para o usuário, sem demandar muita iteração com ele(a)(KUNAVER; POŽRL, 2017). O interesse comercial e acadêmico cresceu consideravelmente desde os anos noventa e trata-se de uma área amplamente estudada, um exemplo disso se dá através da *Netflix Prize*, uma competição de 2006 onde a video-locadora Netflix ofereceu um prêmio de US$ 1 milhão para a equipe que atingisse uma melhoria de 10% no algoritmo de recomendação(KOREN; BELL; VOLINSKY, 2009). A pesar de tudo isso, os sistemas de recomendação tradicionais, na sua maioria, consideram apenas duas entidades como base para suas predições: itens

[1] http://www.spotify.com

e usuários, e não se considera o contexto (e.g. localização, temperatura e clima, horário) para aprimorar a recomendação. Este trabalho visa estudar o estado da arte de sistemas de recomendação, explorando a área de Sistemas de recomendação sensíveis ao contexto, apresentar um modelo para recomendação com filtragem contextual e um protótipo que realize recomendações ao usuário.

Este trabalho está organizado da seguinte maneira: No Capítulo 2, abordamos alguns conceitos básicos para Música que serão usados na extração de características de músicas na construção da base de dados usada. No Capítulo 3 e 4, detalhamos o panorama dos Sistemas de Recomendação, abodando o caso especial de Recomendação de Músicas, e de Sistemas de Recomendação Sensíveis ao Contexto. No Capítulo 5, descrevemos o desenvolvimento, baseado na utilização do Áudio para determinar os fatores latentes para recomendação das Músicas. No Capítulo 6, reportamos os resultados experimentais obtidos com a metodologia proposta. No Capítulo 7, sumarizamos as conclusões deste trabalho e apontamos direções futuras de extensão do mesmo.

1.1 OBJETIVOS

1.1.1 Objetivos Gerais

O presente trabalho tem por objetivo modelar, implementar e avaliar um sistema de recomendação musical sensível ao contexto. A intenção ao se usar um contexto relatado pelo usuário é se obter melhores recomendações.

1.1.2 Objetivos Específicos

Para a obtenção do objetivo geral, os seguintes objetivos específicos são requeridos:

1. Analisar o panorama atual dos Sistemas de recomendação, incluindo aqueles sensíveis ao contexto.
2. Compreender e consolidar conceitos de Inteligência Artificial na área Aprendizado de Máquina. Conhecimentos que posteriormente serão aplicados para realizar a seleção dos melhores conjuntos de musicas para recomendação.
3. Compreender conceitos de Sistemas de recomendação e de sensi-

bilidade ao contexto para modelar o sistema, de modo que seja capaz realizar o analise do contexto do usuário.

4. Definir e Modelar um recomendador que permita realizar a associação das músicas com o contexto do usuário.

5. Desenvolver um protótipo permita a recomendação de músicas para um dado contexto.

6. Especificar casos de uso necessários para verificar a validade do sistema de recomendação como funcional.

7. Realizar uma avaliação dos resultados obtidos a partir dos casos de uso.

2 COMPUTAÇÃO MUSICAL

Computação musical é um ramo da área da Ciência da Computação, dedicado ao estudo multidisciplinar dos problemas musicais. A computação musical faz estudos de métodos, técnicas e algoritmos para geração e processamento de som e música, representação e armazenamento de informação sonora e musical de maneira digital, mantendo sempre a premissa de que a informação tratada é arte diferenciando-se assim de outras áreas similares, como a computação gráfica. (MILETTO et al., 2004)

2.1 REVISÃO DE CONCEITOS MUSICAIS

Esta seção apresenta uma breve introdução a conceitos e termos básicos da música.

2.1.1 Elementos Básicos da Música

O conceito de música pode ser definido como um arranjo ordenado de sons e silêncios, segundo (DUAN et al., 2014), com significado mais dramático do que literal.

O som é constituído por vibrações no ar, em outras palavras, variações na pressão no ar, recebidas pelo tímpano nos ouvidos e captadas por terminações nervosas onde são percebidas pelos humanos. No caso da pressão variar de maneira repetida, o som tem forma de onda periódica. Caso contrario, o som é apenas considerado como ruído(MILETTO et al., 2004).

A Figura 1 mostra como o som pode ser representado graficamente em formas de onda, as quais mostram mudanças da pressão do ar no tempo. O eixo horizontal é representa o tempo e a curva nos pontos superiores, acima da origem, representa a pressão mais alta e nos pontos inferiores uma pressão mais baixa. A repetição de uma onda é chamada de ciclo, e o número de ciclos dentro do intervalo de um segundo é chamado de frequência, cuja unidade é o hertz(Hz) (MILETTO et al., 2004).

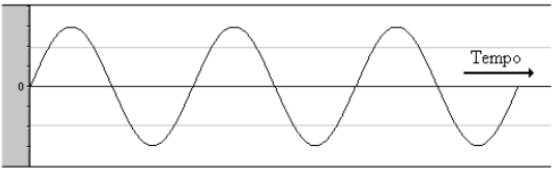


Figura 1 – Representação de uma forma de onda no tempo (MILETTO et al., 2004).

2.1.1.1 Categorias sonoras

Podem ser encontrados dois tipos de sons: complexos e harmônicos(ou parciais). Um som complexo é aquele que formado por ondas simples(senoidais) que são a fundamental e os harmônicos. (MILETTO et al., 2004)

A frequência fundamental ou básica é o componente com mais baixa frequência em um som complexo. Os harmônicos são ou parciais, são múltiplos da frequência fundamental. A frequência fundamental e seus harmônicos constituem a série harmônica (MILETTO et al., 2004).

De acordo com (ORIO et al., 2006), os sons produzidos por um instrumento musical são o resultado de uma combinação de diferentes frequências que compõe uma frequência fundamental(também chamada de F0). Instrumentos de percussão podem não apresentar frequência fundamental e seu som pode ser chamado de barulho, tais barulhos ainda apresentam timbre e volume. Quando diversos sons são tocados em conjunto, são chamados de acorde.

2.1.1.2 Sons musicais e não-musicais

Segundo Miletto et al. (2004), o som pode ser dividido em sons musicais e não-musicais. São considerados sons musicais, aqueles que apresentam vibrações regulares, ou seja, poucos componentes distintos dos harmônicos. Sons não-musicais são aqueles que apresentam vibrações irregulares complicadas, em outras palavras, são sons com muitos componentes harmônicos. A altura tonal de sons não-musicais, não pode ser medida.

2.1.1.2.1 Tons puros

Os tons puros são sons que consistem em uma frequência simples e não contém nenhum outro componente(nem harmônicos). Suas formas de onda são sempre senoidais. Este tipo de onda só pode ser criado artificialmente, não se encontram na natureza (MILETTO et al., 2004).

2.1.1.3 Características de sons musicais

Existem três características básicas de um som musical:

Altura tonal está relacionada com a percepção da frequência fundamental de um som, abrange de tons baixos(ou graves) até tons altos(ou agudos)(ORIO et al., 2006). Sons com alturas tonais diferentes, apresentam frequências distintas. Uma som agudo tem um valor em hertz maior, e um som grave, um valor menor. A faixa de frequências audíveis pelo humano varia de 20 Hz a 20.000Hz (MILETTO et al., 2004).

Volume está relacionado com a amplitude da vibração e sua energia, abrange de suave a forte, também chamada de intensidade(ORIO et al., 2006). A altura de uma onda é chamada de amplitude e determina o volume de um som. Quanto maior a amplitude, o som é mais forte e uma variação na amplitude constitui uma diferença no volume do som (MILETTO et al., 2004).

Timbre é definido como as características do som que permitem diferenciar dois sons com mesma altura tonal e volume (ORIO et al., 2006). Sons com timbres distintos, apresentam formas de onda diferentes. Por exemplo, um som mais suave apresenta formas de onda arredondadas, já um som estridente e penetrante mostra formas de onda pontiagudas (MILETTO et al., 2004).

2.1.1.3.1 Envolvente

Envolventes são as variações dos elementos básicos de um som(altura tonal, timbre e volume) durante um período de tempo. Estas mudanças são o que determinam os timbres característicos de cada som. O exemplo do som de um violino tocado com arco, cujo volume aumenta gra-

dualmente e sua altura tonal e timbre são modificados ligeiramente (MILETTO et al., 2004).

A envolvente da amplitude pode ser descrita em três partes segundo Moorer (1977): a porção de ataque, a porção de sustentação e a porção de decaimento. Miletto et al. (2004) adiciona também uma porção de relaxamento.

2.1.1.4 Terminologia Musical

Na sequência Orio et al. (2006) apresenta alguns termos relevantes:

Andamento refere-se à velocidade que uma obra musical é tocada, ou deve ser tocada. Normalmente é medido em batidas por minuto(bpm) .

Tonalidade de uma música, refere-se ao papel desenvolvido por diferentes papeis em uma obra musical.

Tempo providencia a informação relativa à organização de batidas no eixo de tempo(apresenta-se em números fracionados).

Armadura de Clave com os símbolos # e bemol, apresenta-se como uma representação incompleta da tonalidade, expressa quais notas a serem tocadas de maneira consistente.

2.2 NOÇÕES BÁSICAS DE ÁUDIO

A representação de som chama-se áudio e pode ser tanto analógico quanto digital. Caracteriza-se pelo número de canais, quantidade de bits por amostra chamada de resolução e o número de amostras por segundo conhecido como taxa de amostragem.

Para transformar um som analógico em áudio digital em um computador, precisa-se de um microfone em conjunto com um conversor analógico-digital(ADC) para capturar o som analógico, o caminho inverso requer um conversor digital-analógico(DAC) e um alto-falante (RICE, 2005).

2.2.1 Amostragem

Digitalização de um som, é o processo de representar ele numericamente, para isso é feita a *amostragem* do som. A técnica de *amostragem* consiste em obter um número amostras de uma forma de onda ao longo de um determinado espaço de tempo. Tais amostras são realizadas determinando pontos de amplitude de onda onde cada ponto (instante no tempo) tem sua amplitude representada em um número. (MILETTO et al., 2004)

2.2.2 Taxa de Amostragem

A taxa de amostragem, precisa ser no mínimo o dobro da frequência que está sendo amostrada, em outras palavras, a frequência do som deve estar entre zero e a metade da taxa de amostragem. Caso este limite não seja respeitado, acontecem distorções chamadas de *serrilhamento*. (MILETTO et al., 2004) A taxa de amostragem é importante a medida de que, caso seja muito lenta, a representação será degradada, e se a taxa for muito rápida, o número de amostragens subirá significativamente (WATKINSON, 2001).

2.2.3 Quantização

O processo de quantização nada mais é do que arredondar o nível dinâmico(amplitude) de uma determinada amostra para o valor representável mais próximo (MILETTO et al., 2004). Bosi e Goldberg (2012) acrescenta que o processo de quantificação é inerentemente propenso a perder dados por causa desse arredondamento.

O processo de quantização é ilustrado na Figura 2, onde se mostra uma forma de onda sendo amostrada ao longo de um determinado período de tempo. No ponto 9 de tempo, pode-se observar que o valor amostrado originalmente difere do valor quantificado.

2.3 FORMATOS DE DOCUMENTOS MUSICAIS

Existem diversos formatos para armazenar documentos de áudio, Orio et al. (2006) apresenta eles divididos em duas classes de acordo com duas formas musicais. A primeira, uma forma simbólica onde

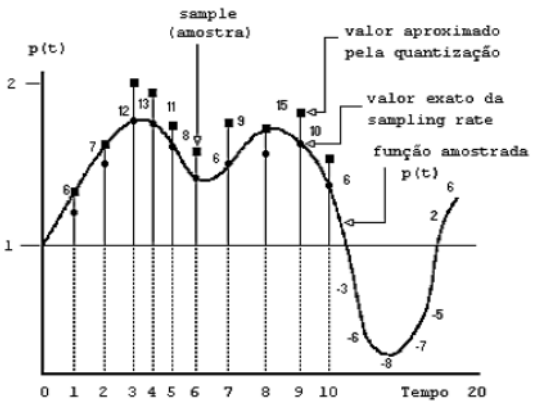


Figura 2 – Processo de quantização (MILETTO et al., 2004).

tem-se partituras que são representações que permitem aos músicos reproduzir a música codificada neles. A segunda, a interpretação musical, onde encontra-se toda a informação codificada na partitura de maneira simbólica, onde o músico pode adicionar novas informações dada a liberdade que a partitura permitir.

2.3.1 Formatos Simbólicos

Segundo Orio et al. (2006) estes formatos propõem-se a produzir uma representação de alta qualidade de partituras musicais. São mais direcionados a representar visualmente as informações das partituras do que a informação nas suas estruturas musicais e dimensões.Orio et al. (2006) apresenta Finale, Siberius e Encore como os softwares de edição de musica simbólica mais populares com interface gráfica. Também apresenta linguagens de marcação para a edição de música simbólica como ABC(apresentada na sequência), GUIDO e MuseData.Good et al. (2001) apresenta MusicXML como uma opção de formato que usa as características do XML para intercâmbio de informações de maneira mais portável. Um exemplo desse formato é a linguagem ABC de notação musical baseada em ASCII. É um dos poucos métodos de codificação que podem ser lidos tanto pelo computador quanto por humanos. Este método é facilmente portado para outros formatos(SØDRING, 2002).

2.3.2 Formatos de Áudio

Segundo (ORIO et al., 2006) estes formatos propõem-se a conter a representação das gravações digitais das performances. A obtenção deses dados digitais se baseia em amostragem de sinais e quantificação desses valores. Métodos que não apresentam compressão, baseiam-se em representação PCM, ou Pulse Code Modulation um codificador que obtém amostras de áudio analógico em intervalos de tempo regulares, onde cada um deles é quantificado e representado de acordo com um código definido para cada faixa de amplitude de onda. Informações relevantes para áudios PCM são a taxa de amostragem(quantidade de amostras igualmente espaçadas na unidade de tempo) e a resolução da amplitude(quantidade de bits usados para a representação da amostra).Este é considerado o codificador mais simples (BOSI; GOLDBERG, 2012). Estes parâmetros são relativos à qualidade percebida do áudio.A seguir estão listadas alguns formatos de áudio:

AIFF De acordo com (SØDRING, 2002)o formato AIFF(Audio Interchange Format File) contém dados não tratados de áudio não comprimido separado em blocos, todos eles associados a um bloco em comum com a descrição das propiedades(sumChannels, sumSampleFrames, sampleSize, sampleRate) para os blocos. Além deste bloco são suportados blocos de Instrumentos(para sintetizar sons), de Marcador (para apontar posições no arquivo), de Comentários e Texto(para armazenar comentários e informações sobre autor e direitos autorais) e por último um bloco de Dados de Som, onde se encontram os dados não tratados de áudio armazenados no arquivo AIFF.

WAV Segundo (SØDRING, 2002), o formato WAV é bastante similar ao AIFF usando diferentes blocos. WAV apresenta um bloco de Formato contendo informação relevante para a reprodução do arquivo. Este formato suporta uma variedade de taxas de amostragem (até 44100Hz) com tamanhos de 8 até 16 bits e utilizando até 2 canais.

AU (SØDRING, 2002) explica que este formato pode armazenar amostras lineares e não-lineares de áudio. Ao fazer armazenamento não linear, atinge-se um nível de compressão onde 12 bits são reduzidos para 8 bits ao se armazenar áudio por logaritmos. Este formato também é bastante versátil ao permitir codificações lineares de 8, 16, 24 e 32 bits de codificação PCM(Pulse Code

Modulation). A pesar da sua versatilidade, não é um formato muito usado fora dos sistemas operacionais derivados do UNIX.

VOC No formato VOC, o arquivo é dividido em um bloco de cabeçalho seguido de blocos de dados, alguns dos quais incluem dados de som não compactado, blocos de silêncio, blocos de repetição e blocos de comentários, de acordo com (SØDRING, 2002).

MIDI (MCNAB et al., 1996) explica que MIDI(Musical Instruments Digital Interface) é um padrão de comunicação e controle de instrumentos musicais eletrônicos, e funciona atribuindo um numero inteiro a cada nota da escala, por exemplo um C é atribuído com 60, a nota seguinte C# com 61 e a anterior B3 com 59. A menor nota possível no MIDI é uma oitava abaixo de C tendo valor 0 (8.176 Hz) e a maior 127 (13344 Hz)(SØDRING, 2002) explica ainda, que arquivos MIDI são os únicos dentre os arquivos de áudio que não armazenam amostras de áudio e sim uma representação operacional da música. Pode se afirmar que é um sistema de controle onde se contem instruções para realizar comandos de ativar notas, predefinir mudanças, eventos e informações de tempo. Arquivos MIDI são compostos de pistas que contem meta-dados e informações de temporização, cada pista podendo conter até 16 canais que armazenam informação relativa ao instrumento sendo tocado.(SØDRING, 2002) Também explica que existem três tipos de arquivo MIDI: Tipo 0, Tipo 1 e Tipo 2. Arquivos de Tipo 0 consistem em uma única pista com informações de andamento e assinatura de tempo. Arquivos de Tipo 1 consistem em múltiplas pistas, com informação do andamento e assinatura de tempo na somente na primeira pista e o sintetizador deve combinar estes meta-dados em um fluxo de eventos antes de sintetizar. E Arquivos de Tipo 2, que consistem de múltiplas faixas, porém com as informações de andamento e assinatura de tempo contidas em cada faixa.

MP3 Segundo (CARPENTIER; BARTHÉLÉMY, 2005), MP3 é uma abreviação para MPEG 1 Audio Layer III. O MP3 realiza uma compressão de dados usando técnicas de codificação que levam em consideração a percepção das ondas sonoras pelo ouvido humano. Por exemplo, se ambos canais de um par de canais de stereo contém a mesma informação, esta redundância é usada para diminuir a quantidade de dados.(SØDRING, 2002) acrescenta que o codificador MP3 realiz asua compressão sobre amostras de áudio

PCM(Pulse Code Modulation) com perdas, ou seja, que as amostras, ao serem reconstruídas, são diferentes das originais, porém, elas soam iguais às originais perceptualmente. Também comenta que é o formato mais popular para transmitir arquivos de áudio pela Internet.

3 SISTEMAS DE RECOMENDAÇÃO

Sistemas de Recomendação são sistemas que ajudam o usuário a encontrar conteúdo, produtos ou serviços agregando e analisando sugestões de outros usuários(PARK et al., 2012).De acordo com Kunaver e Požrl (2017), foram criados para ajudar o usuário a selecionar conteúdo realmente interessante, pois com a crescente quantidade de dispositivos e serviços novos, aceder aos diferentes itens para realizar a escolha torna-se um problema. Assim, sistemas de recomendação rastreiam a interação do usuário com suas escolhas, processam esta informação e uma seleção de itens é filtrada do conteúdo, ajudando a decidir mais facilmente.

Uma definição mais formal, a de (ADOMAVICIUS; TUZHILIN, 2005), explica que o problema de recomendação pode ser formulado da seguinte maneira:

Seja U o conjunto de todos os usuários, e seja I o conjunto de todos os itens possíveis de recomendação, onde tanto o espaço de I quanto o de U pode ser grande(até chegar nos milhões). E seja u uma função de utilidade que mede a utilidade do item i para o usuário u, ou seja, $utilidade : U \times I \rightarrow R$, onde R é um conjunto ordenado. Esta ordenação existe para oferecer os itens do mais relevante ao menos relevante para o usuário.

(RICCI; ROKACH; SHAPIRA, 2011) explica que *item* é o termo genérico usado pra descrever o que é recomendado para o usuário pelo sistema. O valor de um item pode ser positivo se é útil para o usuário, ou negativo se não é apropriado. A modelagem dos itens pode ser feita de acordo com suas as diversas características e propriedades. Usuários, por sua vez, por terem diversas características e objetivos, podem ser modelados de variadas maneiras, o que será definido pela técnica de recomendação escolhida.

Cada elemento do espaço de usuários U pode ser definido com um perfil que inclua várias características de usuário(idade, gênero, salario, etc.)(ADOMAVICIUS; TUZHILIN, 2005). Essencialmente, o modelo do usuário representa suas preferências e necessidades(RICCI; ROKACH; SHAPIRA, 2011). Perfis de usuário são cruciais pro processo de recomendação. Informações sobre perfis de usuário podem ser coletadas de maneira implícita (armazenando informações sobre as ações do usuário para tentar perceber padrões de comportamento) ou explícita (quando o próprio usuário informa ao sistema sua preferencia em relação a objetos apresentados a ele), não se restringindo a utilizar apenas uma destes

maneiras de coleta(PRIMO, 2013).

Da mesma maneira, cada elemento do espaço de itens I é definido por um conjunto de características deste. Normalmente a utilidade é representada por uma avaliação vinda de uma função arbitraria, que pode indicar desde se um usuário em particular gostou do item até o margem de lucro no preço do item(ADOMAVICIUS; TUZHILIN, 2005).

Então, para cada usuário $u \in U$, queremos escolher tais itens $i' \in I$ que maximizem a utilidade do usuário. De maneira mais formal ainda:

$$\forall u \in U, i'_u = arg\ max\ utilidade(u, i) \mid i \in I$$

O problema central em sistemas de recomendação se encontra no fato de que a *utilidade* geralmente não está definida em todo o espaço $U \times I$, mas somente em um subconjunto deste. Em outras palavras, *utilidade* precisa ser extrapolado para o espaço $U \times I$ (ADOMAVICIUS; TUZHILIN, 2005).

Outro conceito relativo a SRs é o de *transação*, que pode ser entendida como uma interação registrada entre um usuário e o SR. Tais dados, normalmente guardados como registros de cada interação humano-computador, podem ser úteis para a geração da recomendação por parte do algoritmo usado pelo sistema. Podem também conter informações sobre a escolha do item pelo usuário, alguns dados sobre o contexto da escolha e/ou feedbacks produzidos pelo usuário (RICCI; ROKACH; SHAPIRA, 2011).

3.1 CLASSIFICAÇÃO DE SISTEMAS DE RECOMENDAÇÃO

De acordo com (BOBADILLA et al., 2013) e (RICCI; ROKACH; SHAPIRA, 2011), um sistema de recomendação pode ser caracterizado pelo seu algoritmo de filtragem. A divisão dos algoritmos de filtragem mais usada, os classifica em: filtragem colaborativa, filtragem baseada em conteúdo, e filtragem híbrida (ADOMAVICIUS; TUZHILIN, 2005). A seguir, será feita uma descrição de cada uma dessas abordagens.

3.1.1 Filtragem Baseada em Conteúdo

Para (BOBADILLA et al., 2013), as recomendações de um sistema baseado em conteúdo se baseiam nas escolhas anteriores feitas pelo usuário. Um sistema baseado em conteúdo analisa um conjunto de do-

cumentos avaliados por um usuário e usa o conteúdo destes, agregados à avaliação do usuário, para inferir um perfil de usuário que seria útil para recomendar outros itens interessantes(PARK et al., 2012) .

Seguindo as definições relativas a SRs dadas anteriormente por (ADOMAVICIUS; TUZHILIN, 2005), em SRs baseados em conteúdo, a utilidade $utilidade(u, i)$ de um item i para um usuário u é estimada baseada nas utilidades $utilidade(u, i_{idx})$ determinadas pelo usuário u aos itens $i_{idx} \in I$ similares ao item i.

A função de utilidade em SR baseados em conteúdo pode ser definida da seguinte maneira:

$$utilidade(u, i) = valor(PerfilBaseadoEmConteudo(u), Conteudo(i))$$

De maneira que $Conteudo(i)$ é uma função que produz um perfil de item, ou seja, um conjunto de atributos que caracterizam o item i. $PerfilBaseadoEmConteudo(u)$ trata-se, por sua vez, da função que produz o perfil do usuário u por meio da análise do conteúdo previamente visto e avaliado pelo usuário, normalmente utilizando análise de palavras-chave. Um exemplo disso é a medida *TF-IDF(Term Frequency/Inverse Document Frequency)*, que é determinado pela frequência de ocorrências de um termo em um documento, em conjunto com a inversa da frequência do termo, por existirem palavras muito frequentes que não são uteis para distinguir documentos relevantes dos irrelevantes(ADOMAVICIUS; TUZHILIN, 2005).

(RICCI; ROKACH; SHAPIRA, 2011) apresenta três componentes que representam três passos do processo de recomendação.

Analizador de Conteúdo A principal tarefa deste componente, é a de representar o conteúdo dos itens, provenientes das fontes de informação, em uma forma que o próximo passo consiga trabalhar com eles. Os itens são analisados usando técnicas de extração de características para atingir a representação necessária. Este passo é necessário quando precisa-se de um pré-processamento pois a informação não é estruturada (RICCI; ROKACH; SHAPIRA, 2011)

Aprendiz de perfis Este módulo é responsável por coletar os dados que representam as preferências do usuário e tenta generalizá-los para, desta maneira, construir o perfil de usuário. Geralmente, para generalizar os dados referentes às preferências de usuário usa-se aprendizado de máquina (RICCI; ROKACH; SHAPIRA, 2011).

Componente de Filtragem Este módulo, por sua vez,utiliza os perfis de usuário para sugerir itens relevantes ao fazer correspondências

entre as representações dos perfis com a dos itens a ser recomendados. Com isto, é possível julgar a relevância de um item de maneira binária(um item somente) ou de maneira continua(uma lista de itens com classificação). (RICCI; ROKACH; SHAPIRA, 2011).

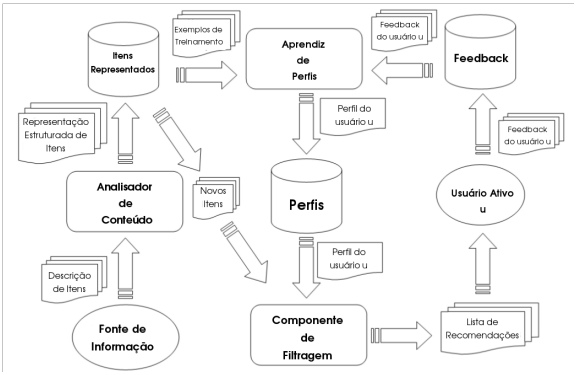


Figura 3 – Arquitetura de um SR baseado em conteúdo(RICCI; ROKACH; SHAPIRA, 2011).

Na Figura 3, (RICCI; ROKACH; SHAPIRA, 2011) mostra detalhadamente a arquitetura de alto nível dos SRs baseados em conteúdo. O primeiro passo é o realizado pelo *Analizador de Conteúdo* que extrai características a partir das *descrições de itens* provenientes da *Fonte de Informação* produzindo representações estruturadas que serão armazenadas no repositório *Itens Representados*.

Para poder construir e/ou atualizar o *perfil do usuário ativo u*, as reações produzidas referentes aos itens são coletadas e armazenadas no repositório *Feedback*. Tais reações, chamadas de feedback ou anotações, junto com as *descrições de itens* relacionados, são utilizadas no processo de aprendizado de um modelo que seja útil para predizer a relevância de novos itens apresentados. Para inicialmente criar o perfil do usuário ativo e armazená-lo no repositório de *Pefis*, o *Aprendiz de Perfis* recebe uma série de exemplos de treinamento já classificados e aplica algoritmos de aprendizagem supervisionada. No caso de um novos itens serem apresentados, o *Componente de Filtragem* faz uma predição sobre se cada item é interessante pro usuário ativo, comparando suas características com as preferências do usuário. Os itens melhor classificados são então colocados na *Lista de Recomendações* para avaliação do usuário, criando um ciclo de aprendizado(RICCI; ROKACH;

SHAPIRA, 2011).

3.1.1.0.1 Vantagens e Desvantagens

SRs baseados em conteúdo apresentam as seguintes vantagens em relação aos SR com filtragem colaborativa (serão apresentados na seção seguinte):

Não dependem de outros usuários para poder realizar uma recomendação pois somente é usado o perfil do usuário ativo(RICCI; ROKACH; SHAPIRA, 2011). São mais transparentes pois é possível explicar o resultado de uma recomendação ao se listar as características que foram usadas para avaliar o item.(RICCI; ROKACH; SHAPIRA, 2011). Também são capazes de recomendar itens que não foram avaliados ainda por nenhum usuário, pois não apresentam o mesmo viés da filtragem colaborativa quando nenhum ou poucos usuários avaliaram item. (RICCI; ROKACH; SHAPIRA, 2011).

SRs baseados em conteúdo porém, também apresentam desvantagens:

As técnicas baseadas em conteúdo utilizadas estão limitadas às características dos itens recomendáveis. O que significa que para ter características suficientes para criar uma boa representação, o conteúdo deve ser extraído automaticamente por um computador ou inserido manualmente(pode ser inviável pela quantidade de recursos). Métodos de extração automática podem apresentar dificuldade de serem usados em itens de multimídia(como é o caso da música).(ADOMAVICIUS; TUZHILIN, 2005).

Outra desvantagem está na sobre-especialização, que ocorre quando o sistema se torna capaz de recomendar apenas itens que sejam muito bem avaliados(tenham pontuação alta) em relação ao perfil do usuário. Isto causa com que o usuário receba sempre recomendações similares às classificadas anteriormente, ou seja, não existe possibilidade de descobrimento para novos tipos de itens.(ADOMAVICIUS; TUZHILIN, 2005)

Uma ultima desvantagem, é a de que um usuário novo não irá receber boas recomendações, pois o sistema ainda não teve oportunidade de aprimorar o seu perfil. (ADOMAVICIUS; TUZHILIN, 2005)

3.1.2 Filtragem Colaborativa

A ideia principal da Filtragem Colaborativa é a de tentar replicar a recomendações feitas entre amigos com gostos em comum no popular *boca-a-boca*. (PRIMO, 2013)

Filtragem Colaborativa é considerada a técnica mais popular e a mais amplamente utilizada. A maneira mais simples de abordar esta técnica é recomendando ao usuário ativo, os itens escolhidos por outros usuários com preferências similares. Tal similaridade é calculada baseando-se no histórico de classificações dos usuários (RICCI; ROKACH; SHAPIRA, 2011). Em outras palavras, SRs com FC tentam prever a utilidade de itens para um usuário em particular baseando-se nos itens avaliados anteriormente por *outros* usuários(ADOMAVICIUS; TUZHILIN, 2005).

Seguindo as definições relativas a SRs já apresentadas por (ADOMAVICIUS; TUZHILIN, 2005), apresenta-se a seguinte definição para sistemas com FC de maneira mais formal: A utilidade $u(c, s)$ do item s para o usuário c é estimada baseando-se nas utilidades $u(c_j, s)$ definidas para o item s pelos usuários $c_j \in S$ similares ao usuário c

3.1.2.0.1 Classificações dos Algoritmos

(PRIMO, 2013) classifica os algoritmos de FC de duas formas relativas à maneira que realizam o processo de recomendação:

Usuário-Usuário Algoritmos do tipo Usuário-Usuário buscam a similaridade entre usuários e tentam prever a opinião de um usuário sobre um item determinado baseando-se na opinião de usuários com avaliações similares que já tenham avaliado o item. Pode ser descrito em três etapas: Avaliação da similaridade do usuário em relação aos demais, seleção de um conjunto de vizinhos e o cálculo da predição.

Item-Item Algoritmos do tipo Item-Item buscam estimar a preferência de um usuário em relação a um item analisando os históricos de acessos entre os perfis. Pode ser descrito em duas etapas: É feita uma recomendação analisando as avaliações dos usuários com mesmo padrão de avaliações que o usuário ativo.

(ADOMAVICIUS; TUZHILIN, 2005) e (BOBADILLA et al., 2013) também apresentam duas classificações para os SR de acordo com os métodos

empregados em sua implementação: os modelos baseados em memória, que acessam a base de dados diretamente e os baseados em modelo que usam as transações para criar um modelo que possa gerar recomendações.

Baseados em memória Modelos baseados em memória basicamente consistem de heurísticas que realizam predições de pontuação baseando-se na coleção dos itens anteriormente avaliados pelos usuários(ADOMAVICIUS; TUZHILIN, 2005). Este método, ao acessar as bases de dados diretamente, se tornam adaptativos a mudanças nos dados, porém requerem um tempo computacional grande de acordo com o tamanho da base de dados(BOBADILLA et al., 2013).

Baseados em modelo Modelos baseados em modelo usam a coleção de avaliações para aprender um *modelo*, que é posteriormente usado para predições de avaliações(ADOMAVICIUS; TUZHILIN, 2005). Este método tem tempo computacional constante, independente do tamanho da base de dados, mas não tem capacidade adaptativa.(BOBADILLA et al., 2013)

De acordo com Bobadilla et al. (2013), o algoritmo mais amplamente utilizado para filtragem colaborativa é o *K Nearest Neighbors (KNN)*. Onde, em uma versão usuário para usuário, são executadas as 3 seguintes tarefas: determinar os k usuários vizinhos para o usuário ativo a, implementar uma *abordagem de agregação* com avaliações para os itens não avaliados por a, extrair predições do passo anterior e selecionar as N primeiras recomendações.

3.1.2.0.2 Vantagens e Desvantagens

SRs com FC apresentam algumas desvantagens similares aos SRs baseados em conteúdo, como é o caso dos seguintes problemas:

Novo Usuário Ao se depararem com um usuário novo no sistema, SRs com FC devem aprender as preferências do usuário antes de fazer recomendações precisas. Dentre as técnicas propostas para solucionar este problema, a maioria são abordagens hibridas que serão abordadas posteriormente neste trabalho (ADOMAVICIUS; TUZHILIN, 2005).

Novo Item Como SRs com FC dependem da avaliação dos usuários para fazer as recomendações, ao se inserir um novo item no sis-

tema, este não é recomendado até que uma quantidade substancial de usuários o avalie. Este problema também encontra sua solução nas abordagens híbridas.(ADOMAVICIUS; TUZHILIN, 2005)

Sparsity A quantidade de avaliações obtidas é geralmente muito menor que o necessário para fazer predições, em qualquer SR, ocasionando que seja importante faze-las a partir de poucos exemplos de maneira efetiva. SR's com FC dependem de uma quantidade grande de usuários para poder realizar as predições com sucesso. Além disso, ao se apresentar um usuário com preferências pouco usuais em relação aos outros usuários, as recomendações não são de boa qualidade por não ter outros usuários de referencia com gostos similares para realizar a comparação. Uma maneira de se sobrepor a este problema, é utilizar informação sobre o perfil do usuário para fazer as comparações entre eles. (ADOMAVICIUS; TUZHILIN, 2005)

3.1.3 Filtragem Híbrida

Comumente, um sistema híbrido se baseia na combinação de filtragem colaborativa com filtragem demográfica ou na combinação de filtragem colaborativa com filtragem baseada em conteúdo, explorando as capacidades de cada uma das técnicas.(BOBADILLA et al., 2013) Também Filtragem hibrida usualmente se baseia em métodos estatísticos ou biologicamente inspirados como algoritmos genéticos, algoritmos genéticos combinados com lógica difusa, redes neurais, redes bayesianas, etc.(BOBADILLA et al., 2013) Também temos o ponto de vista de (BURKE, 2002) que define os sistemas híbridos como sistemas que combinam duas ou mais técnicas de recomendação para ganhar maior performance ao mesmo tempo que evitam os problemas de cada abordagem individual. Para ele, as abordagens mais comuns, consistem em combinar a técnica de filtragem colaborativa combinada com outras.

3.1.3.0.1 Classificação

Segundo (BURKE, 2002) os sistemas híbridos podem ser classificados da seguinte maneira:

Ponderado Em um recomendador híbrido ponderado, calcula-se a

pontuação de um dado item a ser recomendado a partir do resultado das técnicas de recomendação disponíveis no sistema. Uma maneira de fazer isto é por meio de uma combinação linear dos resultados. (BURKE, 2002).

Permuta O sistema utiliza alguns critérios pra trocar entre as técnicas de recomendação. Isto permite que o recomendador seja sensível no nível de itens.

Mista Na recomendação hibrida mista se apresentam recomendações de mais de uma técnica juntas e combinadas. Esta técnica tem a propriedade de evitar o problema de recomendação de itens novos no inicio.

Combinação de características Para obter uma combinação das características das técnicas baseadas em conteúdo e filtragem colaborativa, usam-se técnicas de recomendação baseadas em conteúdo e adicionando informação da filtragem colaborativa nesse conjunto de dados.

Cascata Neste tipo de sistema híbrido, primeiramente é aplicada uma técnica de recomendação para obter um ranking inicial dos candidatos, em seguida é aplicada uma segunda técnica para obter um grupo de candidatos mais refinado para a recomendação.

Acréscimo de características Um recomendador híbrido com acréscimo de características usa uma técnica para produzir uma pontuação ou classificação de um item, e incorpora-se então essa informação no processamento da próxima técnica de recomendação.

Meta-nível Um recomendador híbrido meta-nível é outra maneira de combinar duas técnicas de recomendação, obtendo informação da saída de uma técnica e usando como entrada em uma segunda técnica. A diferença com o acréscimo de características, é que a entrada pro segundo algoritmo não é mais só informação de um item, e sim todo o modelo gerado pela primeira técnica.

3.2 PROBLEMAS TÍPICOS EM SISTEMAS DE RECOMENDAÇÃO

Os desafios de recomendação podem ser listados da seguinte maneira (KUNAVER; POŽRL, 2017):

Esparsidade Trata-se de um problema comum a todos os sistemas de recomendação: trabalhar com conjuntos de dados quase vazios, dado que não existe a possibilidade de obter avaliação de todos os usuários sobre todos os itens disponíveis. Por outro lado, um SR perderia todo sentido se fosse possível obter todas as avaliações de todos os usuários.

Partida a frio Este problema aparece com cada item ou usuário novo no sistema. Para cada item novo, o sistema não tem como categorizar ou dar algum tipo de meta-informação já que não foi avaliado por nenhum usuário ainda. Já para cada usuário novo, ao não se ter um perfil ou modelo para ele, somente pode-se fazer recomendações genéricas.

Big-Data Pode ser encarado como o problema contrário ao problema de esparcidade, pois ao se ter sistemas com números muito grandes, de até bilhões de usuários e itens, precisam-se de estruturas de dados e algoritmos especializados para tratar essa informação.

Sobre Ajuste Acontece quando é gerado um perfil de interesses do usuário muito específicos e especializados. Uma das causas desse problema pode surgir quando o usuário tenta ajudar o sistema de recomendação dando avaliações muito pontuais, com somente seus interesses, o que acaba limitando o espectro de possíveis novas recomendações.

3.3 MÉTODOS DE AVALIAÇÃO DE SISTEMAS DE RECOMENDAÇÃO

(SHANI; GUNAWARDANA, 2011) propõe três maneiras de avaliação para SRs:

Off-line Um experimento off-line é realizado coletando dados de avaliações de itens pelos usuários. Com esses dados é possível tentar simular o comportamento que seria demostrado ao interagir com o SR. Ao fazer isso, assume-se que estes dados de avaliações serão similares ao comportamento do usuário quando o sistema entrar em produção. Este tipo de experimento é usado pelo seu baixo custo pois não requer interação com usuários reais.

Testes com usuários Um teste com usuários é realizado recrutando um conjunto de sujeitos de pesquisa e solicitando que realizem várias tarefas que requerem interação com o SR. Enquanto estes

sujeitos realizam as tarefas, seus comportamentos são observados e registrados, coletando uma medidas quantitativas, tais como a porcentagem de tarefas realizadas, acurácia dos resultados, ou o tempo para realizar a tarefa. Também são feitas perguntas qualitativas antes, durante e depois os testes.

On-line Um experimento on-line visa obter a verdadeira medida do real ganho de valor que o sistema de recomendação pode trazer. Normalmente é realizado pela análise de indicadores de desempenho, e pode usar algumas técnicas de experimentação, como os testes AB, ou testes de divisão em grupos de tratamento e controle.

3.3.1 Precisão e Revocação

(AL-BASHIRI et al., 2018) explica que tradicionalmente, sistemas de recomendação utilizam as métricas *precisão* e *revocação* para medir a performance de SR.
A *Precisão* pode ser determinada pela fração de itens que foram avaliados pelos usuários no conjunto de testes e que foram efetivamente recomendados pelo SR. Representa a razão entre os itens recomendados relevantes com o total de itens recomendados pelo sistema. Já a *Revocação* representa a fração dos itens relevantes que foram capturados. Para entender a definição dessas métricas, o conhecimento dos seguintes termos é necessário:

VP ou Verdadeiro Positivo, que é o número de itens interessantes ao usuário que são recomendadas.

FN ou Falso Negativo, que é o número de itens interessantes ao usuário que não são recomendadas.

VN ou Verdadeiro Negativo, que é o número de itens interessantes ao usuário que não são recomendadas.

FP ou Falso Positivo, que é o número de itens não interessantes ao usuário que são recomendadas.

As definições matemáticas de ambas métricas se encontra a seguir:

$$Precisão = \frac{VP}{VP + FP}$$

$$Revocacão = \frac{VP}{VP + FN}$$

(CRASWELL, 2009) apresenta também uma maneira de medir a precisão de uma maneira mais adequada a conjuntos de dados grandes, que é a da *Precisão em n*, onde os primeiros n documentos apresentados na recomendação são considerados, esse valor de n pode ser escolhido em base a quantos itens são recomendados ao usuário. Considerando R todos os documentos relevantes e r os que foram obtidos na listagem final da recomendação.

$$Precisão\ em\ n = \frac{r}{n}$$

$$Revocacão\ em\ n = \frac{r}{R}$$

Uma possibilidade é que em um cenário que não se tenham identificado todos os itens relevantes, pode não ser viável calcular o R.

(STECK, 2013) apresenta, por último a métrica Mean Average Precision (ou MAP), que aponta como a mais adequada para SRs, isto por que enquanto a Precisão em n captura somente o número de itens relevantes nos primeiros N itens, enquanto MAP também leva em consideração a posição na lista de itens. Isto se deve a que na maioria dos SRs, as listas de itens recomendados é ordenada de mais relevante para o menos. É definido pela seguinte fórmula:

$$MAP = \frac{1}{m^+} \sum_{i}^{N} \frac{i}{Rank_{pos,i}^{+,N}}$$

aonde $Rank_{pos,i}^{+,N}$ é o elemento i na lista ordenada dos N itens relevantes(isto é, em ordem ascendente, onde o item melhor avaliado tem a menor posição *pos*). O número total de elementos relevantes é denotado por m^+.

3.4 APRENDIZADO DE MÁQUINA EM SISTEMAS DE RECOMENDAÇÃO

O aprendizado de máquina converte dados em predições ou classificações em diferentes aplicações, como é o caso de identificação de conteúdo, sistemas de reconhecimento, classificação e busca, filtros de spam e também, como foco deste trabalho, sistemas de recomendação. (LEE; SHIN; REALFF, 2017).

(MICHALSKI; CARBONELL; MITCHELL, 2013) define Aprendizado de Máquina como o estudo e modelagem computacional de processos de aprendizagem nas suas diversas manifestações, que incluem a adquisição de novos conhecimentos declarativos, o desenvolvimento de novas habilidades cognitivas por meio de instrução ou prática e organização de novos conhecimentos no geral, novas representações e novos fatos e teorias por meio da experimentação.

3.4.1 Conceitos Básicos

Em seguida são apresentados alguns conceitos e definições básicas utilizadas na literatura, de acordo com (MONARD; BARANAUSKAS, 2003).

Indutor Um indutor (ou algoritmo de indução) tem como objetivo extrair um bom classificador a partir de exemplos rotulados. Este pode ser usado para classificar exemplos ainda não rotulados, com objetivo de predizer o rótulos deles corretamente(MONARD; BARANAUSKAS, 2003). A indução é uma forma de inferência lógica que permite obter conclusões a partir de um conjunto de exemplos. Se caracteriza por um raciocínio que generaliza da parte para o todo. As hipóteses geradas por este processo podem ou não preservar a verdade, pois os conceitos, na indução, são gerados pela inferência indutiva sobre os exemplos apresentados por um processo externo ao sistema(REZENDE, 2003).

Exemplo Um exemplo (ou caso), é uma tupla de valores de atributos, e descreve o objeto de interesse.

Atributo Descreve uma característica ou aspecto do exemplo. Podem ser nominais, quando não existe ordem entre os valores,(i.e. cor: vermelho, azul) ou contínuos, quando existe uma ordem entre os valores(i.e. peso $\in \Re$).

Classe A classe (ou rótulo) descreve o fenômeno de interesse e é a meta que se deseja aprender e fazer previsões. Pertencem a um conjunto discreto de classes $\{C_1, C_2, C_3\}$.

Conjunto de Exemplos É composto por exemplos contendo valores de atributos e sua classe.

A Tabela 1 mostra um conjunto no formato atributo-valor, onde apresentam-se o conjunto T com n exemplos e m atributos.

Tabela 1 – Conjunto de Exemplos na forma de atributo-valor.

	X_1	X_2	...	X_m	Y
T_1	x_{11}	x_{12}	...	x_{1m}	y_1
T_2	x_{21}	x_{22}	...	x_{2m}	y_2
...	...	...	...	...	...
T_n	x_{n1}	x_{n2}	...	x_{nm}	y_n

Como pode se inferir a partir da tabela, exemplos são tuplas $T_i = (x_1, x_2, ..., x_{im}, y_i) = (x_i, y_i)$, logo x_i é um vetor e $y_i = f(x)$ é a função que tenta-se predizer.

Usualmente um conjunto de exemplos é dividido em um conjunto de treinamento e um conjunto de teste, estes são conjuntos disjuntos.

Classificador Ao receber um conjunto de exemplos de treinamento, o indutor gera um classificador(descrição de conceito ou hipótese), por meio dele é capaz de definir o rótulo de um novo exemplo.

Ruído São dados imperfeitos, que podem ter sido gerados tanto na adquisição dos dados quanto na classificação incorreta dos rótulos.

Bias Trata-se de uma preferência de uma hipótese sobre outra além da consistência com os exemplos.

Under e Overfitting Overfitting acontece quando a hipótese gerada resulta ser muito específica para um conjunto de treinamento, isto pode ser detectado ao testar a hipótese no conjunto de teste e obter um desempenho ruim. Underfitting acontece quando a hipótese ajusta-se muito pouco ao conjunto de treinamento, isto é causado por um conjunto de treinamento com poucos exemplos representativos.

Overtuning Trata-se do excesso de ajuste do algoritmo em todos os exemplos disponíveis para otimizar seu desempenho, ou seja, não há separação entre conjunto de treinamento e conjunto de teste .

Poda Trata-se de uma técnica para lidar com ruído e overfitting, por meio do aprendizado de uma hipótese bem genérica a partir do conjunto de treinamento para melhorar o desempenho perante exemplos não vistos.

Completude e Consistência A completude de uma hipótese refere-se à sua capacidade de classificar todos os exemplos, já a sua consistência refere-se à sua capacidade de classificar todos os exemplos corretamente.

3.4.2 Aprendizado Supervisionado e Não-supervisionado

No aprendizado supervisionado, um conjunto de exemplos de treinamento, com rótulo já conhecido, é fornecido ao algoritmo de aprendizado, ou indutor. Como foi definido anteriormente, um exemplo é descrito por um vetor de seus atributos e o rótulo da classe associada. O indutor tem como objetivo criar um classificador capaz de rotular exemplos novos não-rotulados(sem classe). Quando os valores de rótulos de classe são discretos, o problema passa a ser chamado de classificação. Quando se usam valores de rótulos de classe contínuos, o problema é chamado de regressão.(MONARD; BARANAUSKAS, 2003).

No aprendizado não-supervisionado, a figura do professor é ausente, logo, o indutor precisa analisar os exemplos fornecidos e tenta determinar se é possível formar agrupamentos ou clusters. Normalmente, após os agrupamentos serem determinados, é necessário realizar uma analise para determinar o que cada agrupamento significa no contexto do problema(MONARD; BARANAUSKAS, 2003).

3.4.3 Técnicas de Aprendizagem

3.4.3.1 Processo de Classificação

De acordo com (DOMINGOS, 2012), Classificação é o aprendizado mais maduro e mais amplamente utilizado. Um classificador é um sistema que recebe um vetor de valores discretos ou contínuos e retorna o valor discreto referente à classe. O processo de classificação encontra-se descrito na Figura 4, a entrada ao indutor pode ser melhor escolhida com conhecimento sobre o domínio de um especialista. Após a indução, a saída é avaliada pelo especialista e o processo pode ser repetido com o ajuste de alguns dados ou parâmetros, obtendo no fim do processo um classificador.

Algumas técnicas de aprendizagem são apresentadas a seguir:

k-Nearest Neighbor (ou K-Vizinhos mais próximos) consiste em encontrar os k pontos no conjuntos de treinamento mais próximos

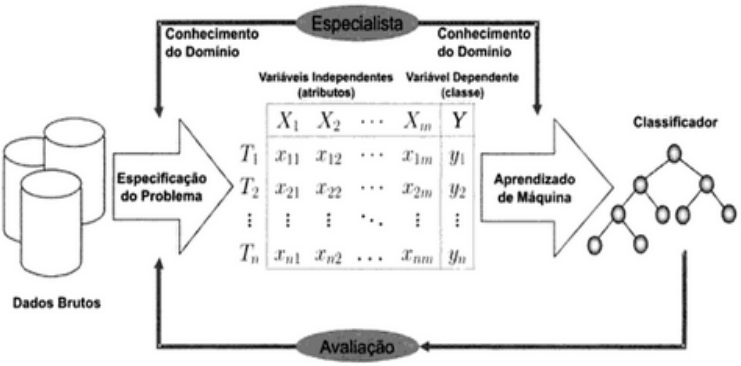


Figura 4 – Processo de Classificação.(REZENDE, 2003)

do ponto de teste, e rotular o ponto de teste pela votação da maioria dos k pontos.(RÄTSCH, 2004) Dado um ponto qualquer a ser classificado, um classificador KNN encontra os k pontos mais próximos, ou vizinhos do conjunto de dados, então determina sua classificação de acordo com os vizinhos mais próximos. A grosso modo, a ideia é que se um objeto cair onde uma classe específica é predominante, é por que também deve pertencer a essa classe (RICCI; ROKACH; SHAPIRA, 2011) Uma das vantagens desse tipo de classificador é o fato dele ser do tipo *lazy* ou preguiçoso, isto é, que não requer que se mantenha um modelo, logo o sistema pode se adaptar a mudanças rápidas na matriz de avaliações de usuário, que é outro dos requisitos necessários para este sistema.

Análise Discriminante Linear funciona computando um hiperplano no espaço de entrada, minimizando a variância entre classes e maximizando a distancia entre elas(RÄTSCH, 2004).

Árvores de Decisão Estes algoritmos funcionam por meio da partição repetitiva do espaço na entrada, tentando montar uma arvore com os nodos o mais puros possível(com uma só classe). Um novo exemplo será classificado percorrendo a árvore a partir do nodo até as folhas(RÄTSCH, 2004).

Redes Neurais São modelos computacionais inspirados pela conectividade de sistemas nervosos animados(RÄTSCH, 2004).

Máquina de Vetores de Suporte (Support Vector Machines - SVMs)

funciona por meio do mapeamento do conjunto de treinamento em um espaço de atributos com ajuda de uma função núcleo e separando os dados usando um hiperplano(RÄTSCH, 2004).

k-Means Clustering (ou Agrupamento pelas K-Médias) tem como objetivo particionar n observações dentre k grupos onde cada observação pertence ao grupo mais próximo da média(CUNNINGHAM,).

Fuzzy Clustering trata-se de uma generalização do algoritmo K-Means Clustering, o que permite que objetos façam parte de diferentes Clusters(Agrupamentos), definindo o grau de pertinência pelos peso probabilístico(CUNNINGHAM,).

Clustering Hierárquico objetiva gerar uma hierarquia de conceitos para produzir clusters aninhados e formar uma estrutura em forma de árvore. Caso se opte por utilizar uma estrategia bottom-up para gerar esta estrutura, o algoritmo será Aglomerativo. Caso o algoritmo comece este processo com um só cluster com os n elementos e fizer a divisão, pertencerá aos algoritmos Divisivos(CUNNINGHAM,).

3.4.3.2 Técnicas de Similaridade

Em SRs é muito comum precisar calcular a similaridade entre dois itens ou usuários, em seguida apresentaremos medidas de similaridade para este propósito:

Coeficiente de Semelhança Simples O Coeficiente de Semelhança Simples é uma medida de similaridade para objetos com diversos atributos onde para cada atributo se conta as presenças e ausências de cada um de maneira igual. Este método apresenta como resultado o valor 1 quando os objetos são completamente iguais e 0 nos casos em que são totalmente distintos, tipicamente os valores que este calculo assume oscilam entre 0 e 1 (TAN, 2018). Esse cálculo é realizado da seguinte maneira:

Sejam x e y objetos que consistem de n atributos, a comparação de ambos objetos, que são dois vetores, é feita a través das frequências

f_{00} = o número de atributos não presentes em x e não presentes em y

f_{01} = o número de atributos não presentes em x e presentes em y
f_{10} = o número de atributos presentes em x e presentes não em y
f_{11} = o número de atributos presentes em x e presentes em y

$$c(x, y) = \frac{f_{11} + f_{00}}{f_{01} + f_{10} + f_{11} + f_{00}}$$

Similaridade Cosseno É uma medida que calcula o quão similares dois vetores são, é definida da seguinte maneira(TAN, 2018):

Sejam x e y dois vetores

$$cos(x, y) = \frac{x \cdot y}{||x|| \cdot ||y||}$$

onde o produto escalar $x \cdot y = \sum_{k=1}^{n} x_k y_k$ e $||x||$ é tamanho do vetor x. $||x|| = \sum_{k+1}^{n} x_k^2 = \sqrt{x \cdot x}$.

Um olhar mais atento permite perceber que esse cálculo de similaridade trata apenas da diferença de ângulo entre 2 vetores x e y, tanto que se o resultado da similaridade for 1, significa que ambos estão a 0^o e por tanto são iguais, sem considerar o tamanho de cada. Se a similaridade cosseno tem como resultado 0, então o angulo entre x e y é de 90^o e por tanto não tem características em comum. A figura 10 ilustra essa situação, aonde θ representa o ângulo da distância entre os vetores x e y

3.5 RECOMENDAÇÃO MUSICAL

Recomendação musical é o procedimento de apresentar ao ouvinte uma lista de peças musicais que ele provavelmente gostaria de ouvir, para isso, deve basear-se em um bom entendimento das preferências do usuário e das músicas da coleção deste (SHAO et al., 2009). Recomendação musical é um tanto diferente da recomendação de itens de diferentes domínios (tais como livros ou filmes), isso se deve a que na recomendação musical, deve ser levar o contexto em consideração com maior relevância, pois as pessoas mudam o seu consumo de músicas de

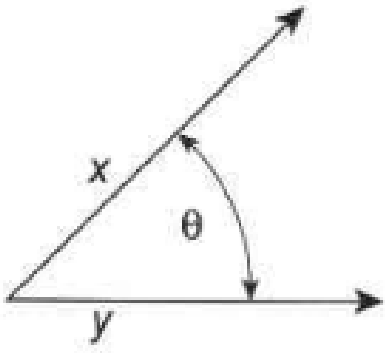


Figura 5 – Ilustração geométrica da medida cosseno(TAN, 2018)

acordo com seu contexto. Por exemplo, uma mesma pessoa pode gostar de ouvir pop-rock ao acordar mas ouvir musica clássica pra trabalhar, e jazz antes de dormir. Em relação às técnicas aplicadas, deve-se levar em consideração que um usuário pode ouvir as músicas várias vezes, inclusive de maneira repetida e continua (CELMA, 2010). Estes múltiplos parâmetros influenciam as preferências de usuários, que podem variar de ser fatores geográficos a sociais, fazem com que a contagem de itens recomendáveis seja muito grande. Este número pode ser reduzido por meio da recomendação de álbuns inteiros ou artistas mas isso nem sempre é compatível com a aplicação em questão. Além disso, não leva em consideração de que repertórios de um mesmo artista podem variar de diversas maneiras, ou que usuários podem gostar mais de uma do que de outra música. (JAYASHREE; MANIAN; SRIVATSAV, 2016)

Os objetivos principais da recomendação musical são, de acordo com (SHAO et al., 2009):

Alta precisão na recomendação Um bom SR devia de produzir uma lista relativamente curta de músicas, onde a maioria é do agrado do usuário.

Alto grau de inovação na recomendação Inovação pode ser definida como uma rica variedade de artistas e uma variedade bem balanceada no conteúdo musical(timbre, ritmo, altura tonal). Uma variedade bem balanceada significa que o conteúdo musical é diverso e informativo, sem divergir das preferências do usuário.

3.5.1 Representação de perfis de Usuários

A modelagem de usuário é um dos aspectos mais importantes na recomendação musical, pois nela é modelada a diferença entre perfis. Por exemplo, região geográfica, idade, ou inclusive gênero, estilo de vida e interesses podem afetar nas preferências musicais do usuário. A recomendação musical também se beneficia ao incluir os estados de ânimo do usuário (JAYASHREE; MANIAN; SRIVATSAV, 2016). As pessoas, no geral, consomem música em diversos contextos, por exemplo: rock ao acordar, música clássica ao trabalhar ou jazz enquanto jantam. Por tanto, um recomendador musical precisa ser capaz de lidar com informação contextual complexa(HERRADA, 2009).

3.5.2 Representação de perfis de Itens

A pesar das diversas técnicas para descrever áudio e música, existe uma lacuna semântica entre as propriedades que é possível extrair da música e a percepção humana, o que dificulta muito a recomendação musical. Existem três categorias de meta-dados para o gerenciamento de conhecimento musical: (CELMA, 2010)

Editoriais dados inseridos pelo editor, como autor, estrutura do conteúdo, etc. Usados por SRs não baseados em conteúdo.

Culturais dados inseridos por usuários em geral, como gênero, artista, tags. Usados por SRs baseados em contexto.

Acústicos são dados obtidos pela extração de propriedades de áudio fazendo um analise de conteúdo.

4 SISTEMAS DE RECOMENDAÇÃO SENSÍVEIS AO CONTEXTO

SRs convencionais normalmente se valem de dos dados do histórico dos usuários, porém esta abordagem assume que o comportamento do usuário não muda rapidamente e que observações armazenadas no passado podem ajudar a prever o comportamento futuro. Estas assunções são válidas somente até certo ponto.(BALTRUNAS, 2008).

(DEY; ABOWD, 2000) define que sistemas são sensíveis ao contexto se usam contexto para providenciar informação relevante ou serviços ao usuário, onde a relevância depende da tarefa realizada.

Tradicionalmente, SRs lidam somente com dois tipos de entidades: usuários e itens, e se baseiam em avaliações conhecidas para estimar classificações de itens ainda não vistos por usuários.

Informação sobre o contexto pode ser adicionado de maneira que a função de avaliação R é definida por:

$R : Usuários \times Items \times Contexto \rightarrow Avaliações$

onde $Usuários$ e $Items$ são conjuntos de usuários e de itens respectivamente,$Avaliações$ é um conjunto discreto e finito de avaliações, e finalmente $Contexto$ é um conjunto de *atributos contextuais* K que podem ter alguma estrutura elaborada. Tais estruturas podem ser desde muito complexas ou até mais simples onde K é definido por um conjunto q de atributos $K = (K_1, K_1, ..., K_q)$ (PANNIELLO et al., 2009)

Essas informações podem mudar muito a predição da recomendação e que a tomada de decisões por parte do consumidor pode ser alterada pelo contexto de acordo com estudos comportamentais. Para ele, contexto pode ser definido como informação adicional que possivelmente é relevante para a recomendação.(STEEG, 2015)

4.1 CONTEXTO

A definição de contexto depende muito do campo de estudo onde se revisa a definição de contexto, e para tentar trazer uma definição mais relativa a SRs ao trazer as definições de campos relacionados contexto é definido como (ADOMAVICIUS; TUZHILIN, 2011):

Mineração de dados(Data Mining) Segundo esta comunidade, contexto é definido como eventos que caracterizam os estágios da vida do consumidor e podem determinar mudanças em suas pre-

ferências, status e valor para uma companhia.

Sistemas sensíveis ao contexto mobile e ubíquos Para esta área, contexto é definido como a localização do usuário, a identidade das pessoas perto dele, os objetos ao redor e mudanças nestes elementos. Também são consideradas datas, estações, temperatura. E ainda estados físicos e emocionais do usuário.

Segundo Primo (2013), informações demográficas também podem ser utilizadas como contexto do usuário. É possível classificar os contextos em quatro tipos: localização, tempo, atividade e identidade. (JAYASHREE; MANIAN; SRIVATSAV, 2016)

Já (DEY, 2001) define contexto, de uma maneira mais geral, como qualquer informação a ser usada para caracterizar a situação de uma entidade, seja esta uma pessoa, lugar ou objeto considerado relevante para a interação usuário-aplicação (incluindo eles próprios). Esta é a definição utilizada neste trabalho.

4.2 PARADIGMAS DE INCORPORAÇÃO DE CONTEXTO

Para (ADOMAVICIUS; TUZHILIN, 2011) e (STEEG, 2015) existem três paradigmas para incorporar o contexto com a recomendação utilizada.

4.2.1 Pre-filtragem contextual

Na Pré-filtragem contextual os dados para avaliação são filtrados de acordo com um certo contexto antes de que o SR calcule as predições. Assim, as avaliações são preditas usando um subconjunto de dados de treinamento que contém somente avaliações relevantes ao contexto. (STEEG, 2015)

Na Figura 6 apresenta-se a contextualização dos dados de entrada, a informação contextual c é utilizada para direcionar a seleção de dados ou construir conjuntos de dados bidimensionais relevantes(sendo $Usuário \times Item$ as duas dimensões), assim, pode-se usar métodos de recomendação convencionais. Esta é uma das suas maiores vantagens. (ADOMAVICIUS; TUZHILIN, 2011)

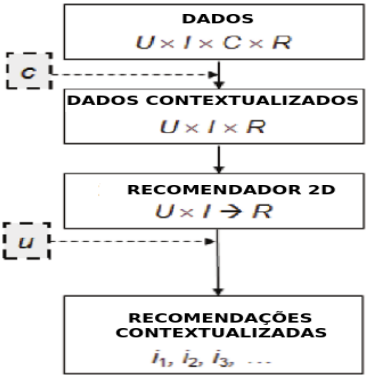


Figura 6 – Incorporação de contexto com pré-filtragem contextual (ADOMAVICIUS; TUZHILIN, 2011).

4.2.2 Pós-filtragem contextual

Na Pós-filtragem contextual, a predição é feita baseando-se nos dados de avaliação em sua totalidade, mas somente é feita a recomendação dos resultados relevantes para o contexto. (STEEG, 2015)

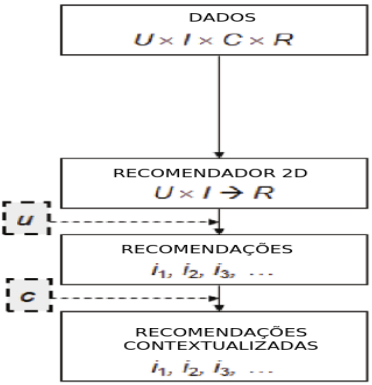


Figura 7 – Incorporação de contexto com pós-filtragem contextual (ADOMAVICIUS; TUZHILIN, 2011).

Na Figura 7 é apresentada a abordagem de pós filtragem contex-

tual onde a informação de contexto nos dados de entrada é ignorada, ou seja, quando a lista dos melhores itens candidatos a recomendação é gerada. Então, a lista de recomendações é ajustada de acordo com cada usuário, usando a informação contextual. Isto pode ser feito, tanto filtrando as recomendações que são irrelevantes pro contexto, quanto ajustando o ranking das recomendações na lista, baseando-se no contexto. (ADOMAVICIUS; TUZHILIN, 2011)

4.2.3 Modelagem contextual

Na modelagem contextual, o contexto é usado diretamente enquanto se realiza a predição. Isto requer um tipo diferente de recomendador, onde dados multidimensionais podem ser usados para a predição.(STEEG, 2015).

Nesse paradigma, a informação contextual é diretamente aplicada na modelagem dos métodos de classificação, ou seja, a função de recomendação já é contextualizada.

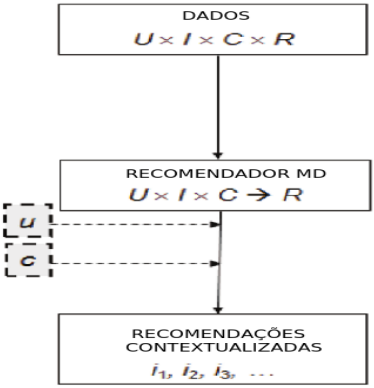


Figura 8 – Incorporação de contexto com modelagem contextual (ADOMAVICIUS; TUZHILIN, 2011).

Na Figura 8 é ilustrada a maneira como a informação contextual é diretamente usada na função de recomendação como um preditor explícito da avaliação do usuário para o item. Ao contrário da modelagem usada pelas abordagens de pré e pós filtragem que utilizam modelos de duas dimensões, na modelagem contextual funções de recomendação multidimensionais são utilizadas, que essencialmente são

modelos preditivos ou cálculos heurísticos que incorporam o contexto em adição ao usuário e itens. (ADOMAVICIUS; TUZHILIN, 2011)

4.3 MÉTODOS DE OBTENÇÃO DE CONTEXTO

Informação contextual pode ser obtida das seguintes maneiras:

4.3.1 Explicitamente

Nesse método, pessoas relevantes ou outras fontes de informação contextual são diretamente abordadas, seja por meio de perguntas diretas ou elicitando tal informação por outros meios. Um exemplo claro seria um site na Web que solicita ao usuário o preenchimento de algum formulário ou a resposta de perguntas para permitir o total acesso ao site. (ADOMAVICIUS; TUZHILIN, 2011).

4.3.2 Implicitamente

Implicitamente, é possível obter informação contextual ao se acessar diretamente a fonte de dados e usar esses dados relativos ao ambiente do usuário, sem ter algum tipo de interação com o usuário. Por exemplo, as mudanças no local do usuário detectadas por uma companhia de telefonia, informação temporal sendo extraída de transações (ADOMAVICIUS; TUZHILIN, 2011)

4.3.3 Inferência

Por meio do uso de estatística ou de mineração de dados, o contexto pode ser inferido. Para isto, torna-se necessária a construção de um modelo preditivo (um classificador) e o treinamento deste com dados apropriados. A qualidade de tal classificador influencia significativamente na inferência da informação contextual, e varia em diferentes aplicações. (ADOMAVICIUS; TUZHILIN, 2011)

4.4 TRABALHOS RELACIONADOS

4.4.1 Improvise

Em (DIAS; FONSECA; CUNHA, 2014) é apresentado um sistema de recomendação sensível ao contexto chamado Improvise, o qual apresenta uma capacidade de recomendar músicas apropriadas para diferentes atividades.
A arquitetura do Improvise pode ser descrita como um sistema que recebe como entrada a atividade que o usuário pretende realizar com os gêneros que deseja ouvir, então, cria-se uma associação entre as atividades e as características das músicas obtidas por meio da *API EchoNest* [1]. Após consumir o conteúdo, o usuário retorna um feedback que auxiliará na recomendação das próximas músicas. Para delimitar o contexto, foram consideradas cinco atividades: caminhar, relaxar, correr, dormir e comprar. O recomendador foi construído de maneira que tivesse dois modos de funcionamento, um onde as recomendações eram centradas no usuário, logo as avaliações no histórico dele eram levadas em consideração, o outro, onde se tentava fazer uma recomendação genérica considerando as características extraídas das músicas. Para avaliar os resultados obtidos, foram feitos testes com usuários, onde estes respondiam a questionários de satisfação após interagir com o recomendador.

4.4.2 Move

Silva e Bernardini (2014) mostra um sistema de recomendação musical que tem como objetivo indicar qual é a melhor música pra determinada atividade física. O nome do sistema é MOVE e foi desenvolvido com a ferramenta *Weka*(especializada em AM) para construção do classificador que se baseia em árvores de decisão J48. Também foi usada a *API EchoNest* para extração de atributos dos áudios para a criação do conjunto de treinamento, as atividades que foram consideradas para a classificação pelo sistema foram *Correr, Pedalar, Caminhar*, logo foram criadas classes para cada uma dessas atividades. A maneira de incorporação de contexto se dava por meio de uma inferência, aonde o sistema de recomendação media a velocidade do usuário dada os dados coletados pelo *GPS*, assim inferia a atividade sendo realizada e qual

[1] http://the.echonest.com/

classe de músicas devia ser recomendada. Este ST leva em consideração apenas o contexto que o usuário e a relação entre as características das músicas e as classes de contextos, logo a técnica utilizada por este sistema trata-se de uma filtragem baseada em conteúdo.

4.4.3 Lifetrack

Por último, Reddy e Mascia (2006) demonstra também um sistema de recomendação musical sensível ao contexto chamado Lifetrack, o qual possui capacidade de inferir o contexto atual do usuário a través dos sensores no celular (GPS, acelerômetro, microfone, tempo), complementado com informações sobre temperatura e similares, e recomendar uma música adequada à atividade sendo executada. A maior diferença deste recomendador para outros é que não realiza um análise do áudio, ao invés disso, solicita que o usuário insira *tags* nas músicas previamente com informação contextual(i.e. *Manhã*, *Quente*) para modelagem e treinamento do classificador, após isso, perante cada novo contexto, o sistema de recomendação tenta fazer uma recomendação. Outro detalhe interessante é que tem um módulo para *equalização de contexto*, em outras palavras, permite ajustar que tanta importância é dada para cada aspecto do contexto, (i. e. maior atenção à informação temporal e menos à espacial). O Lifetrack se vale das tags fornecidas por diverss usuários para utilizar filtragem colaborativa.

A Tabela 2 mostra um comparativo entre todos os trabalhos relacionados, oferecendo assim um panorama do estado da arte em sistemas de recomendação musicais sensíveis ao contexto.

Tabela 2 – Comparativo entre os trabalhos relacionados. Fonte: Elaboração Própria

Sistema	Sensível ao Contexto	Incorporação Contextual	Atividades Contempladas	Tipo de Filtragem	Recomendação Genérica	Recomendação Personalizada
Improvise	Sim	Explicita	Caminhar, Meditar, Correr, Dormir, Comprar	Híbrida	Sim	Sim
Move	Sim	Inferida	Correr, Pedalar, Caminhar	Baseada em Conteúdo	Sim	Não
Lifetrack	Sim	Inferida	Parado, Caminhar, Correr, Dirigir	Colaborativa	Sim	Sim

5 MODELO E IMPLEMENTAÇÃO DE UM SISTEMA DE RECOMENDAÇÃO MUSICAL SENSÍVEL AO CONTEXTO

Seguindo a ideia de (JAYASHREE; MANIAN; SRIVATSAV, 2016) de que a maioria das pesquisas unicamente levam em consideração as transações e dados de preferências, e que um caminho para a evolução de sistemas de recomendação musicais é a de envolver o contexto que cercam os usuários, este trabalho se propõe a desenvolver um sistema de recomendação que se baseia no contexto de uma atividade para fazer uma recomendação, visando esclarecer a hipótese de que um recomendador contextual tem um desempenho melhor que um que desconsidera o contexto.

Como resultado do estudo realizado é possível elaborar uma lista de requisitos mínimos desejáveis para um Sistema de Recomendação Musical Sensível ao Contexto. Estes requisitos são os seguintes:

- Identificação de usuários com perfis similares
- Identificação de contextos similares
- Adaptabilidade a mudanças no contexto
- Recomendação de músicas de acordo com o interesse e o contexto dos usuários

Considerando os requisitos apresentados acima, foi definido um modelo para o sistema de recomendação. Como explicado anteriormente um SR baseado em memória tem capacidade de adaptação a mudanças nos dados. Por este motivo foi o método escolhido para a implementação do SR, dado que o mudanças no contexto de uso são recorrentes.

Outro fator decisivo na escolha das técnicas foi a necessidade de se ter recomendações altamente personalizadas e com um alto grau de inovação, características inerentes à recomendação musical. Por estes motivos decidiu-se construir um SR com uma abordagem híbrida, ou seja, que aplicassem recomendação por Filtragem Colaborativa e Baseada em Conteúdo. Além disso, como para cumprir com o requisito de recomendar de acordo com o contexto em que o usuário está inserido, aplicamos também uma filtragem contextual.

A Figura 9 mostra a arquitetura seguida pelo SR, onde se observa a abordagem híbrida que apresenta uma etapa de filtragem colaborativa, seguido por uma etapa de pós-filtragem contextual e em

seguida uma última etapa de filtragem baseada em conteúdo. Segundo a classificação de recomendadores híbridos apresentada, este modelo demonstra uma abordagem em cascata, onde a saída de um método se torna a entrada do próximo.

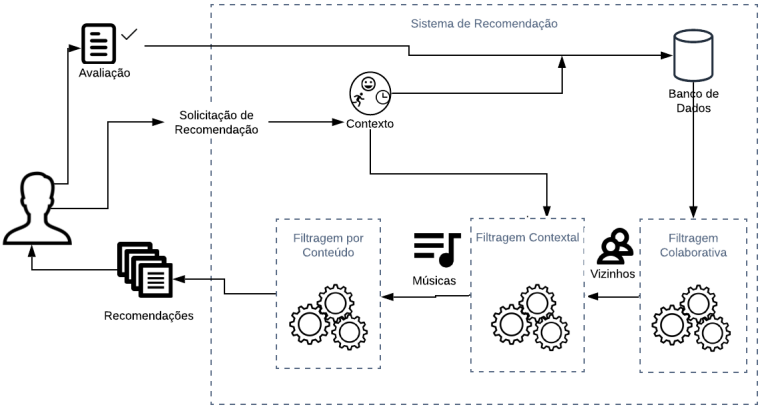


Figura 9 – Processo de Recomendação(Fonte: Elaboração própria)

Em resumo, este sistema recebe como entrada um usuário específico para o qual se deseja realizar a recomendação e informação sobre o contexto, com a atividade a ser realizada e a cultura à qual pertence o usuário, e tem como saída uma lista de n recomendações de músicas que seriam adequadas para o contexto apresentado e de acordo com o gostos do usuário que foram registrados anteriormente. De maneira mais clara e objetiva, o que tenta-se classificar são as músicas que pertencem a uma classe de um dado contexto.

O funcionamento do sistema se dará da seguinte maneira:

O usuário informa explicitamente o contexto da atividade a ser realizada dentre 5 opções de atividades: Trabalhar, Correr, Comer, Caminhar, Relaxar. Após isso, dentre uma lista músicas predeterminadas, ele informa um mapeamento entre a música e a classe do contexto realizado. Ao iniciar a execução da sua atividade, o recomendador faz uma série de recomendações avaliando o perfil do usuário, e antes de efetivar a recomendação realiza uma pós-filtragem contextual de acordo com a classe da atividade em execução.

A Figura 10 mostra o sistema de recomendação em funcionamento solicitando ao usuário ativo selecionar um contexto atual e re-

tornando uma lista de músicas produzidas no processo de recomendação :

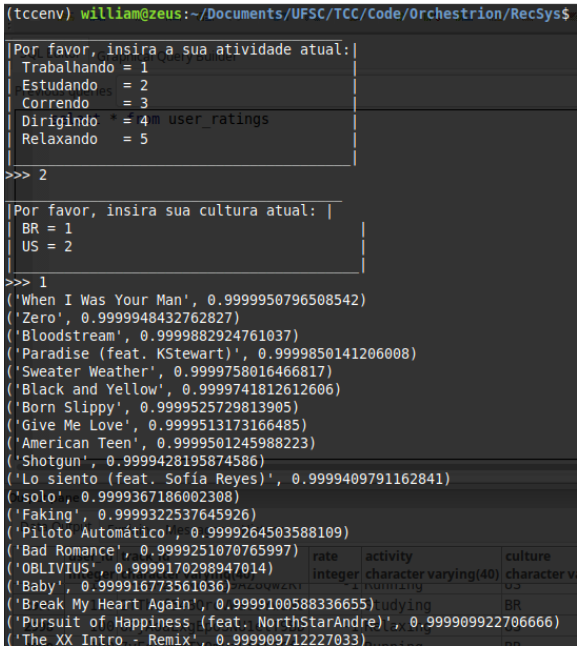


Figura 10 – Sistema de Recomendação em funcionamento Fonte: Elaboração própria

5.1 PERFIL DO USUÁRIO

Na Figura 11 é representado o usuário ativo e seu papel na interação com o sistema, como agente principal que realiza a solicitação de recomendações para um contexto informado por ele e faz avaliações para a lista de músicas produzidas pelo processo de recomendação.

Um usuário dentro do sistema é modelado por meio de uma representação do seu perfil de usuário, que consiste no registro do seu histórico de transações e suas avaliações em um sistema binário, ou seja, positivas ou negativas(1 ou 0 respectivamente), para cada música escutada e uma anotação de contexto. A Figura 12 demonstra, de

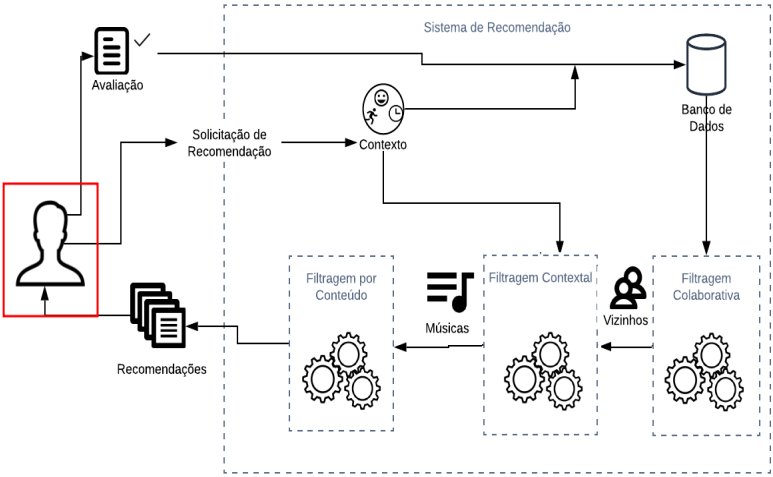
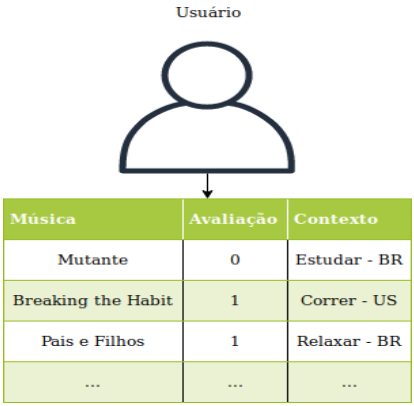


Figura 11 – Módulo do sistema referente ao perfil do usuário (Fonte: Elaboração própria)

maneira mais gráfica, um exemplo de como um perfil de usuário é modelado.



Música	Avaliação	Contexto
Mutante	0	Estudar - BR
Breaking the Habit	1	Correr - US
Pais e Filhos	1	Relaxar - BR
...	...	...

Figura 12 – Modelo do Perfil de Usuário (Fonte: Elaboração própria)

5.2 CONTEXTO

Na Figura 13 é representado o contexto e seu papel na modelagem do sistema, onde é aplicado tanto na anotação de contexto presente nas avaliações dos usuários quanto na incorporação do contexto de maneira explicita, usado para a filtragem contextual.

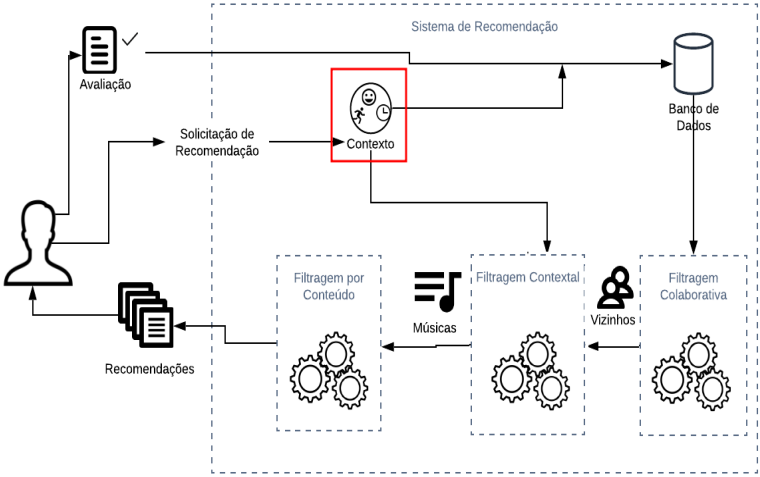


Figura 13 – Módulo do sistema referente ao contexto (Fonte: Elaboração própria)

Para este trabalho, o contexto com o que foi delimitado em duas categorias, a primeira sendo a *Atividade* que o usuário ativo pretende realizar e a segunda a *Cultura* que esse usuário pertence. Estas categorias serão usadas como classes para a classificação das músicas posteriormente.

Os contextos na categoria *Atividade* considerados foram os seguintes:

- Correr
- Trabalhar
- Relaxar

- Dirigir
- Estudar

E para a a categoria *Cultura* foram consideradas as seguintes categorias de acordo com o idioma do usuário, inglês para o primeiro e português para o segundo:

- US
- BR

5.3 BASE DE DADOS

Na Figura 14 podemos observar o módulo do sistema referente à base de dados, por se tratar de uma modelagem baseada em memória, trata-se de um dos módulos mais importantes do sistema. A maneira como essa base de dados funciona e como foi elaborada é descrita mais detalhadamente a seguir.

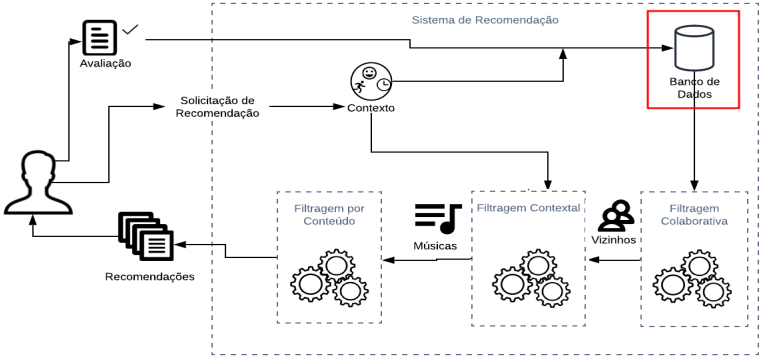


Figura 14 – Módulo do sistema referente à base de dados (Fonte: Elaboração própria)

5.3.1 Pré-Processamento

A Figura 15 ilustra o processo realizado para a criação do conjunto de dados por meio da obtenção de arquivos de áudio, por meio de uma varredura na Spotify API, e sua extração de características.

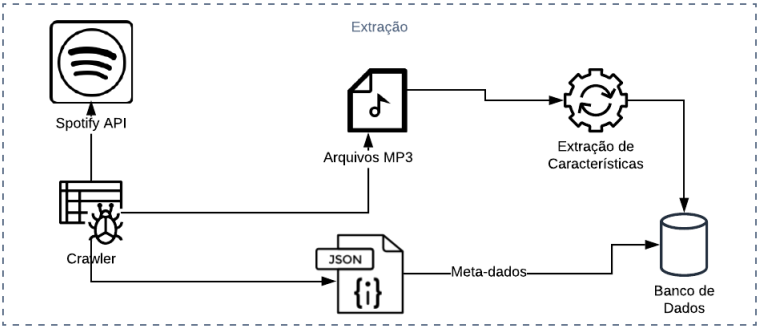


Figura 15 – Processo de obtenção da base de músicas e extração de suas características(Fonte: Elaboração própria)

5.3.1.1 Spotify Web API

Pode se definir API como um conjunto de funções que são disponibilizadas por um software e que podem ser utilizadas inclusive por aplicativos de terceiros. A Web API do Spotify é uma API acessível via Internet que fornece funções de busca musical, por exemplo: busca de música, de artista, de álbum etc. Esta API está disponível em (SPOTIFY, 2019). Existem funções de busca de itens (que podem ser músicas, artistas, álbuns, listas de reprodução...) que recebe como parâmetro o nome do item e o tipo do item. A Figura 16 mostra um exemplo resultados retornados ao executar uma busca de itens do tipo *playlist* com o valor de consulta similar a esse `https://api.spotify.com/v1/playlists/5csGcfoTIPFOeEHagfdlWr/tracks`.

Utilizando esta API, foi possível obter os arquivos mp3 dessas músicas e alguns meta-dados para enriquecer as informações a respeito de cada música.

```
{
  "items": [
    {
      "added_by": {
        "id": "thesoundsofspotify"
      },
      "track": {
        "album": {
          "href": "https://api.spotify.com/v1/albums/7CicGIdLpcRlpX6IgBD4CP",
          "name": "Andre: Durch, …Zu… , …In & …Als… II"
        },
        "href": "https://api.spotify.com/v1/tracks/5087OmhOi9C97y9iUwVWN5",
        "name": "…zu…"
      }
    },
    {
      "added_by": {
        "id": "thesoundsofspotify"
      },
      "track": {
        "album": {
          "href": "https://api.spotify.com/v1/albums/4NLkob6Mk5fOSlgUuCjbh4",
          "name": "Furrer: 3 Klavierstücke, Voicelessness (The Snow Has No Voice) & Phasma"
        },
        "href": "https://api.spotify.com/v1/tracks/724ZTpep4XL2tf44nHFCSN",
        "name": "Voicelessness. The Snow has no Voice"
      }
    }
  ]
}
```


Figura 16 – Exemplo de resposta a uma consulta feita à Spofy web API

5.3.1.2 Crawler

Para este trabalho foram coletadas músicas de diversos gêneros disponíveis para download através da API do Spotify. Para isso foi desenvolvido um *Crawler* que realizou uma varredura para conseguir arquivos de áudio em mp3 e metadados de 2106 músicas que estavam contidas em listas de reprodução. O critério para escolhas destas listas de reprodução foi que o nome delas contivesse palavras ou frases que determinassem os contextos com que as músicas contidas nelas estivessem relacionados (e.g. *correndo à noite*, *músicas para relaxar*, *trabalhando com rock*). Como Pichl, Zangerle e Specht (2015) explica, de fato é possível extrair dados sobre o contexto de músicas observando o nome da lista de reprodução que as contém.

As Figura 17 e 18 mostram a distribuição das músicas por contexto, onde pode se notar um distribuição bastante variada.

Os meta-dados coletados a través da varredura foram utilizados para agregar as características disponibilizadas pela Spotify Web API ao perfil de cada item, além das que seriam extraídas posteriormente.

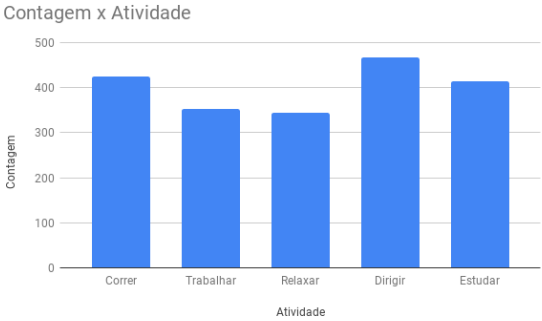


Figura 17 – Distribuição da contagem de músicas coletadas por atividade (Fonte: Elaboração própria)

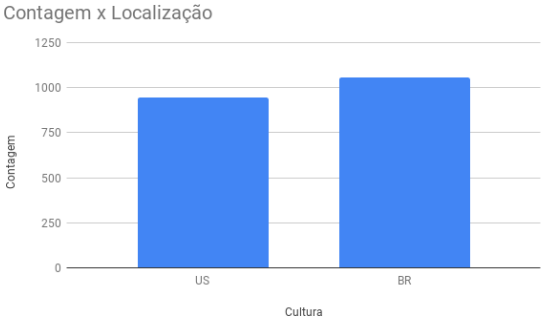


Figura 18 – Distribuição da contagem de músicas coletadas por cultura (Fonte: Elaboração própria)

5.3.2 Extração de Características

Uma vez que se tem uma base com arquivos de áudio torna-se possível a extração de características para cada música e para isso foi utilizada a biblioteca *Essentia*. A politica definida para definir quais características extrair de cada música foi a de coletar a maior quantidade de características que fosse possível para enriquecer a base de dados e posteriormente obter uma vantagem ao se fazer recomendações.

5.3.2.1 Biblioteca Essentia

Essentia é uma biblioteca *open-source* para C++ e Python focada em análise de áudio e recuperação de informação musical baseada em áudio. Contém uma extensa coleção de algoritmos que implementam funcionalidades de entrada/saída, processamento de sinais digitais, caracterização estatística de dados e um grande conjunto de descritores musicais nos âmbitos espectral, temporal, tonal e de alto nível.(ESSENTIA, 2019)

A Tabela 3 mostra os parâmetros de configuração que foram usados na extração das músicas pelo Essentia:

Tabela 3 – Parâmetros de configuração do Essentia. Fonte: Elaboração Própria

Parâmetro	Valor
Tamanho do Frame	30 segs
Janela	Hann
Taxa de amostragem	44100Hz

O parâmetro de tamanho de frame foi configurado usando um valor de 30 segundos, por se tratar do tamanho das amostras de áudio. Já o parâmetro de tipo de janela foi configurado para usar o valor de Hann dado que esse tipo de janela evita o problema de serrilhamento no sinal. A taxa de amostragem escolhida foi a de 44100Hz para não se ter nenhum tipo de perda na informação.

5.3.2.2 Características

A Tabela 4 mostra o conjunto de características escolhidas para a construção do conjunto de dados:

Tabela 4 – Conjunto de características extraídas. Fonte: Elaboração Própria

Característica	Descrição	Motivação
MaxMagFreq	Calcula a frequência com maior magnitude em um espectro.	Tem relação com a altura tonal em um som.
RollOff	Calcula a frequência de roll-off em um espectro	Usado para distinguir sons harmônicos e não harmônicos
Spectral Centroid	Calcula a Centroide Espectral de um sinal	Tem relação com a frequência dominante de um som.
BPM	Calcula o ritmo em BPM em um sinal	Tem relação com o tempo da música
Danceability	Estima numericamente que tão dançável é um sinal de áudio	Interessante para o usuário segundo o contexto
Energy	Calcula a energia em um sinal	Interessante para o usuário segundo o contexto
Power	Calcula o poder instantâneo apresentado por uma amostra	Interessante para o usuário segundo o contexto
Loudness	Calcula o volume apresentado por um sinal	Interessante para o usuário segundo o contexto
Instrumentalness	Calcula a ausência de Vocais em um sinal	Interessante para o usuário segundo o contexto
Speechiness	Calcula a presença de palavras faladas no sinal.	Interessante para o usuário segundo o contexto
Liveness	Calcula a presença de audiência no sinal da gravação	Interessante para o usuário segundo o contexto
Valence	Calcula a “positividade” no sinal de uma música. i.e. Alegria, Felicidade, Euforia.	Interessante para o usuário segundo o contexto

5.3.3 Conjunto de Dados

Para poder trabalhar com um sistema de recomendação como o proposto neste trabalho precisa-se de dados de usuários e histórico de avaliações destes, como esse não é um dado disponível via Spotify API, foram gerados 100 usuários junto com 30 avaliações positivas ou negativas, um sistema binário, para cada um deles a qual contém uma anotação de contexto de acordo com a classificação de contexto da música. Esta classificação de contexto da música é usada neste momento para obter uma simulações de iteração do usuário com o sistema de maneira mais realista e posteriormente para avaliar a recomendação fornecida pelo sistema. Essa quantidade de avaliações por usuário é usada para evitar um problema de arranque a frio, como descrito anteriormente. Desta maneira, foi gerado um histórico de 3000 iterações de todos os usuários que forma o conjunto de dados a ser empregado. As Figura 19 e 20 mostram a distribuição das avaliações por atividade e também por cultura, percebe-se que se trata de uma distribuição bem diversificada porém não uniforme.

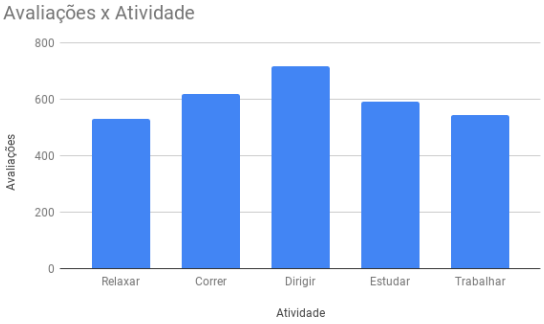


Figura 19 – Distribuição da contagem de avaliações por atividade (Fonte: Elaboração própria)

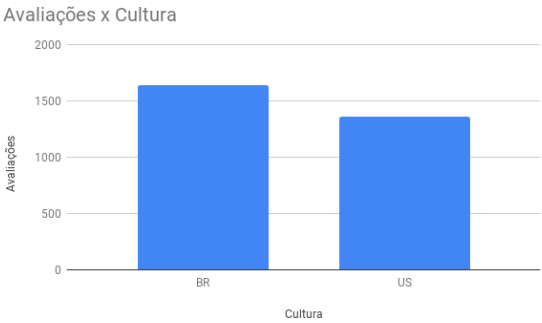


Figura 20 – Distribuição da contagem de avaliações por cultura (Fonte: Elaboração própria)

5.4 FILTRAGEM COLABORATIVA

Na Figura 21 é representado o módulo de filtragem colaborativa, que tem como objetivo encontrar os usuários com gostos similares para expandir o espaço dos itens levados em consideração na busca por potenciais recomendações.

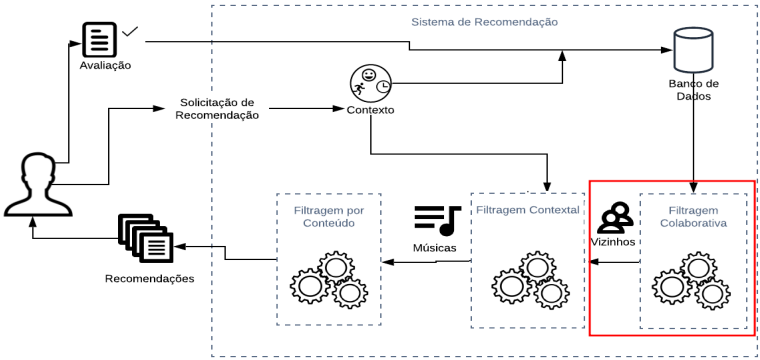


Figura 21 – Módulo do sistema referente à filtragem colaborativa (Fonte: Elaboração própria)

Para realizar a filtragem colaborativa, precisamos levar em consideração não só as avaliações anteriores do usuário para atingir recomendações personalizadas, mas também descobrir quais outros usuários existem com gostos similares, pois poderiam ter ouvido músicas que seriam de interesse para o usuário ativo. De acordo com a literatura, existem diversos algoritmos que podem ser utilizados para a FC como é o caso de K-Vizinhos Mais Próximos(KNN), Máquinas de Vetores de Suporte (SVM), Árvores de Decisão ou K-Médias, entre outras. Neste módulo do sistema foi utilizada uma técnica bastante tradicional na FC, o algoritmo K-Vizinhos Mais Próximos (KNN) apresentado no Capítulo 3, a ideia por trás disso é achar similaridades nas avaliações anteriores com outros usuários e ao escolher esses usuários, obter uma lista com as possíveis músicas para recomendar, isto é, usar as avaliações positivas desses usuários selecionados. A motivação principal para ter escolhido esta técnica é a de que se trata de um algoritmo *lazy*, logo o sistema pode se adaptar a mudanças rápidas na matriz de avaliações de usuário.

5.5 FILTRAGEM CONTEXTUAL

Na Figura 22 é representado o módulo de filtragem contextual, que tem por objetivo encontrar os usuários similares que já estiveram em situações contextuais semelhantes para assim poder realizar uma busca melhor por músicas relacionadas com o contexto do usuário ativo.

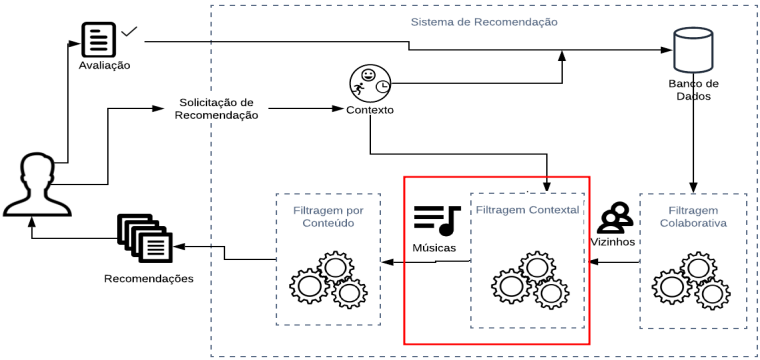


Figura 22 – Módulo do sistema referente à filtragem contextual (Fonte: Elaboração própria)

Uma vez realizada a FC, obtendo os usuários mais próximos ao usuário ativo, é realizada mais uma filtragem, dessa vez reduzindo ainda mais o escopo desses usuários próximos para obter somente os que avaliaram músicas em contextos similares com o apresentado pelo próprio usuário ativo. Para fazer essa seleção, surge a necessidade de se calcular a similaridade entre dois contextos. A técnica escolhida para fazer esse cálculo foi Coeficiente de Semelhança Simples, uma vez que o contexto atrelado à avaliação do usuário em relação a uma música é um dado qualitativo ao invés de quantitativo, ao comparar dois deles, apenas podemos afirmar se são iguais ou diferentes em cada categoria de contexto. Este calculo é aplicado para todo usuário da lista de vizinhos que a etapa de FC produziu, os usuários que na média, tiverem mais avaliações com contexto mais similar ao contexto apresentado pelo usuário ativo são escolhidos.

5.6 FILTRAGEM BASEADA EM CONTEÚDO

Na Figura 23 é representado o módulo de filtragem baseada em conteúdo, que tem por objetivo encontrar as músicas semelhantes às já avaliadas pelo usuário para assim poder realizar uma recomendação mais personalizada. Na última etapa antes de realizar a recomendação

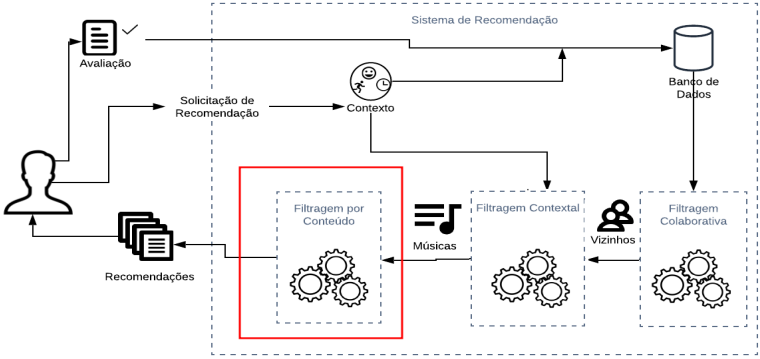


Figura 23 – Módulo do sistema referente à filtragem baseada em conteúdo (Fonte: Elaboração própria)

de fato, é realizada mais uma filtragem. Nesse ponto, existe uma lista com os usuários mais similares de acordo com o contexto. Assim, temos

acesso às músicas que esses usuários tem maior afinidade em um contexto similar, logo, desejamos selecionar quais dessas músicas são mais parecidas com as músicas já avaliadas anteriormente pelo usuário de maneira positiva, em outras palavras, gostaríamos de descobrir quais que tem maior potencial de ser bem avaliadas por serem similares às avaliadas positivamente pelo usuário ativo e por tanto, poderiam ser melhores candidatos a recomendações bem sucedidas.

Para solucionar este problema, devemos aplicar uma técnica que nos permita calcular a similaridade entre duas músicas. Durante a construção do conjunto de dados, diversas características de cada música foram extraídas e agregadas, esta série de características pode ser facilmente interpretada como um vetor. Logo, precisa-se de uma medida onde podemos calcular o quão similares dois vetores são, é o caso da *similaridade cosseno* que foi definida anteriormente no Capitulo 3, uma vantagem em relação à *correlação de pearson* é que este último tende a mostrar um desempenho inferior quando os usuários tendem a ter menos itens em comum.

Após aplicar essa técnica comparando todas as músicas do usuário ativo com aquelas escutadas e avaliadas pelos usuários que tem a maior semelhança com o usuário ativo e seu contexto, e posteriormente ordenando por relevância, obtemos uma lista com as n músicas a serem recomendadas ao usuário. Esta lista, se encontra ordenada do item mais relevante, para o item menos relevante.

6 AVALIAÇÃO DE RESULTADOS

Para avaliar o SR desenvolvido, foram criados alguns experimentos descritos e avaliados nas seções a seguir.

6.1 DESCRIÇÃO DOS EXPERIMENTOS

Por questões de limitações de recursos, uma avaliação On-line ou mesmo uma avaliação de Testes com usuários não foram factíveis, logo faz-se necessário o uso de uma avaliação Off-line. Como mencionado na literatura respectiva, ao se realizar uma avaliação Off-line, considera-se que os dados modelados irão se comportar de maneira similar aos dos usuários interagindo com o sistema. Para avaliar alguns parâmetros serão fixados e alguns serão variados para poder obter resultados que nos permitam fazer comparações e assim obter conclusões. Para os testes a serem realizados, foram escolhidos 10(dez) usuários a esmo e um contexto em comum que manterão com valor fixo de 'Estudar' para atividade e 'BR' para cultura, com intuito de garantir que a entrada seja sempre a mesma para o contexto e assim isolar variáveis que afetam as mudanças no resultado da recomendação. As variáveis propostas para os experimentos são:

K serão feitos testes com diferentes valores de K para o algoritmo KNN, para avaliar os possíveis efeitos do tamanho da lista de vizinhos similares, recuperada dentre todos os usuários, na recomendação final, os valores assumidos serão $k = \{5, 10, 15, 20\}$.

Contexto representa o valor que o filtro de contexto pode assumir, este nível indica como serão feitos os testes, no caso assumindo valores do conjunto $Contexto = \{Com\ Contexto, Sem\ Contexto\}$. A ideia por trás do valor *Sem Contexto* é avaliar se realmente a filtragem contextual consegue trazer benefícios para a recomendação.

Usuário serão feitos testes com dez usuários diferentes para poder ter dez cenários diferentes para avaliar a recomendação usando o mesmo contexto.

Como sabemos a classificação de cada música, advinda da criação do conjunto de dados, podemos tomar vantagem disso e validar o modelo de recomendação por meio da métricas *MAP* e *precisão em n*, avaliando a lista de músicas recomendadas para o usuário e contexto

fixados para teste, poderemos avaliar se o contexto em que a música foi classificada na sua extração é o mesmo que o contexto do teste, isso configuraria uma recomendação acertada, pois significaria que o recomendador chegou numa conclusão de classificação similar para essa música que o humano que criou a lista de reprodução com essa temática.

6.2 RESULTADOS DOS EXPERIMENTOS

Foram realizados em torno de 240 testes diferentes para compilar os dados dos experimentos. Estes dados foram coletados verificando a quantidade de VP, VN, FP e FN de acordo com as métricas *Precisão em N* para formar matrizes de confusão e *MAP* para criar um gráfico comparativo, indicando assim a taxa de acertos do recomendador. Nos gráficos a seguir, temos a comparação de matrizes de confusão para um usuário em específico sobre os quais se realizou testes e foi selecionado a esmo para esta análise, comparando em cada uma delas, diferentes valores para K do KNN, e aplicação de filtragem contextual.

(A)		Classe Predita	
		Positivo	Negativo
Classe Observada	Positivo	5,00	200,00
	Negativo	5,00	1896,00

(B)		Classe Predita	
		Positivo	Negativo
Classe Observada	Positivo	3,64	200,00
	Negativo	6,36	1896,00

Figura 24 – Matrizes de confusão para k = 5 com uso de contexto em (A) e sem uso de contexto em (B). Fonte: Elaboração própria

(A)		Classe Predita	
		Positivo	Negativo
Classe Observada	Positivo	5,00	200,00
	Negativo	5,00	1896,00

(B)		Classe Predita	
		Positivo	Negativo
Classe Observada	Positivo	0,10	200,00
	Negativo	9,90	1896,00

Figura 25 – Matrizes de confusão para k = 10 com uso de contexto em (A) e sem uso de contexto em (B). Fonte: Elaboração própria

(A)		Classe Predita	
		Positivo	Negativo
Classe Observada	Positivo	5,00	200,00
	Negativo	5,00	1896,00

(B)		Classe Predita	
		Positivo	Negativo
Classe Observada	Positivo	3,18	200,00
	Negativo	6,82	1896,00

Figura 26 – Matrizes de confusão para k = 15 com uso de contexto em (A) e sem uso de contexto em (B). Fonte: Elaboração própria

(A)

		Classe Predita	
		Positivo	Negativo
Classe Observada	Positivo	3,64	200,00
	Negativo	6,36	1896,00

(B)

		Classe Predita	
		Positivo	Negativo
Classe Observada	Positivo	4,55	200,00
	Negativo	5,45	1896,00

Figura 27 – Matrizes de confusão para k = 20 com uso de contexto em (A) e sem uso de contexto em (B). Fonte: Elaboração própria

Avaliando as Figuras 24,25,26 e 27 percebe-se que abordagens contextualizadas aparentam obter recomendações melhores que as descontextualizadas ao compararmos a taxa de verdadeiros positivos. Outro ponto a ser avaliado é que o valor de K não parece ser um valor muito influente para este modelo de recomendação, pois não parecem existir grandes alterações com as mudanças deste parâmetro. Isto pode ser causado pelas ordem entre os filtros, pois ao se ordenar sempre pelos usuários próximos mais similares ao usuário ativo, não é provável que seja achado entre usuários mais distantes, e por tanto menos similares, candidatos melhores na FC. A Figura 27 chama à atenção, pois com um valor K maior, a filtragem contextualizada perde precisão e a não contextualizada melhora sua performance. Uma hipótese para este comportamento, é a de que um usuário pode ser muito similar a outro, porém em contextos diferentes(i.e., dois usuários onde ambos gostam de correr escutando rock, porém, ao estudar, um prefere hip-hop e o outro prefere música clássica) e quando é ampliado o espaço de usuários, a busca por candidatos mais similares fica mais semelhante a uma busca exaustiva, essa abordagem é avaliada como ruim, pois para um conjunto de dados grande, fazer uma busca exaustiva demoraria um tempo impraticável e a essência por trás de um sistema de recomendação se perde, o ideal é que o SR consiga por seus próprios meios ajustar e melhorar o espaço inicial de busca.

Em seguida, na Figura 28, temos o comparativo para MAP entre todos os cenários com diferentes usuários.

O primeiro detalhe que chama a atenção avaliando os MAP para todos os usuários é que no geral, o recomendador apresenta uma taxa aceitável de recomendações corretas, o único momento que esta taxa se apresentou inferior a 20% foi no teste realizado com o Usuário 7 sem aplicação de filtragem contextual. Ressaltando que o MAP é a média das médias de Precisão, percebemos que é um bom resultado para um recomendador simples como este. Um fator que chama à atenção é o limite de precisão de 50%, isto pode ser causado por alguma relação matemática entre a quantidade fixa de avaliações positivas e negativas dos usuários, dada a maneira como foi gerada essa base de usuários. Um

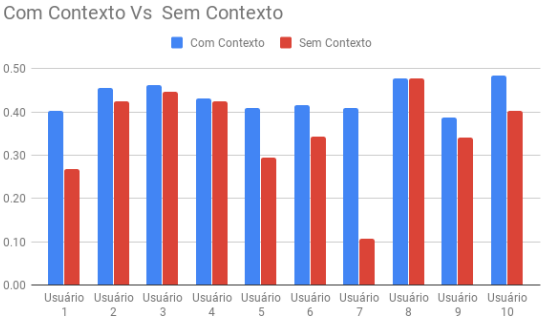


Figura 28 – Gráfico de MAP para todos os Usuários. Fonte: Elaboração própria

outro fator que poderia ser investigado posteriormente, em decorrência disto, seria se essas taxas de acerto não poderiam ser maiores um tamanho maior de conjunto de dados. Outro ponto que deve ser discutido é que não necessariamente a recomendação se beneficia da filtragem contextual se os dados disponíveis não forem de utilidade para melhorar o filtro. Em relação a isso, pode se perceber que na maioria dos cenários a recomendação contextual apresentou melhorias em relação a uma recomendação sem contexto, porém é possível observar que em alguns casos parece não apresentar melhorias, como é o caso do cenário do Usuário 8 onde a precisão se manteve igual em ambos casos.

7 CONCLUSÃO

Embora muitas melhorias possam ser realizadas ainda nesse SR, este trabalho conseguiu atingir seu objetivo de desenvolver e avaliar um Sistema de Recomendação Musical Sensível ao Contexto e ainda expor alguns problemas detectados, inclusive propondo abordagens para solucionar estes. O estudo em sistemas de recomendação no capítulo 3 e o estudo das SRs sensíveis ao contexto apresentados no capítulo 4 foram a base para o desenvolvimento deste trabalho. Com ajuda destes conhecimentos, foi possível criar uma base de dados com anotações em contexto, que ajudou a modelar o sistema, e a desenvolver uma solução capaz de realizar recomendações e ainda testá-las. Com os resultados dos testes apresentados, a pesar de serem bastante simples, foi possível avaliar que a recomendação musical pode de fato se beneficiar da aplicação de contexto para seu calculo. No entanto, cabe ressaltar que as capacidades do SR proposto são bastante limitadas por alguns fatores, como o fato do conjunto de dados ser bastante pequena em relação a um conjunto usado em um cenário real. Isso impede o recomendador de inferir maiores peculiaridades sobre o gosto do usuário em relação à música em certos contexto. A resolução deste problema é por meio da obtenção continua de informações tanto sobre músicas quanto sobre os usuários, porém pode ser algo bastante custoso, tanto em termos computacionais, quanto financeiros. A obtenção de contexto também pode ser melhorada em muitos aspectos desde a maneira em como é obtida até a quantidade de categorias a serem incluídas. Seguem algumas sugestões de melhorias a serem feitas e de trabalhos futuros.

7.1 TRABALHOS FUTUROS

Em seguida estão reunidas algumas melhorias e continuidade nos estudos relativos ao sistema de recomendação apresentado.

- Estudar e implementar uma solução que possa extrair e inferir o contexto do usuário ativo no momento.
- Realizar novos experimentos com novas variáveis e maior quantidade de músicas envolvidas.
- Desenvolver um aplicativo que possa disponibilizar o sistema de recomendação para testes reais com usuários.

- Implementar algoritmos no módulo de filtragem baseada em conteúdo que consigam inferir a relação direta entre as características das músicas e as classes de contextos diferentes.
- Implementar novos algoritmos para realizar recomendações alternativos para combater problemas como arranque a frio ou sobre-especialização.
- Levantar novos requisitos que possam vir a ser incluídos no sistema para aumentar sua aplicabilidade em um cenário real

REFERÊNCIAS

ADOMAVICIUS, G.; TUZHILIN, A. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. **IEEE transactions on knowledge and data engineering**, IEEE, v. 17, n. 6, p. 734–749, 2005.

ADOMAVICIUS, G.; TUZHILIN, A. Context-aware recommender systems. In: **Recommender systems handbook**. [S.l.]: Springer, 2011. p. 217–253.

AL-BASHIRI, H. et al. An improved memory-based collaborative filtering method based on the topsis technique. **PloS one**, Public Library of Science, v. 13, n. 10, p. e0204434, 2018.

BALTRUNAS, L. Exploiting contextual information in recommender systems. In: ACM. **Proceedings of the 2008 ACM conference on Recommender systems**. [S.l.], 2008. p. 295–298.

BOBADILLA, J. et al. Recommender systems survey. **Knowledge-based systems**, Elsevier, v. 46, p. 109–132, 2013.

BOSI, M.; GOLDBERG, R. E. **Introduction to digital audio coding and standards**. [S.l.]: Springer Science & Business Media, 2012.

BURKE, R. Hybrid recommender systems: Survey and experiments. **User modeling and user-adapted interaction**, Springer, v. 12, n. 4, p. 331–370, 2002.

CARPENTIER, G.; BARTHÉLÉMY, J. The interactive-music network. 2005.

CELMA, O. Music recommendation. In: **Music Recommendation and Discovery**. [S.l.]: Springer, 2010. p. 43–85.

CRASWELL, N. Precision at n. **Encyclopedia of database systems**, Springer, p. 2127–2128, 2009.

CUNNINGHAM, P. Unsupervised learning and clustering.

DEY, A. K. Understanding and using context. **Personal and ubiquitous computing**, Springer-Verlag, v. 5, n. 1, p. 4–7, 2001.

DEY, A. K.; ABOWD, G. D. Providing architectural support for building context-aware applications. College of Computing, Georgia Institute of Technology, 2000.

DIAS, R.; FONSECA, M. J.; CUNHA, R. A user-centered music recommendation approach for daily activities. In: **CBRecSys@ RecSys**. [S.l.: s.n.], 2014. p. 26–33.

DOMINGOS, P. A few useful things to know about machine learning. **Communications of the ACM**, ACM, v. 55, n. 10, p. 78–87, 2012.

DUAN, S. et al. A survey of tagging techniques for music, speech and environmental sound. **Artificial Intelligence Review**, v. 42, n. 4, p. 637–661, Dec 2014.

ESSENTIA. **Essentia lib**. 2019. `https://essentia.upf.edu/documentation/documentation.html`. Accessed: 2019-03-25.

GOOD, M. et al. Musicxml: An internet-friendly format for sheet music. In: **XML Conference and Expo**. [S.l.: s.n.], 2001. p. 03–04.

HERRADA, Ò. C. Music recommendation and discovery in the long tail. Universitat Pompeu Fabra, 2009.

IFPI. **IFPI Global Music Report 2019**. 2019. `https://www.ifpi.org/news/IFPI-GLOBAL-MUSIC-REPORT-2019`. Accessed: 2019-04-30.

JAYASHREE, D.; MANIAN, S. G.; SRIVATSAV, C. P. Music recommendation system. **Asian Journal of Information Technology**, v. 15, n. 21, p. 4250–4254, 2016.

KOREN, Y.; BELL, R.; VOLINSKY, C. Matrix factorization techniques for recommender systems. **Computer**, IEEE, n. 8, p. 30–37, 2009.

KUNAVER, M.; POŽRL, T. Diversity in recommender systems–a survey. **Knowledge-Based Systems**, Elsevier, v. 123, p. 154–162, 2017.

LEE, J. H.; SHIN, J.; REALFF, M. J. Machine learning: Overview of the recent progresses and implications for the process systems engineering field. **Computers & Chemical Engineering**, Elsevier, 2017.

MCNAB, R. J. et al. Towards the digital music library: Tune retrieval from acoustic input. In: ACM. **Proceedings of the first ACM international conference on Digital libraries**. [S.l.], 1996. p. 11–18.

MICHALSKI, R. S.; CARBONELL, J. G.; MITCHELL, T. M. **Machine learning: An artificial intelligence approach**. [S.l.]: Springer Science & Business Media, 2013.

MILETTO, E. M. et al. Introdução à computação musical. In: **IV Congresso Brasileiro de Computação**. [S.l.: s.n.], 2004.

MONARD, M. C.; BARANAUSKAS, J. A. Conceitos sobre aprendizado de máquina. **Sistemas Inteligentes-Fundamentos e Aplicações**, v. 1, n. 1, 2003.

MOORER, J. A. Signal processing aspects of computer music: A survey. **Proceedings of the IEEE**, IEEE, v. 65, n. 8, p. 1108–1137, 1977.

ORIO, N. et al. Music retrieval: A tutorial and review. **Foundations and Trends® in Information Retrieval**, Now Publishers, Inc., v. 1, n. 1, p. 1–90, 2006.

PANNIELLO, U. et al. Experimental comparison of pre-vs. post-filtering approaches in context-aware recommender systems. In: ACM. **Proceedings of the third ACM conference on Recommender systems**. [S.l.], 2009. p. 265–268.

PARK, D. H. et al. A literature review and classification of recommender systems research. **Expert Systems with Applications**, Elsevier, v. 39, n. 11, p. 10059–10072, 2012.

PICHL, M.; ZANGERLE, E.; SPECHT, G. Towards a context-aware music recommendation approach: What is hidden in the playlist name? In: IEEE. **2015 IEEE International Conference on Data Mining Workshop (ICDMW)**. [S.l.], 2015. p. 1360–1365.

PRIMO, T. T. Método de representação de conhecimento baseado em ontologias para apoiar sistemas de recomendação educacionais. 2013.

RÄTSCH, G. A brief introduction into machine learning. **Friedrich Miescher Laboratory of the Max Planck Society**, 2004.

REDDY, S.; MASCIA, J. Lifetrak: music in tune with your life. In: ACM. **Proceedings of the 1st ACM international workshop on Human-centered multimedia**. [S.l.], 2006. p. 25–34.

REZENDE, S. O. **Sistemas inteligentes: fundamentos e aplicações**. [S.l.]: Editora Manole Ltda, 2003.

RICCI, F.; ROKACH, L.; SHAPIRA, B. **Recommender systems handbook**. [S.l.]: Springer, 2011.

RICE, S. V. A survey course on computer audio. **Journal of Computing Sciences in Colleges**, Consortium for Computing Sciences in Colleges, v. 20, n. 6, p. 118–124, 2005.

SHANI, G.; GUNAWARDANA, A. Evaluating recommendation systems. In: **Recommender systems handbook**. [S.l.]: Springer, 2011. p. 257–297.

SHAO, B. et al. Music recommendation based on acoustic features and user access patterns. **IEEE Transactions on Audio, Speech, and Language Processing**, IEEE, v. 17, n. 8, p. 1602–1611, 2009.

SILVA, R. B.; BERNARDINI, F. C. Move! um sistema de recomendação de músicas para uma atividade física. In: SBC. **Anais da I Escola Regional de Sistemas de Informação do Rio de Janeiro**. [S.l.], 2014. p. 33–40.

SØDRING, T. **Content-based retrieval of digital music**. Tese (Doutorado) — Dublin City University, 2002.

SPOTIFY. **Spotify API**. 2019. `https://developer.spotify.com/web-api/`. Accessed: 2019-04-30.

STECK, H. Evaluation of recommendations: rating-prediction and ranking. In: ACM. **Proceedings of the 7th ACM conference on Recommender systems**. [S.l.], 2013. p. 213–220.

STEEG, F. v. **Context-aware recommender systems**. Dissertação (Mestrado), 2015.

TAN, P.-N. **Introduction to data mining**. [S.l.]: Pearson Education India, 2018.

TIMES, N. Y. **Spotify Reaches 100 Million Subscribers, but Not Without Some Dissonance**. 2019. `https://www.nytimes.com/2019/04/29/business/media/`

spotify-100-million-subscribers-apple-podcasts.html. Accessed: 2019-04-30.

WATKINSON, J. **The art of digital audio**. [S.l.]: Taylor & Francis, 2001.

ANEXO A – Código do Sistema de Recomendação

```
#json_reader.py

import psycopg2
import sys
import os
import json
from pprint import pprint


connstring = "host='localhost' dbname='musicdb' user='willi

def insert_jsons():
    rootdir = './json/'
    con = None

    try:
        id = 0
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        for subdir, dirs, files in os.walk(rootdir):
            for file in files:
                with open(rootdir + file) as json_file:
                    data = json.load(json_file)
                    print("tamanho: "+ str(len(data['items
                    for i in data['items']:
                        id=id+1
                        track_name = i['track']['name']
                        track_url = i['track']['href'].spl
                        track_id = track_url[len(track_url)

                        track_album_name = str(i['track'][

                        track_album_url = i['track']['albun
                        track_album_id = track_album_url[le

                        cur.execute("INSERT INTO Music VAL

        con.commit()
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()
```

```
        print('Error %s', e)
        sys.exit(1)

    finally:
        if con:
            con.close()


def create_table():

        con = None

        try:
            con = psycopg2.connect(connstring)
            cur = con.cursor()
            cur.execute("select exists(select relname from pg_c
            exists = bool(cur.fetchone()[0])
            if not exists:
                cur.execute("CREATE TABLE Music(Id track_name V
            con.commit()
        except psycopg2.DatabaseError as e:
            if con:
                con.rollback()

            print('Error %s', e)
            sys.exit(1)

        finally:
            if con:
                con.close()


def main():
    create_table()
    insert_jsons()
if __name__ == "__main__":
    main()
```

#spotify_crawler.py

```
import json
import requests
import os
import shutil
import psycopg2
import spotipy
from spotipy.oauth2 import SpotifyClientCredentials
from pprint import pprint

CLIENT_ID = '2bce4433c4334c479ba951e9372167d3'
CLIENT_SECRET = 'ebb81857968142149e62b2ebcabea8fd'
#CLIENT_ID = '68c35d58f44948488ab7b4ce1c491db2'
#CLIENT_SECRET = '0e311992c3b04a46910329463e29d898'
connstring = "host='localhost' dbname='musicdb' user='willi

downloading = 'dirigir'

def main():
    #download_tracks()
    download_features()

def download_features():
    con = None
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()

        cur.execute("SELECT exists(SELECT column_name FROM
        exists = bool(cur.fetchone()[0])

        if not exists:
            cur.execute("ALTER TABLE music ADD COLUMN f_key
            cur.execute("ALTER TABLE music ADD COLUMN f_mod
            cur.execute("ALTER TABLE music ADD COLUMN f_dan
            cur.execute("ALTER TABLE music ADD COLUMN f_ene
            cur.execute("ALTER TABLE music ADD COLUMN f_lou
            cur.execute("ALTER TABLE music ADD COLUMN f_spe
            cur.execute("ALTER TABLE music ADD COLUMN f_aco
            cur.execute("ALTER TABLE music ADD COLUMN f_ins
            cur.execute("ALTER TABLE music ADD COLUMN f_liv
            cur.execute("ALTER TABLE music ADD COLUMN f_val
```

```
            cur.execute("ALTER TABLE music ADD COLUMN f_tempo r

        cur.execute("SELECT track_id FROM music where track_gen
        client_credentials_manager = SpotifyClientCredentials(c
        sp = spotipy.Spotify(client_credentials_manager=client_c
        tracks = cur.fetchall()
        for track in tracks:
            if track is None:
                print("Inexistent Track: ", track_id)
                continue
            track_id = track[0]
            urn = 'spotify:track:'+ track_id
            track_features = sp.audio_features(urn)
            key = track_features[0]['key']
            mode = track_features[0]['mode']
            danceability = track_features[0]['danceability']
            energy = track_features[0]['energy']
            loudness = track_features[0]['loudness']
            speechiness = track_features[0]['speechiness']
            acousticness = track_features[0]['acousticness']
            instrumentalness = track_features[0]['instrumentaln
            liveness = track_features[0]['liveness']
            valence = track_features[0]['valence']
            tempo = track_features[0]['tempo']
            cur.execute("UPDATE music SET f_key = %s, f_mode = %
        con.commit()
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()

        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()


def download_tracks():
    con = None
    try:
```

```
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        cur.execute("SELECT track_id, track_genre FROM mus
        for track in cur:
            track_id = track[0]
            track_genre = track[1]
            download_track(track_id, track_genre)
        con.commit()
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()

        print('Error %s', e)
        sys.exit(1)

    finally:
        if con:
            con.close()



def download_track(track_id, track_genre):
    client_credentials_manager = SpotifyClientCredentials(
    sp = spotipy.Spotify(client_credentials_manager=client_

    urn = 'spotify:track:'+ track_id
    track = sp.track(urn)
    track_url = track['preview_url']
    if track_url is None:
        print("'" +track_id+ "',")
        return
    dump_directory = os.path.join(os.getcwd(), 'mp3/'+track
    if not os.path.exists(dump_directory):
        os.makedirs(dump_directory)
    r = requests.get(track_url)
    if r.status_code == 200:
        with open('./mp3/'+track_genre+'/'+track_id+'.mp3'
            r.raw.decode_content = True
            for chunk in r:
                f.write(chunk)
```

```
if __name__ == main():
    main()


#extrator.py

import sys
import os
import shutil
import psycopg2
import essentia
import essentia.standard as std
from essentia.standard import *

connstring = "host='localhost' dbname='musicdb' user='william' 

def create_columns(cur, con):

    cur.execute("SELECT exists(SELECT column_name FROM informat
    exists = bool(cur.fetchone()[0])
    if not exists:
        cur.execute("ALTER TABLE music ADD COLUMN f_max_mag_fre
        cur.execute("ALTER TABLE music ADD COLUMN f_roll_off re
        cur.execute("ALTER TABLE music ADD COLUMN f_spectral_ce
        cur.execute("ALTER TABLE music ADD COLUMN f_bpm real")
        cur.execute("ALTER TABLE music ADD COLUMN f_danceabilit
        cur.execute("ALTER TABLE music ADD COLUMN f_energy_es r
        cur.execute("ALTER TABLE music ADD COLUMN f_power real"
        cur.execute("ALTER TABLE music ADD COLUMN f_loudness_es

        con.commit()



def main():
    con = None
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        create_columns(cur,con)

        rootdir = '/home/william/Documents/UFSC/TCC/Code/Orches
```

```
for subdir, dirs, files in os.walk(rootdir):
        for file in files:
            track_id = os.path.splitext(file)[0]
            full_path = (subdir+'/'+file)
            loader = std.MonoLoader(filename=full_p
            audio = loader()

            w = Windowing(type = 'hann')
            frame = audio[0*44100 : 30*44100]

            # #MaxMagFreq
            func_maxmagfreq = MaxMagFreq()
            maxmagfreq = func_maxmagfreq(w(frame))

            # #RollOff
            func_rolloff = RollOff()
            rolloff = func_rolloff(w(frame))

            # #Spectral Centroid
            func_spectral_centroid = SpectralCentro
            spectralcentroid = func_spectral_centro

            #Rithm

            #BPM
            func_bpm = RhythmExtractor2013()
            bpm, ticks, confidence, estimates, bpmi

            #Danceability
            func_dance = Danceability()
            danceability, dfa = func_dance(w(frame

            # #Math
            # #Energy
            func_energy = Energy()
            energy = func_energy(w(frame))

            # #power
            func_power = InstantPower()
            power = func_power(w(frame))
```

```
                    # #Loudness
                    func_loudness = Loudness()
                    loudness = func_loudness(w(frame))

                    cur.execute("UPDATE music SET f_max_mag_fre

        con.commit()
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()

        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()


if __name__ == "__main__":
    main()


#Persistance.py

import os
import sys
import shutil
import psycopg2
import Model
from Model import Context as Context
from Model import Rating as Rating

connstring = "host='localhost' dbname='musicdb' user='william' 

def create_users_table():
    con = None
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        if(not check_table_exists('users')):
            cur.execute("CREATE TABLE users ( user_id SERIAL PR
```

```
            con.commit()
        except psycopg2.DatabaseError as e:
            if con:
                con.rollback()

            print('Error ', e)
            sys.exit(1)

        finally:
            if con:
                con.close()


def create_user_ratings_table():
    con = None
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        if(not check_table_exists('user_ratings')):
            cur.execute("CREATE TABLE user_ratings(user_id
        con.commit()
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()

        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()


def check_table_exists(name):
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        cur.execute("select exists(select relname from pg_
        exists = bool(cur.fetchone()[0])
        con.commit()
    except psycopg2.DatabaseError as e:
```

```
        if con:
            con.rollback()

        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()
        return exists


def insert_user(name):
    con = None
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        cur.execute("INSERT INTO users(user_name) VALUES('"+nam
        con.commit()
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()
        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()


def get_all_users():
    con = None
    users = []
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        cur.execute("SELECT * FROM users")
        con.commit()
        for user in cur:
            user_id = user[0]
            user_name = user[1]
```

```
            users.append(Model.User(user_id,user_name))
        return users
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()
        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()


def get_all_user_ids():
    con = None
    user_ids = []
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        cur.execute("SELECT user_id FROM users")
        con.commit()
        for user in cur:
            user_id = user[0]
            user_ids.append(user_id)
        return user_ids
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()
        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()


def get_all_ratings():
    con = None
    ratings = []
    try:
        con = psycopg2.connect(connstring)
```

```
        cur = con.cursor()
        cur.execute("SELECT user_id, track_id, rating FROM user_r
        con.commit()
        for rating in cur:
            row = []
            row.append(rating[0])
            row.append(rating[1])
            row.append(rating[2])
            ratings.append(row)
        return ratings
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()
        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()

def get_all_ratings_user(user_id):
    con = None
    ratings = dict()
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        cur.execute("SELECT * FROM user_ratings where user_id =
        con.commit()
        for rating in cur:
            track_id = rating[1]
            rate = rating[2]
            context_activity = rating[3]
            context_culture = rating[4]
            ratings.update({track_id:Rating(track_id,rate,Conte
        return ratings
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()
        print('Error ', e)
        sys.exit(1)
```

```
    finally:
        if con:
            con.close()

def get_all_musics_ids():
    con = None
    track_ids = []
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        cur.execute("SELECT track_id FROM music")
        con.commit()
        for track in cur:
            track_id = track[0]
            track_ids.append(track_id)
        return track_ids
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()
        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()

def get_all_musics():
    con = None
    tracks = []
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        cur.execute("SELECT * FROM music")
        con.commit()
        for track in cur:
            track_name = track[0]
            track_id = track[1]
            track_album_name = track[2]
            track_album_id = track[3]
            track_genre = track[4]
            music = Model.Music(track_id,track_name,track_a
```

```
f_key = track[5]
music.set_feature('key',f_key)
f_mode = track[6]
music.set_feature('mode',f_mode)
f_danceability = track[7]
music.set_feature('danceability',f_danceability)
f_energy = track[8]
music.set_feature('energy',f_energy)
f_loudness = track[9]
music.set_feature('loudness',f_loudness)
f_speechiness = track[10]
music.set_feature('speechiness',f_speechiness)
f_acousticness = track[11]
music.set_feature('acousticness',f_acousticness)
f_instrumentalness = track[12]
music.set_feature('instrumentalness',f_instrumental
f_liveness = track[13]
music.set_feature('liveness',f_liveness)
f_valence = track[14]
music.set_feature('valence',f_valence)
f_tempo = track[15]
music.set_feature('tempo',f_tempo)
f_max_mag_freq = track[16]
music.set_feature('max_mag_freq',f_max_mag_freq)
f_roll_off  = track[17]
music.set_feature('roll_off',f_roll_off)
f_spectral_centroid = track[18]
music.set_feature('spectral_centroid',f_spectral_ce
f_bpm = track[19]
music.set_feature('bpm',f_bpm)
f_danceability_es = track[20]
music.set_feature('danceability_es',f_danceability_
f_energy_es = track[21]
music.set_feature('energy_es',f_energy_es)
f_power = track[22]
music.set_feature('power',f_power)
f_loudness_es  = track[23]
music.set_feature('loudness_es',f_loudness_es)
uid  = track[24]
music.set_uid(uid)
activity  = track[25]
```

```
            music.set_activity(activity)
            culture  = track[26]
            music.set_culture(culture)

            tracks.append(music)
        return tracks
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()
        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()


def get_track_features(track_id):
    con = None
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        cur.execute("SELECT * FROM music WHERE track_id='{
        con.commit()
        for track in cur:
            track_name = track[0]
            track_id = track[1]
            track_album_name = track[2]
            track_album_id = track[3]
            track_genre = track[4]
            music = Model.Music(track_id,track_name,track_a
            f_key = track[5]
            music.set_feature('key',f_key)
            f_mode = track[6]
            music.set_feature('mode',f_mode)
            f_danceability = track[7]
            music.set_feature('danceability',f_danceability
            f_energy = track[8]
            music.set_feature('energy',f_energy)
            f_loudness = track[9]
            music.set_feature('loudness',f_loudness)
```

```
            f_speechiness = track[10]
            music.set_feature('speechiness',f_speechiness)
            f_acousticness = track[11]
            music.set_feature('acousticness',f_acousticness)
            f_instrumentalness = track[12]
            music.set_feature('instrumentalness',f_instrumental
            f_liveness = track[13]
            music.set_feature('liveness',f_liveness)
            f_valence = track[14]
            music.set_feature('valence',f_valence)
            f_tempo = track[15]
            music.set_feature('tempo',f_tempo)
            f_max_mag_freq = track[16]
            music.set_feature('max_mag_freq',f_max_mag_freq)
            f_roll_off  = track[17]
            music.set_feature('roll_off',f_roll_off)
            f_spectral_centroid = track[18]
            music.set_feature('spectral_centroid',f_spectral_ce
            f_bpm = track[19]
            music.set_feature('bpm',f_bpm)
            f_danceability_es = track[20]
            music.set_feature('danceability_es',f_danceability_
            f_energy_es = track[21]
            music.set_feature('energy_es',f_energy_es)
            f_power = track[22]
            music.set_feature('power',f_power)
            f_loudness_es  = track[23]
            music.set_feature('loudness_es',f_loudness_es)
        return music
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()
        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()
```

```
def insert_user_rating(user_id,track_id,rate,context):
    con = None
    try:
        con = psycopg2.connect(connstring)
        cur = con.cursor()
        cur.execute("INSERT INTO user_ratings(user_id,trac
        con.commit()
    except psycopg2.DatabaseError as e:
        if con:
            con.rollback()
        print('Error ', e)
        sys.exit(1)

    finally:
        if con:
            con.close()


#Model.py

from enum import Enum

class User:
    def __init__(self, id, name):
       self._user_id = id
       self._user_name = name
       self._friends = []
       self._ratings = dict()

    def add_friend(self,user):
        self._friends.append(user)

    def get_name(self):
        return self._user_name

    def get_friends(self):
        return self._friends

    def get_id(self):
        return self._user_id

    def get_ratings(self):
```

```
        return self._ratings

    def add_rating(self,track_id, rating):
        self._ratings.update({track_id:rating})

    def set_ratings(self,ratings):
        self._ratings = ratings

class Rating:
    def __init__(self,track_id, rate, context):
        self._track_id = track_id
        self._rate = rate
        self._context = context

    def get_context(self):
        return self._context

    def get_track_id(self):
        return self._track_id

    def get_rate(self):
        return self._rate

class Music:
    def __init__(self, id, name, album_name, genre):
       self._track_id = id
       self._track_name = name
       self._track_genre = genre
       self._labum_name = album_name
       self._features = dict()

    def get_track_id(self):
        return self._track_id
    def get_track_name(self):
        return self._track_name
    def get_track_genre(self):
        return self._track_id


    def get_uid(self):
        return self._uid
```

```
    def set_uid(self, uid):
        self._uid = uid

    def get_activity(self):
        return self._activity
    def set_activity(self, activity):
        self._activity = activity

    def get_culture(self):
        return self._culture
    def set_culture(self, culture):
        self._culture = culture

    def get_features(self):
        return self._features

    def set_feature(self, feature, value):
        self._features.update({feature:value})


class Context:
    def __init__(self, activity, culture):
       self._activity = activity
       self._culture = culture



    def get_activity(self):
        return self._activity

    def get_culture(self):
        return self._culture

# activity : working, studying, running, driving
class ActivityTypes(Enum):
    Working = 1
    Studying = 2
    Running = 3
    Driving = 4
    Relaxing = 5
```

```
# culture : US, BR
class CultureTypes(Enum):
    US = 1
    BR = 2


import Persistance
import random
import Model


def define_users_ratings():
    tracks = Persistance.get_all_musics()
    users = Persistance.get_all_users()
    for user in users:
        print(user.get_name())
        i = 1
        tracks_rated = dict()
        #likes
        while(i <= 15):
            idx = random.randint(1,(len(tracks)-1))
            track = tracks[idx]
            track_id = track.get_track_id()
            if(not tracks_rated.__contains__(track_id)):
                context = Model.Context(track.get_activity(),tra
                Persistance.insert_user_rating(user.get_id(), t
                i+=1
        #dislikes
        i = 1
        while(i <= 15):
            idx = random.randint(1,(len(tracks)-1))
            track = tracks[idx]
            track_id = track.get_track_id()

            if(not tracks_rated.__contains__(track_id)):
                context = Model.Context(track.get_activity(),tra
                Persistance.insert_user_rating(user.get_id(), t
                i+=1


def main():
    if(not Persistance.check_table_exists('users')):
```

```
        Persistance.create_users_table()
        for i in range(1,101):
            Persistance.insert_user("User"+str(i))

    if(not Persistance.check_table_exists('user_ratings')):
        Persistance.create_user_ratings_table()
        define_users_ratings()


if __name__ == "__main__":
    main()


import sys, os
import Persistance
from Model import *
import numpy as np
import pandas as pd
import sklearn.metrics as metrics
from scipy.stats import pearsonr
from sklearn.neighbors import NearestNeighbors
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics.pairwise import cosine_similarity
from sklearn.metrics import pairwise_distances
import matplotlib.pyplot as plt

from matplotlib.colors import ListedColormap

#Parameters
total_contexts = 2
knn_k = 20
context_n = 10
n_recommendations = 20

def load_users():
    users = Persistance.get_all_users()
    for user in users:
        ratings = Persistance.get_all_ratings_user(user.ge
        user.set_ratings(ratings)
    return users
```

```
def calculate_smc(context_a, context_b):
    activity = 0
    culture = 0
    if(context_a.get_activity() == context_b.get_activity()):
        activity = 1
    if(context_a.get_culture() == context_b.get_culture()):
        culture = 1
    smc = (activity + culture )/total_contexts
    return smc



def filter_users_by_context(_user, context, knearest_users, size
    users_weights = dict()
    for user in knearest_users:
        if(user.get_id() == _user.get_id()):
            continue
        ratings = user.get_ratings()
        smc_acum = 0
        for track_id, rating in ratings.items():
            smc_acum =  smc_acum + calculate_smc(context,rating
        smc_avg = smc_acum/len(ratings)
        users_weights.update({user.get_id():smc_avg})
        sorted_users = sorted(users_weights.items(), key=lambda
    filtered_users = []
    for i in range(0,size):
        filtered_users.append(users[sorted_users[i][0]-1])
    return filtered_users

def get_ratings_dataframe(filtered_users):
    musics_ids = Persistance.get_all_musics_ids()
    users_ids = []
    df_data = []
    for user in filtered_users:
        users_ids.append(user.get_id())
        row = dict()
        row.update({'user_id': user.get_id()})
        ratings = user.get_ratings()
        for music in musics_ids:
            if(music not in ratings):
                row.update({music: -1})
```

```
            else:
                row.update({music: ratings[music].get_rate
        df_data.append(row)
    return pd.DataFrame(df_data,columns=musics_ids, index=


def get_recommendation(user_active, context, users, n_recon
    ######## FC ############
    users = load_users()

    ratings_df = get_ratings_dataframe(users)
    similarities, indexes = findksimilarusers(user_active.
    similar_users = []
    for index in indexes[0]:
        if((index + 1) == user_active.get_id()):
            continue
        similar_users.append(users[index])
    context_filtered_users = filter_users_by_context(user_a
    # ######## CB ##########
    related_users_music = []
    for user in context_filtered_users:
        ratings = user.get_ratings()
        for track_id, rating in ratings.items():
            if(rating.get_rate() == -1):
                continue
            music = Persistance.get_track_features(track_id
            related_users_music.append(music)

    user_musics = []
    filtered_user_ratings = filter_ratings_by_context(user
    for rating in filtered_user_ratings:
        if(rating.get_rate() == -1):
            continue
        music = Persistance.get_track_features(rating.get_
        user_musics.append(music)


    similarities = dict()
    recommendations=[]
    for music in user_musics:
        for candidate_music in related_users_music:
```

```
            profile_vector = np.asarray(create_feature_vector(m
            item_vector = np.asarray(create_feature_vector(cand
            cos_result = cosine_similarity(profile_vector,item_
            similarities.update({candidate_music.get_track_name
        break


    recommendations = sorted(similarities.items(), key=lambda k
    return recommendations[0:n_recommendations]


def create_feature_vector(music):
    feature_vector = []
    features = music.get_features()
    for label,value in features.items():
        feature_vector.append(value)
    return feature_vector

def filter_ratings_by_context(user,context):
    ratings_weights = dict()
    ratings = user.get_ratings()
    for track_id, rating in ratings.items():
        if(rating.get_rate() == -1):
            continue
        smc = calculate_smc(context,rating.get_context())
        ratings_weights.update({track_id:smc})
    sorted_tracks = sorted(ratings_weights.items(), key=lambda

    filtered_ratings = []
    for i in range(0,len(sorted_tracks)):
        filtered_ratings.append(ratings[sorted_tracks[i][0]])
    return filtered_ratings


def findksimilarusers(user_id, ratings, metric = 'cosine', k=10
    similarities=[]
    indexes=[]
    model_knn = NearestNeighbors(metric = metric, algorithm = '
    model_knn.fit(ratings)
    distances, indexes = model_knn.kneighbors(ratings.iloc[user
    similarities = 1-distances.flatten()
```

```
    return similarities, indexes


def main():
    users = load_users()
    user = users[0]
    prompt_activity = ('----------------------------------
        '|Por favor, insira a sua atividade atual:|\n'
        '| Trabalhando = 1                        |\n'
        '| Estudando   = 2                        |\n'
        '| Correndo    = 3                        |\n'
        '| Dirigindo   = 4                        |\n'
        '| Relaxando   = 5                        |\n'
        '|----------------------------------------|\n'
        '>>> ')
    activity_selection = int(input(prompt_activity))
    prompt_day_part = ('----------------------------------
        '|Por favor, insira sua cultura atual: |\n'
        '| BR = 1                                 |\n'
        '| US = 2                                 |\n'
        '|----------------------------------------|\n'
        '>>> ')
    culture_selection = int(input(prompt_day_part))

    recommendations = get_recommendation(user, Context(Act

    # k_evaluations = [5,10,15,20,25,30,40,50]
    # for k in k_evaluations:
    #     print('Knn: '+ str(k))
    #     knn_k = k
    #recommendations = get_recommendation(user, Context(Ac
    # musics = Persistance.get_all_musics()
    for recommendation in recommendations:
        print(recommendation)
        #     tp = 1
        #     for music in musics:
        #         if music.get_track_id() == recommendatio
        #             if(music.get_activity() == ActivityT
        #                 tp+=1
        #             break
```

```
        # print(tp/n_recommendations)
if __name__ == "__main__":
    main()
```

ANEXO B – Artigo

Desenvolvimento de um Sistema de Recomendação Musical sensível ao contexto

William K. Aliaga[1]

[1]Departamento de Informática e Estatística – Universidade Federal do Santa Catarina (UFSC)
Florianópolis – SC – Brazil

william.kraemer@inf.ufsc.br

Abstract. *Recommendation Systems have gained huge relevance for boosting several markets with their capacity to leverage sales, for that reason they have been widely studied by the academic communitysinde the ninety's. Recommendation systems are not trivial systems, they require multidisciplinary knowledge and specific field knowledge. The present work aims to propose and develop a musical recommendation model based on recommendation theory, seeking to use classic recommendation techniques and applying an extra layer of contextual filtering, adding contextual information to the process in order to test the benefits for the recommendation.*

Resumo. *Os sistemas de recomendação tem ganhado imensa relevância ao impulsionar diversos mercados com sua capacidade de alavancar vendas, por tal motivo têm sido amplamente estudados pela comunidade acadêmica desde os anos noventa. Sistemas de recomendação não são sistemas triviais, requerem conhecimentos multidisciplinares e conhecimentos específicos da área. O presente trabalho tem por objetivo propor e desenvolver um modelo de recomendador musical baseando-se na teoria de recomendação, buscando se valer de técnicas clássicas de recomendação e aplicando uma camada extra de filtragem contextual, adicionando informações contextuais ao processo com o intuito de testar os benefícios em potencial para a recomendação.*

1. Introdução

A industria musical é um dos maiores ramos do entretenimento e um dos mais relevantes da atualidade. Um relatório da Federação Internacional da Indústria Fonográfica (IFPI, na sigla em inglês) apresenta dados sobre o mercado de música digital, que em 2018 representou US$ 19,8 bilhões a nível global e obteve um crescimento de 9,7% em relação a 2017, sendo o quarto ano seguido de crescimento. Sem dúvidas os serviços de streaming de músicas tem um papel importante impulsionando o crescimento desse mercado, pois representam um total de 46.8% do total com aproximadamente 255 milhões de assinantes pagantes[IFPI 2019]. Tamanha popularidade dos serviços de streaming se deve em boa parte ao tamanho e variedade do acervo de músicas que apresentam. Um exemplo disso é o acervo do Spotify[1], maior empresa do mercado com 100 milhões de assinantes ao redor do mundo, que em 2019 atingiu a marca de 50 milhões de faixas, e que cresce em torno de 40 mil músicas por dia[Times 2019]. Com acervos de um tamanho tão grande, descobrir quais músicas são as mais apropriadas para serem apresentadas a

[1]http://www.spotify.com

um usuário se torna um desafio. É neste cenário que sistemas de recomendação se tornam ferramentas relevantes, pois sistemas de recomendação foram desenvolvidos para ajudar nesse problema, criando uma seleção de itens que seriam de interesse para o usuário, sem demandar muita iteração com ele(a)[Kunaver and Požrl 2017]. A pesar de tudo isso, os sistemas de recomendação tradicionais, na sua maioria, consideram apenas duas entidades como base para suas predições: itens e usuários, e não se considera o contexto (e.g. localização, temperatura e clima, horário) para aprimorar a recomendação. Este trabalho visa estudar o estado da arte de sistemas de recomendação, explorando a área de Sistemas de recomendação sensíveis ao contexto, apresentar um modelo para recomendação com filtragem contextual e um protótipo que realize recomendações ao usuário.

2. Fundamentação Teórica

É fundamental para a compreensão deste trabalho o entendimento de conceitos de Computação Musical, Sistemas de Recomendação e Sistemas de Recomendação Sensíveis ao Contexto.

2.1. Computação Musical

A computação musical faz estudos de métodos, técnicas e algoritmos para geração e processamento de som e música, representação e armazenamento de informação sonora e musical de maneira digital, mantendo sempre a premissa de que a informação tratada é arte diferenciando-se assim de outras áreas similares, como a computação gráfica.[Miletto et al. 2004]

Existem três características básicas de um som musical segundo[Orio et al. 2006]:

Altura tonal está relacionada com a percepção da frequência fundamental de um som, abrange de tons baixos(ou graves) até tons altos(ou agudos).

Volume está relacionado com a amplitude da vibração e sua energia, abrange de suave a forte, também chamada de intensidade.

Timbre é definido como as características do som que permitem diferenciar dois sons com mesma altura tonal e volume.

2.2. Sistemas de Recomendação

Sistemas de Recomendação são sistemas que ajudam o usuário a encontrar conteúdo, produtos ou serviços agregando e analisando sugestões de outros usuários[Park et al. 2012]. De acordo com [Bobadilla et al. 2013] e [Ricci et al. 2011], um sistema de recomendação pode ser caracterizado pelo seu algoritmo de filtragem. A divisão dos algoritmos de filtragem mais usada, os classifica em: filtragem colaborativa, filtragem baseada em conteúdo e filtragem híbrida [Adomavicius and Tuzhilin 2005]. Para [Bobadilla et al. 2013], as recomendações de um sistema baseado em conteúdo se baseiam nas escolhas anteriores feitas pelo usuário. Um sistema baseado em conteúdo analisa um conjunto de documentos avaliados por um usuário e usa o conteúdo destes, agregados à avaliação do usuário, para inferir um perfil de usuário que seria útil para recomendar outros itens interessantes[Park et al. 2012]. A ideia principal da Filtragem Colaborativa é a de tentar replicar a recomendações feitas entre amigos com gostos em comum no popular *boca-a-boca*. [Primo 2013]

Já a Filtragem Colaborativa é considerada a técnica mais popular e a mais amplamente utilizada. A maneira mais simples de abordar esta técnica é recomendando ao usuário ativo, os itens escolhidos por outros usuários com preferências similares. Tal similaridade é calculada baseando-se no histórico de classificações dos usuários [Ricci et al. 2011]. Em outras palavras, sistemas de recomendação com filtragem colaborativa tentam prever a utilidade de itens para um usuário em particular baseando-se nos itens avaliados anteriormente por *outros* usuários[Adomavicius and Tuzhilin 2005].

Por último, [Burke 2002] define os sistemas híbridos como sistemas que combinam duas ou mais técnicas de recomendação para ganhar maior performance ao mesmo tempo que evitam os problemas de cada abordagem individual. Para ele, as abordagens mais comuns, consistem em combinar a técnica de filtragem colaborativa combinada com outras.

3. Sistemas de Recomendação Sensíveis ao Contexto

[Dey and Abowd 2000] define que sistemas são sensíveis ao contexto se usam contexto para providenciar informação relevante ou serviços ao usuário, onde a relevância depende da tarefa realizada.

[Dey 2001] define contexto como qualquer informação a ser usada para caracterizar a situação de uma entidade, seja esta uma pessoa, lugar ou objeto considerado relevante para a interação usuário-aplicação (incluindo eles próprios).

Para [Adomavicius and Tuzhilin 2011] e [Steeg 2015] existem três paradigmas para incorporar o contexto com a recomendação utilizada. Informação contextual pode ser obtida das seguintes maneiras segundo[Adomavicius and Tuzhilin 2011]:

Explicitamente onde pessoas relevantes ou outras fontes de informação contextual são diretamente abordadas, seja por meio de perguntas diretas ou obtendo tal informação por outros meios.

Implicitamente onde é possível obter informação contextual ao se acessar diretamente a fonte de dados e usar esses dados relativos ao ambiente do usuário, sem ter algum tipo de interação com o usuário.

Inferência por meio do uso de estatística ou de mineração de dados, o contexto pode ser inferido. Para isto, torna-se necessária a construção de um modelo preditivo (um classificador) e o treinamento deste com dados apropriados, que influencia significativamente na inferência da informação contextual.

4. Modelo e Implementação de um Sistema de Recomendação Musical Sensível ao Contexto

Este trabalho se propõe a desenvolver um sistema de recomendação que se baseia no contexto de uma atividade para realizar uma recomendação, visando esclarecer a hipótese de que um recomendador contextual tem um desempenho melhor do que um que desconsidera o contexto. Como resultado do estudo realizado é possível elaborar uma lista de requisitos mínimos desejáveis para um Sistema de Recomendação Musical Sensível ao Contexto. Estes requisitos são os seguintes:

- Identificação de usuários com perfis similares
- Identificação de contextos similares

- Adaptabilidade a mudanças no contexto
- Recomendação de músicas de acordo com o interesse e o contexto dos usuários

Considerando os requisitos apresentados acima, foi definido um modelo para o sistema de recomendação. A Figura 1 mostra a arquitetura seguida pelo SR, onde se observa a abordagem híbrida que apresenta uma etapa de filtragem colaborativa, seguido por uma etapa de pós-filtragem contextual e em seguida uma última etapa de filtragem baseada em conteúdo.

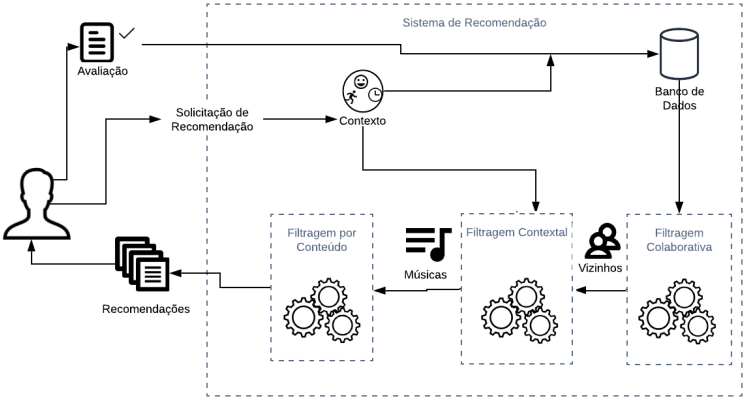


Figura 1. Processo de Recomendação(Fonte: Elaboração própria)

4.1. Perfil do Usuário

Um usuário dentro do sistema é modelado por meio de uma representação do seu perfil de usuário, que consiste no registro do seu histórico de transações e suas avaliações em um sistema binário, ou seja, positivas ou negativas(1 ou 0 respectivamente), para cada música escutada e uma anotação de contexto.

4.2. Contexto

Para este trabalho, o contexto com o que foi delimitado em duas categorias, a primeira sendo a *Atividade* que o usuário ativo pretende realizar, podendo assumir os valores de *Correr, Trabalhar, Relaxar, Dirigir ou Estudar* e a segunda a *Cultura* que esse usuário pertence, podendo assumir os valores de *BR ou US*. Estas categorias serão usadas como classes para a classificação das músicas posteriormente.

4.3. Base de dados e Pré-Processamento

A Figura 2 ilustra o processo realizado para a criação do conjunto de dados por meio da obtenção de arquivos de áudio, por meio de uma varredura na Spotify API [Spotify 2019], e sua extração de características.

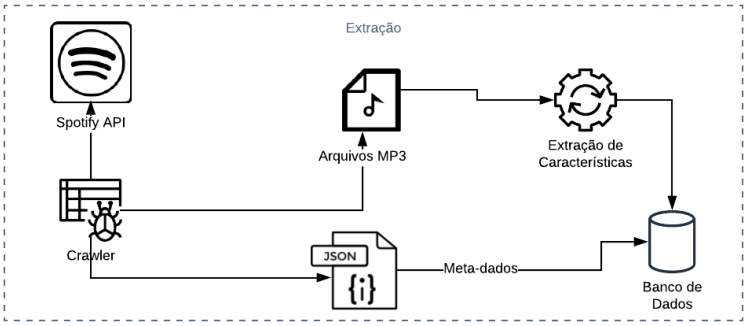


Figura 2. Processo de obtenção da base de músicas e extração de suas características(Fonte: Elaboração própria).

Para este trabalho foram coletadas músicas disponíveis para download através da API do Spotify. Para isso foi desenvolvido um *Crawler* que realizou uma varredura para conseguir arquivos de áudio em mp3 e metadados de 2106 músicas que estavam contidas em listas de reprodução. O critério para escolhas destas listas de reprodução foi que o nome delas contivesse palavras ou frases que determinassem os contextos com que as músicas contidas nelas estivessem relacionados (e.g. *correndo à noite*, *músicas para relaxar*). Uma vez que se tem uma base com arquivos de áudio torna-se possível a extração de características para cada música e para isso foi utilizada a biblioteca *Essentia*[Essentia 2019]. Tentou-se coletar a maior quantidade de características que fosse possível para enriquecer a base de dados para obter uma vantagem ao se fazer recomendações. As características extraídas foram as apresentadas na Tabela1:

Os parâmetro de configuração usados foram tamanho de frame de 30 segundos, o parâmetro de tipo de janela usando Hann e a taxa de amostragem escolhida foi a de 44100Hz.

4.4. Conjunto de Dados

Para poder trabalhar com um sistema de recomendação como o proposto neste trabalho precisa-se de dados de usuários e histórico de avaliações destes, como esse não é um dado disponível via Spotify API, foram gerados 100 usuários junto com 30 avaliações positivas ou negativas, um sistema binário, para cada um deles a qual contém uma anotação de contexto de acordo com a classificação de contexto da música. Esta classificação de contexto da música é usada neste momento para obter uma simulações de iteração do usuário com o sistema de maneira mais realista e posteriormente para avaliar a recomendação fornecida pelo sistema. Essa quantidade de avaliações por usuário é usada para evitar um problema de arranque a frio, como descrito anteriormente. Desta maneira, foi gerado um histórico de 3000 iterações de todos os usuários que forma o conjunto de dados a ser empregado.

Tabela 1. Conjunto de características extraídas. Fonte: Elaboração Própria

Característica	Descrição	Motivação
MaxMagFreq	Calcula a frequência com maior magnitude em um espectro.	Tem relação com a altura tonal em um som.
RollOff	Calcula a frequência de roll-off em um espectro	Usado para distinguir sons harmônicos e não harmônicos
Spectral Centroid	Calcula a Centroide Espectral de um sinal	Tem relação com a frequência dominante de um som.
BPM	Calcula o ritmo em BPM em um sinal	Tem relação com o tempo da música
Danceability	Estima numericamente que tão dançável é um sinal de áudio	Interessante para o usuário segundo o contexto
Energy	Calcula a energia em um sinal	Interessante para o usuário segundo o contexto
Power	Calcula o poder instantâneo apresentado por uma amostra	Interessante para o usuário segundo o contexto
Loudness	Calcula o volume apresentado por um sinal	Interessante para o usuário segundo o contexto
Instrumentalness	Calcula a ausência de Vocais em um sinal	Interessante para o usuário segundo o contexto
Speechiness	Calcula a presença de palavras faladas no sinal.	Interessante para o usuário segundo o contexto
Liveness	Calcula a presença de audiência no sinal da gravação	Interessante para o usuário segundo o contexto
Valence	Calcula a "positividade" no sinal de uma música. i.e. Alegria, Felicidade, Euforia.	Interessante para o usuário segundo o contexto

4.5. Filtragem Colaborativa

Para realizar a filtragem colaborativa, precisamos levar em consideração não só as avaliações anteriores do usuário para atingir recomendações personalizadas, mas também descobrir quais outros usuários existem com gostos similares, pois poderiam ter ouvido músicas que seriam de interesse para o usuário ativo. Neste módulo do sistema foi utilizada uma técnica bastante tradicional na FC, o algoritmo K-Vizinhos Mais Próximos

(KNN), a ideia por trás disso é achar similaridades nas avaliações anteriores com outros usuários e ao escolher esses usuários, obter uma lista com as possíveis músicas para recomendar, isto é, usar as avaliações positivas desses usuários selecionados. A motivação principal para ter escolhido esta técnica é a de que se trata de um algoritmo *lazy*, que não gera um modelo que precisa de treinamento, logo o sistema pode se adaptar a mudanças rápidas na matriz de avaliações de usuário.

4.6. Filtragem Contextual

Ao se obter os usuários mais próximos ao usuário ativo, por meio da FC, é realizada mais uma filtragem, dessa vez reduzindo ainda mais o escopo desses usuários próximos para obter somente os que avaliaram músicas em contextos similares com o apresentado pelo próprio usuário ativo. Para fazer essa seleção, surge a necessidade de se calcular a similaridade entre dois contextos. A técnica escolhida para fazer esse cálculo foi Coeficiente de Semelhança Simples, uma vez que o contexto atrelado à avaliação do usuário em relação a uma música é um dado qualitativo ao invés de quantitativo, ao comparar dois deles, apenas podemos afirmar se são iguais ou diferentes em cada categoria de contexto. Este calculo é aplicado para todo usuário da lista de vizinhos que a etapa de FC produziu, os usuários que na média, tiverem mais avaliações com contexto mais similar ao contexto apresentado pelo usuário ativo são escolhidos.

4.7. Filtragem Baseada em Conteúdo

Na última etapa antes de realizar a recomendação de fato, é realizada mais uma filtragem. Nesse ponto, existe uma lista com os usuários mais similares de acordo com o contexto. Assim, temos acesso às músicas que esses usuários tem maior afinidade em um contexto similar, logo, desejamos selecionar quais dessas músicas são mais parecidas com as músicas já avaliadas de maneira positiva anteriormente pelo usuário. Para solucionar este problema, é aplicada uma técnica que nos permita calcular a similaridade entre duas músicas. Durante a construção do conjunto de dados, diversas características de cada música foram extraídas e agregadas, esta série de características pode ser facilmente interpretada como um vetor. Logo, precisa-se de uma medida onde podemos calcular o quão similares dois vetores são, é o caso da *similaridade cosseno*, após aplicar essa técnica comparando todas as músicas do usuário ativo com aquelas escutadas e avaliadas pelos usuários que tem a maior semelhança com o usuário ativo e seu contexto, e posteriormente ordenando por relevância, obtemos uma lista com as n músicas a serem recomendadas ao usuário. Esta lista, se encontra ordenada do item mais relevante, para o item menos relevante.

5. Avaliação de Resultados

Para os testes a serem realizados, foram escolhidos 10(dez) usuários a esmo e um contexto em comum que manterão com valor fixo de 'Estudar' para atividade e 'BR' para cultura, com intuito de garantir que a entrada seja sempre a mesma para o contexto e assim isolar variáveis que afetam as mudanças no resultado da recomendação. As variáveis propostas para os experimentos são:

K diferentes valores de K para o algoritmo KNN, para avaliar os possíveis efeitos do tamanho da lista de vizinhos similares, os valores assumidos serão $k = \{5, 10, 15, 20\}$.

Contexto representa o valor que o filtro de contexto pode assumir, no caso, valores do conjunto $Contexto = \{Com\ Contexto, Sem\ Contexto\}$. A ideia é avaliar se realmente a filtragem contextual consegue trazer benefícios para a recomendação.

Usuário serão feitos testes com dez usuários diferentes para poder ter dez cenários diferentes para avaliar a recomendação usando o mesmo contexto.

Foram realizados em torno de 240 testes diferentes para compilar os dados dos experimentos. Estes dados foram coletados verificando a quantidade de VP, VN, FP e FN de acordo com as métricas *Precisão em N* e *MAP* para criar um gráfico comparativo, indicando assim a taxa de acertos do recomendador.

Em seguida, na Figura 3, temos o comparativo para *MAP* entre todos os cenários com diferentes usuários.

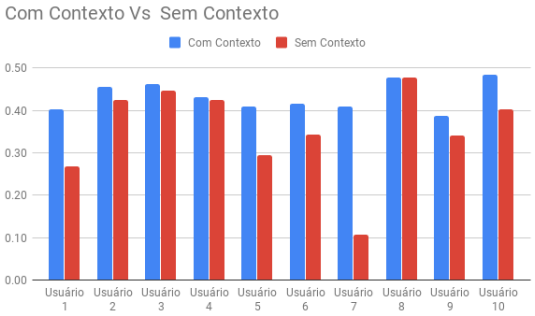


Figura 3. Gráfico de MAP para todos os Usuários. Fonte: Elaboração própria

O primeiro detalhe que chama a atenção avaliando os *MAP* para todos os usuários é que no geral, o recomendador apresenta uma taxa aceitável de recomendações corretas, o único momento que esta taxa se apresentou inferior a 20% foi no teste realizado com o Usuário 7 sem aplicação de filtragem contextual. Ressaltando que o *MAP* é a média das médias de Precisão, percebemos que é um bom resultado para um recomendador simples como este. Um fator que chama à atenção é o limite de precisão de 50%, isto pode ser causado por alguma relação matemática entre a quantidade fixa de avaliações positivas e negativas dos usuários, dada a maneira como foi gerada essa base de usuários. Um outro fator que poderia ser investigado posteriormente, em decorrência disto, seria se essas taxas de acerto não poderiam ser maiores um tamanho maior de conjunto de dados. Outro ponto que deve ser discutido é que não necessariamente a recomendação se beneficia da filtragem contextual se os dados disponíveis não forem de utilidade para melhorar o filtro. Em relação a isso, pode se perceber que na maioria dos cenários a recomendação contextual apresentou melhorias em relação a uma recomendação sem contexto, porém é possível observar que em alguns casos parece não apresentar melhorias, como é o caso do cenário do Usuário 8 onde a precisão se manteve igual em ambos casos.

6. Conclusão

Embora muitas melhorias possam ser realizadas ainda nesse SR, este trabalho conseguiu atingir seu objetivo de desenvolver e avaliar um Sistema de Recomendação Musical Sensível ao Contexto e ainda expor alguns problemas detectados, inclusive propondo abordagens para solucionar estes. O estudo em sistemas de recomendação no capítulo 3 e o estudo das SRs sensíveis ao contexto apresentados no capítulo 4 foram a base para o desenvolvimento deste trabalho. Com ajuda destes conhecimentos, foi possível criar uma base de dados com anotações em contexto, que ajudou a modelar o sistema, e a desenvolver uma solução capaz de realizar recomendações e ainda testá-las. Com os resultados dos testes apresentados, a pesar de serem bastante simples, foi possível avaliar que a recomendação musical pode de fato se beneficiar da aplicação de contexto para seu calculo. No entanto, cabe ressaltar que as capacidades do SR proposto são bastante limitadas por alguns fatores, como o fato do conjunto de dados ser bastante pequena em relação a um conjunto usado em um cenário real. Isso impede o recomendador de inferir maiores peculiaridades sobre o gosto do usuário em relação à música em certos contexto. A resolução deste problema é por meio da obtenção continua de informações tanto sobre músicas quanto sobre os usuários, porém pode ser algo bastante custoso, tanto em termos computacionais, quanto financeiros. A obtenção de contexto também pode ser melhorada em muitos aspectos desde a maneira em como é obtida até a quantidade de categorias a serem incluídas.

6.1. Trabalhos Futuros

Em seguida estão reunidas algumas melhorias e continuidade nos estudos relativos ao sistema de recomendação apresentado.

- Estudar e implementar uma solução que possa extrair e inferir o contexto do usuário ativo no momento.
- Realizar novos experimentos com novas variáveis e maior quantidade de músicas envolvidas.
- Desenvolver um aplicativo que possa disponibilizar o sistema de recomendação para testes reais com usuários.
- Implementar algoritmos no módulo de filtragem baseada em conteúdo que consigam inferir a relação direta entre as características das músicas e as classes de contextos diferentes.
- Implementar novos algoritmos para realizar recomendações alternativos para combater problemas como arranque a frio ou sobre-especialização.
- Levantar novos requisitos que possam vir a ser incluídos no sistema para aumentar sua aplicabilidade em um cenário real

Referências

Adomavicius, G. and Tuzhilin, A. (2005). Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE transactions on knowledge and data engineering*, 17(6):734–749.

Adomavicius, G. and Tuzhilin, A. (2011). Context-aware recommender systems. In *Recommender systems handbook*, pages 217–253. Springer.

Bobadilla, J., Ortega, F., Hernando, A., and Gutiérrez, A. (2013). Recommender systems survey. *Knowledge-based systems*, 46:109–132.

Burke, R. (2002). Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370.

Dey, A. K. (2001). Understanding and using context. *Personal and ubiquitous computing*, 5(1):4–7.

Dey, A. K. and Abowd, G. D. (2000). Providing architectural support for building context-aware applications.

Essentia (2019). Essentia lib. `https://essentia.upf.edu/documentation/documentation.html`. Accessed: 2019-03-25.

IFPI (2019). Ifpi global music report 2019. `https://www.ifpi.org/news/IFPI-GLOBAL-MUSIC-REPORT-2019`. Accessed: 2019-04-30.

Kunaver, M. and Požrl, T. (2017). Diversity in recommender systems–a survey. *Knowledge-Based Systems*, 123:154–162.

Miletto, E. M., Costalonga, L. L., Flores, L. V., Fritsch, E. F., Pimenta, M. S., and Vicari, R. M. (2004). Introdução à computação musical. In *IV Congresso Brasileiro de Computação*.

Orio, N. et al. (2006). Music retrieval: A tutorial and review. *Foundations and Trends® in Information Retrieval*, 1(1):1–90.

Park, D. H., Kim, H. K., Choi, I. Y., and Kim, J. K. (2012). A literature review and classification of recommender systems research. *Expert Systems with Applications*, 39(11):10059–10072.

Primo, T. T. (2013). Método de representação de conhecimento baseado em ontologias para apoiar sistemas de recomendação educacionais.

Ricci, F., Rokach, L., and Shapira, B. (2011). *Recommender systems handbook*. Springer.

Spotify (2019). Spotify api. `https://developer.spotify.com/web-api/`. Accessed: 2019-04-30.

Steeg, F. v. (2015). Context-aware recommender systems. Master's thesis.

Times, N. Y. (2019). Spotify reaches 100 million subscribers, but not without some dissonance. `https://www.nytimes.com/2019/04/29/business/media/spotify-100-million-subscribers-apple-podcasts.html`. Accessed: 2019-04-30.