Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classificação de Parâmetros de Efeitos de Distorção a partir de Áudio Digital

Arthur da Veiga F. Borges

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia de Computação

Orientador
Prof. Dr. Díbio Leandro Borges

Brasília
2021

Universidade de Brasília
Instituto de Ciências Exatas
Departamento de Ciência da Computação

Classificação de Parâmetros de Efeitos de Distorção a partir de Áudio Digital

Arthur da Veiga F. Borges

Monografia apresentada como requisito parcial
para conclusão do Curso de Engenharia de Computação

Prof. Dr. Díbio Leandro Borges (Orientador)
CIC/UnB

Prof. Geraldo P. R. Filho
CIC/UnB

Prof. Luís Paulo Faina Garcia
CIC/UnB

Prof. Dr. Joao José Costa Gondim
Coordenadora do Curso de Engenharia de Computação

Brasília, 7 de maio de 2021

Dedicatória

Dedicado a meus pais Nonato e Rosana, e a meus avós Ozair e Célia. Sem vocês eu não teria chegado até aqui. Louvo muito a Deus pela vida de vocês.

Agradecimentos

Antes de mais nada, quero agradecer a Deus por até aqui ter me ajudado e me capacitado nesta jornada.

Também quero agradecer a meus pais Nonato e Rosana, meus avós Ozair e Celia, ao meu irmão, minhas irmãs, meus tios e primos que se fizeram presentes, que me apoiaram e estiveram comigo ao longo deste curso, e ainda antes dele. Vocês são muito importantes para mim.

Ao meu orientador Díbio Leandro Borges, que me direcionou neste trabalho e nos anteriores, e tem me ensinado muito. À Sociedade Brasileira de Computação, especialmente os pesquisadores que se especializam na área de Computação Musical, que me abriu os horizontes para esta área tão promissora, em 2019. Ao professor Tiago Fernandes Tavares, da FEEC/Unicamp, que me ajudou a idealizar este trabalho final durante o Simpósio Brasileiro de Computação Musical de 2019. Muito obrigado a todos vocês.

À minha namorada Beatriz, que tem me apoiado muito nos momentos mais importantes.

A todos os amigos que ganhei na UnB. Cada um de vocês somou de alguma forma.

Desde já agradeço demais por tudo.

Resumo

O processo de um músico que testa valores de parâmetros em efeitos de distorção sonora dado um áudio digital é típico no campo da Produção Musical. Mas, neste processo pode ocorrer erros humanos. Com isso, pôde-se observar que este é um processo de classificação supervisionada. Por isso, foi proposta uma simulação de classificação de parâmetros de efeitos de distorção a partir de áudio digital, onde três estimadores (SVM, KNN e Random Forest) comprovadamente apropriados para classificação de gêneros musicais, de parâmetros de efeitos de reverb e de detecção de intrusos em florestas sob proteção ambiental por áudio foram otimizados, testados e avaliados para quatro efeitos de distorção diferentes (Fuzz, Bitcrusher, Overdrive e Decimator). O dataset foi gerado a partir da aplicação de algoritmos de distorção em sinal senoidal e, a partir daí, da extração de características espectrais. Para avaliação, foram usadas matrizes de confusão e f1-score das classificações para cada parâmetro ou conjunto de parâmetros de cada efeito. Mesmo com limitações no dataset, o modelo Random Forest estimou mais de 90% de f1-score em 4 das 6 predições de parâmetros individuais. A partir deste resultado um dos possíveis trabalhos futuros é melhorar o dataset considerando utilizar a resposta ao impulso como áudio antes do pré-processamento e explicitar a correlação amostra-a-amostra ao extrair as características espectrais.

Palavras-chave: Aprendizado supervisionado, inteligência artificial, áudio, efeitos sonoros digitais, distorção

Abstract

The process of a musician testing parameter values in sound distortion effects given a digital audio is typical in the field of Music Production. But, in this process, human error can occur. It is clear that this process is a supervised classification. So, this work proposes a simulation on classification of distortion effect parameters from digital audio, where three estimators (SVM, KNN and Random Forest) proven to be appropriate for classification of musical genres, parameters of reverb effects and wildlife intruder detection by audio were optimized, tested and evaluated on four different distortion effects (Fuzz, Bitcrusher, Overdrive and Decimator). A dataset was generated by applying distortion algorithms in a sinusoidal signal and, from this, its spectral features were extracted. Confusion matrices and f1-score metric were used for evaluation on each parameter or parameter set of each effect. Even with limitations in the dataset, Random Forest estimated more than 90% f1-score in 4 of the 6 predictions on individual parameters. From this result one possible future work is to improve the dataset by considering using the impulse response as audio before preprocessing and making explicit the frame-by-frame correlation when extracting the spectral features.

Keywords: Supervised learning, artificial intelligence audio, digital sound effects, distortion

Sumário

Lista de Figuras

Lista de Tabelas

Capítulo 1

Introdução

Para aprofundar neste trabalho, é importante tratar sobre áudio digital e o fenômeno que compõe o arquivo de áudio: o som.

1.1 Conceitos sobre Som e Áudio Digital [1]

Sons são variações periódicas na pressão atmosférica, que referimos como ondas sonoras, e têm três características principais: amplitude, frequência, fase e forma de onda.

- ***Amplitude:*** descreve a intensidade da onda sonora. Em termos de percepção auditiva, se trata da intensidade do som. Quanto maior a distância vertical entre o pico de uma onda sonora e seu zero, mais alto é o som.

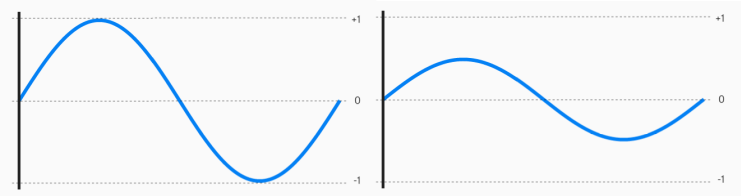

Figura 1.1: Pode-se perceber que a amplitude do sinal da direita é a metade da amplitude do sinal à esquerda.

- ***Frequência:*** o número de ciclos por segundo, onde um ciclo é definido de um pico positivo até o próximo pico positivo. Em termos de percepção auditiva, é a indicação de tom, isto é, quão agudo ou grave o som é. Quanto mais ciclos na onda sonora, mais agudo é o som.

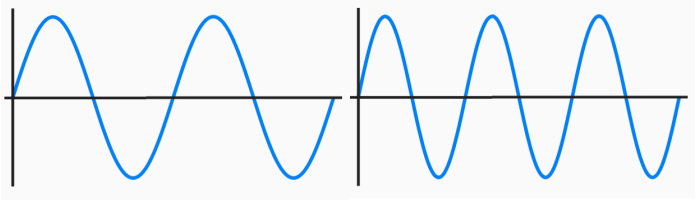

Figura 1.2: Representação gráfica de que frequência se trata de número de ciclos por segundo em um sinal. O sinal à esquerda tem 2 Hz, e o sinal da direita tem 3 Hz. Quanto mais ciclos na onda sonora, mais agudo é o som.

- ***Fase:*** É o ponto do ciclo em que a onda se inicia. É medido em graus (entre 0 e 359).
- ***Forma de onda:*** se trata de um espectro de frequências que compõem uma onda sonora complexa. É ela que compõe o **timbre**. Cada frequência que faz parte deste espectro é chamada de harmônica. Em termos de percepção auditiva, o timbre é análogo à impressão digital, uma identidade sonora de um elemento em que se pode produzir som. A alteração de amplitude ou de frequência de pelo menos um destes harmônicos modifica o timbre como um todo. Analogicamente, esta característica pode ser facilmente percebida na interação com um objeto qualquer. E isto se deve ao material e ao formato dos objetos que interagiram entre si, e à forma como foi feita esta interação. Por exemplo, tocar um violão com palheta tem um timbre diferente em relação a tocar sem palheta, por causa da interação do plástico da palheta ou da pele com a corda. Já em termos mais digitais, de acordo com a figura **??**, é possível perceber a mudança que ocorre em um timbre ao alterar seus harmônicos.

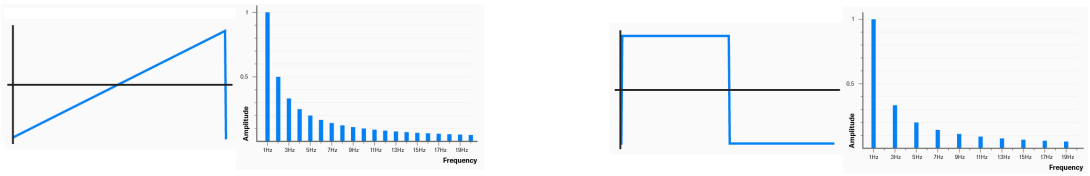

Figura 1.3: A figura da esquerda representa uma onda dente-de-serra, que é composta por todos os harmônicos. Se retirar os harmônicos pares, resulta na representação da onda quadrada, que é composta somente por harmônicos ímpares. Como consequência, há a mudança de timbre.

E áudio digital é a representação digital de uma onda sonora a partir de um código binário em um arquivo de computador. Isto é possível devido à conversão do som analógico para o digital na captação. E este processo tem como consequencia uma certa perda de dados. Por isso, o som digital nunca poderá representar o som analógico completamente. Mas, com o avanço tecnológico, este processo de conversão atingiu uma precisão a ponto de não haver diferenças entre o som analógico e sua representação digital, na perspectiva

do ouvido humano. Sua precisão varia de acordo com a taxa de amostragem de frequência e a quantidade, ou profundidade, de bits para cada amostra, ou bit depth. Quanto maiores esses valores, maior será a fidelidade do som digital em relação ao som analógico.

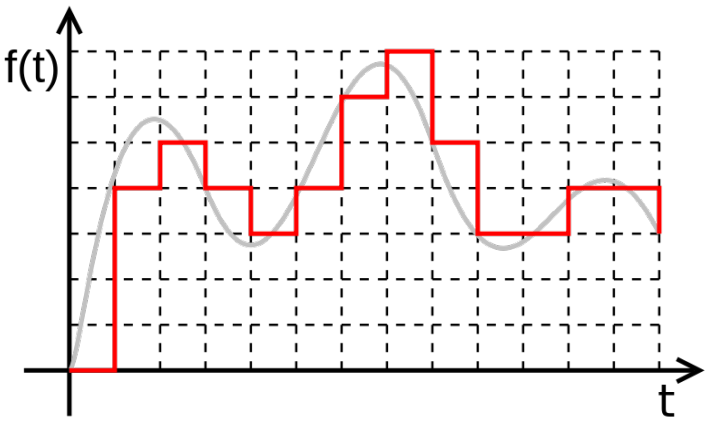

Figura 1.4: A onda sonora em cinza é representada digitalmente no sinal em vermelho. Observe que ocorrem algumas perdas de dados.

Com isso, é possível criar e manipular o som de forma analógica ou digital. A forma analógica se refere à criação ou replicação de sons utilizando objetos físicos em geral, como instrumentos musicais, etc. Podem-se encontrar muitos exemplos de síntese analógica dentro do escopo cinematográfico, principalmente no uso da técnica Foley, que é a reprodução de efeitos sonoros complementares para pós-produção de um filme, melhorando a qualidade do áudio. Já a digital se refere à criação ou replicação de sons utilizando ferramentas e técnicas aplicadas a sinais digitais. Alguns exemplos desta área de síntese digital envolvem design de som utilizando plugins VST (Virtual Instruments - Instrumentos Virtuais), que são muito utilizados no escopo de produção musical em geral. Um exemplo que é possível mencionar é a criação de efeitos sonoros para um jogo, utilizando a tecnologia MIDI.

1.2 Síntese Sonora e Processamento de Som [1]

Com o ajuste destas características, é possível manipular som em forma de áudio digital a partir da criação por meio de síntese sonora e/ou a partir da modificação deste arquivo existente por meio de processamento de som.

A síntese sonora é o processo de criação de um som a partir de dispositivos eletrônicos e/ou softwares. É definir os componentes de frequência para geração de um som complexo de forma analógica ou digital. Analogicamente, se trata do instrumento em si interagindo com um microfone para captação/gravação. Já digitalmente, a síntese se dá com a as-

censão dos sintetizadores, que permite a manipulação de formas de onda mais simples (senóide, quadrada, triangular e dente-de-serra) para a criação de sons mais complexos.

E, com um áudio gravado ou sintetizado, é possível aplicar o processamento de som para uma finalidade específica. Processamento de som se trata de transformação de sons definidos utilizando dispositivos eletrônicos e/ou softwares. São os chamados efeitos sonoros. Esta área está relacionada à síntese sonora porque toda ferramenta e dispositivo relacionado a isso é, de alguma forma, um processador de áudio. E, utlilzando-os, é possível criar sons novos a partir da modificação de timbres existentes, por mais simples que sejam.

Sabe-se que sons são compostos por vários componentes de frequência. E modelar esses componentes de alguma forma se torna viável utilizando filtros e efeitos para sinais já definidos. Entre alguns exemplos é possível mencionar o efeito de reverb, que é um processo natural da interação do som com o ambiente. É quando o som bate nos materiais do ambiente e retornam alterados aos ouvidos de alguém. Por exemplo, quando uma pessoa fala em uma sala sem móveis, ela ouve "ecos"distorcidos de sua própria voz. Ou, quando fala em uma câmara anecóica, ela não ouve nenhum retorno do ambiente. E também existem efeitos que são derivados de filtros e manipulação direta ou indireta de parâmetros de um áudio, como flange, wah-wah, distorção, chorus, etc. Portanto, como o resultado da transformação de um áudio é a criação de um novo áudio, pode-se dizer que isto também é uma forma de síntese sonora.

1.3 Problema e Motivação

Em cada efeito sonoro há parâmetros que os definem, e que variam de acordo com a natureza deste efeito. Por exemplo, o reverb e a distorção são efeitos sonoros diferentes entre si. Portanto, também adotam parâmetros diferentes, de acordo com sua montagem, como observado na figura 1.5. E essas transformações sonoras dependem de um conjunto de parâmetros que são controlados pelos próprios usuários, seja por automação ou manualmente ao longo do tempo.

Figura 1.5: Pode-se observar que, tanto analógica quanto digitalmente é possível observar que cada efeito sonoro tem seus parâmetros específicos.

Um dos problemas recorrentes na área de produção musical é: dada uma referência, como saber qual o efeito sonoro e os parâmetros que configuram o timbre resultante em um instrumento específico? A solução mais direta para isso é aplicando um ou vários efeitos sonoros no sinal de saída, testando os valores dos parâmetros para aproximar o máximo possível da referência. E nem sempre estes efeitos aplicados são exatamente os que o autor da referência utilizou. Por exemplo, um guitarrista que tem pedais de efeitos sonoros geralmente busca entender melhor o funcionamento de cada um deles, e o que cada parâmetro traz como resultado no som, para ajustar mais rapidamente a um timbre desejado. E consequentemente, várias configurações diferentes podem ser testadas podendo resultar em um desejado resultado aproximado. Mas também isto abre viés para erro da parte do guitarrista em conseguir aproximar os parâmetros dos seus elementos de processamento sonoro, etc.

Portanto, dado um áudio desejado, podemos perceber que este processo de busca paramétrica é uma forma de classificação onde os parâmetros são as classes e o sinal de áudio resultante desta configuração traz as características para a classificação.

Com isso, este trabalho visa simular este processo utilizando aprendizado de máquina supervisionado para classificar parâmetros de efeitos de distorção a partir de áudio digital, testando e avaliando duas formas de redução de dimensionalidade no dataset e três modelos que tiveram bons resultados para outros conteúdos de áudio digital.

Capítulo 2

Fundamentação Teórica

Primeiro, é importante mencionar sobre o tipo de dado a ser trabalhado. O arquivo de áudio é um material uni-dimensional, que armazena sons em uma sequência de amostras chamadas *frames*, onde têm sua quantidade de bits pré-determinada.

A partir dele, é possível extrair características temporais, como duração do áudio, andamento (caso seja uma música) ou amplitude de cada amostra que compõe o áudio. Mas, a maioria das características que um áudio pode oferecer são baseadas na Transformada de Fourier e, consequentemente, são espectrais. Além disso, é possível adquirir e medir o timbre de instrumentos e a sua periodicidade espectral ao longo do tempo, a partir dos Coeficientes Cepstrais de Mel-Frequência, que será abordado mais à frente.

O conteúdo dentro de um arquivo de áudio pode ser interpretado como música, voz ou um som específico de curta duração, derivado de captação ou de síntese sonora. E, como a finalidade deste trabalho é lidar diretamente com o efeito sonoro dentro de um contexto de produção musical, o conteúdo adotado foi o de som específico gerado por síntese.

Com isso, as principais literaturas que inspiraram este trabalho foram estudadas tendo em mente esta perspectiva.

2.0.1 Classificação de Gêneros Musicais via Aprendizado de Máquina [2]

Gênero musical é uma característica chave de qualquer música, que pode direcionar usuários à suas categorias preferidas. Vários desses usuários criam playlists baseadas nesta característica, levando a possíveis aplicações como a recomendação e gerenciamento de playlists.

Em 2017, Li Guo et al. dissertaram sobre o estabelecimento de um sistema baseado em várias técnicas de aprendizado de máquina, como o SVM e o KNN. O objetivo deste

trabalho é utilizar este classificador de gênero musical para classificar uma nova música corretamente, dadas suas características associadas.

Este foi o artigo que mais influenciou o trabalho proposto nesta monografia, uma vez que utilizou a biblioteca LibROSA para extração de características e os modelos SVM e KNN. A partir deste processo surgiu a premissa de testar estes modelos para parâmetros de efeitos de distorção.

2.0.2 Classificação Automática de Gêneros Musicais usando modulação de característica de Contraste Espectral [3]

Tendo o gerenciamento de grandes bancos de dados de música digital como premissa, Lee et al. propuseram uma nova característica chamada Modulação de Contraste Espectral baseada em oitavas (OMSC), que é extraída da análise de longo termo da modulação espectral para representar o comportamento de variação temporal dos sinais musicais.

2.0.3 Uma abordagem de Aprendizado de Máquina para aplicação em Reverberação por Inteligência Artificial [4]

Efeitos de Áudio Digital são transformações em um ou vários sinais de áudio, em que estas transformações dependem de conjuntos de parâmetros de controle. Geralmente, seus usuários utilizam da automação para dinamizar estes parâmetros de controle e dar mais movimento a seus áudios. Eles definem características específicas de áudio à parâmetros específicos.

Por isso, em 2017, Chourdakis e Reiss observaram que, além de ser possível classificar os dados sonoros com reverb de forma supervisionada, o retorno de tal classificação são os parâmetros necessários para replicar o efeito de reverb, podendo aplicá-los em outros áudios. Para que isso fosse possível, foi necessário treinar os classificadores com um grande dataset e adotar uma arquitetura de reverb (Moorer Reverberator), para que os parâmetros fossem estáticos.

Foram utilizados quatro classificadores neste trabalho:

- Gaussian Naive Bayes;
- One-vs-All Linear Support Vector Machine (SVM);
- Hidden Markov Model (HMM) Maximum-A Posteriori;
- HMM híbrido com observações adquiridas a partir de um conjunto de SVMs.

A influência deste artigo foi devido à própria questão de classificação do efeito sonoro de reverb. Os bons resultados deste trabalho mostraram uma possibilidade para aplicar também um estudo semelhante para efeitos de distorção.

2.0.4 Classificação de sinais de áudio utilizando Codificação Linear Preditiva e Random Forests [5]

Os motivos deste trabalho são o aumento da série de eventos como desmatamento e caça ilegal, invasão de reservas naturais, parques e florestas na última década. Com isso, Grama et al. apresentaram um sistema de classificação de sinais de áudio baseados em Codificação Linear Preditiva e Random Forests. O problema de datasets não-balanceados foi considerado, adotando 5 tipos de áudio representando pássaros, disparos de arma, tratores, serras elétricas e voz humana.

O alto grau de classificação de 99.25%, com baixo grau de alarme falso para todos os datasets adotados, mostra a efetividade deste modelo completo. Por isso, o estimador Random Forest também foi acrescentado na simulação deste trabalho.

2.0.5 Extração de Features [6]

Em 2017, Martinez-Murcia et al. dissertaram sobre o contexto geral referente à extração de características, categorizando tais algoritmos como de Filtragem, Wrapper ou de Abordagem Embutida (que une métodos de filtragem e wrapping), e expondo alguns dos métodos de decomposição mais utilizados, como o Principal Component Analysis (PCA) e o Linear Discriminant Analysis (LDA), entre outros métodos de decomposição.

Este trabalho também tratou de características disponíveis em materiais multimedia e suas particularidades e; é diretamente relacionado à área de Recuperação de Informações Musicais [7].

Capítulo 3

Materiais e Métodos

Antes de detalhar sobre a geração do dataset, é importante ressaltar que todo o processo de classificação foi separado em três fases: **Criação de Dataset**, **Redução de Dimensionalidade** e **Categorização de Classes e Classificação de Dataset**. As ferramentas, classificadores e efeitos de distorção que foram adotados também serão abordados.

Todo o trabalho foi feito utilizando a linguagem de programação *Python3*, utilizando o editor de texto *Jupyter Notebook*, para melhor organização.

3.1 Fase 1: Criação de Dataset

Uma vez que o objetivo é diretamente relacionado ao efeito sonoro, foi necessário criar um dataset relevante para o experimento. E, por isso, as seguintes premissas foram estabelecidas para gerá-lo:

1. *O sinal de entrada deverá ser pré-definido e simplificado* - Por causa do objetivo do trabalho, é importante que o sinal de entrada seja uma forma de onda simples, como uma senóide;

2. *Os algoritmos de distorção de áudio escolhidos deverão ser pré-estabelecidos* - Estes algoritmos de distorção deverão ser selecionados, replicados para Python3 e utilizados no sinal de entrada;

3. *Para cada amostra, os parâmetros de cada algoritmo de distorção deverão ser guardados de alguma forma* - Cada algoritmo tem seu conjunto de argumentos, ou parâmetros, que influenciam na aplicação da distorção. E, já que classificar estes parâmetros é o objetivo do trabalho, eles deverão ser guardados e categorizados de alguma forma.

Nesta fase a abordagem foi a mais direta e simples: cria o sinal senoidal e aplica a distorção, salvando os parâmetros no próprio nome do arquivo.

3.1.1 Ferramentas e Bibliotecas Utilizadas

As bibliotecas principais utilizadas foram *numpy*, para criar o sinal de entrada senoidal e; *soundfile*: para guardar o áudio .wav a partir da senóide que passou pelo processo de distorção.

3.1.2 Efeitos de Distorção

Distorção é tudo aquilo que modifica um sinal ou o seu formato durante ou antes de sua transmissão. Pode também ser descrita como informação perdida ou adicionada em relação ao sinal original. Ele pode ser dividido em duas categorias:

- *Tipo 1*: linear - no conteúdo da amostra, amplitudes alteradas. Envolve erros em amplitude (mudanças em volume) e fase. Podem ser corrigidas com processamento de sinais e;
- *Tipo 2*: não-linear - no conteúdo da amostra, frequências adicionadas. Envolve adição de novas frequências além das já existentes. Este tipo não pode ser corrigido completamente após a aplicação;

Todos os efeitos abordados neste trabalho são do tipo 2. Os seguintes efeitos foram selecionados:

Fuzz

Fuzz é um efeito de distorção baseado na função exponencial, tendo como consequência um comportamento não-linear, resultando em um impressão sonora "mais pesada e áspera". O algortimo utilizado para o experimento se encontra na página 125 do livro DAFX [8] e, como o algoritmo está em Matlab, foi necessário replicá-lo para Python3.

Bitcrusher

Bitcrusher é um efeito que produz uma distorção a partir da redução da resolução ou largura de banda dos dados de áudio. Por isso, é um efeito de distorção digital de baixa fidelidade. Aplicando-o, resulta em uma impressão sonora "mais quente"ou "mais áspera".

O algoritmo foi adquirido a partir do site `musicdsp.org`[9], que disponibiliza à comunidade vários algoritmos referentes a áudio. Como o algoritmo está em Matlab, foi necessário replicá-lo para Python3.

Overdrive

Overdrive é um efeito que simula um amplificador no volume máximo, distorcendo o sinal (*clipping*). É um efeito de processamento não-linear e fornece uma impressão sonora "quente e pouco suave". Ainda assim é uma das distorções mais leves. O algortimo foi adquirido na página 118 do livro DAFX [8] e, como o algoritmo está em Matlab, foi necessário replicá-lo para Python3.

Decimator

Decimator, ou downsampling, se trata de gerar uma amostra aproximada da original, dividindo a taxa de amostragem (menor resolução sonora). Esta manipulação em específico tem como consequência o chamado *aliasing*, faz com que o sinal manipulado perca informações antes existentes.

O algoritmo foi adquirido a partir do site `musicdsp.org`[9], que disponibiliza à comunidade vários algoritmos referentes a áudio. Como o algoritmo está em Matlab, foi necessário replicá-lo para Python3.

Todos os algoritmos dos efeitos de distorção demonstrados estão disponíveis no link do Github disponível no apêndice.

3.1.3 Geração do Dataset

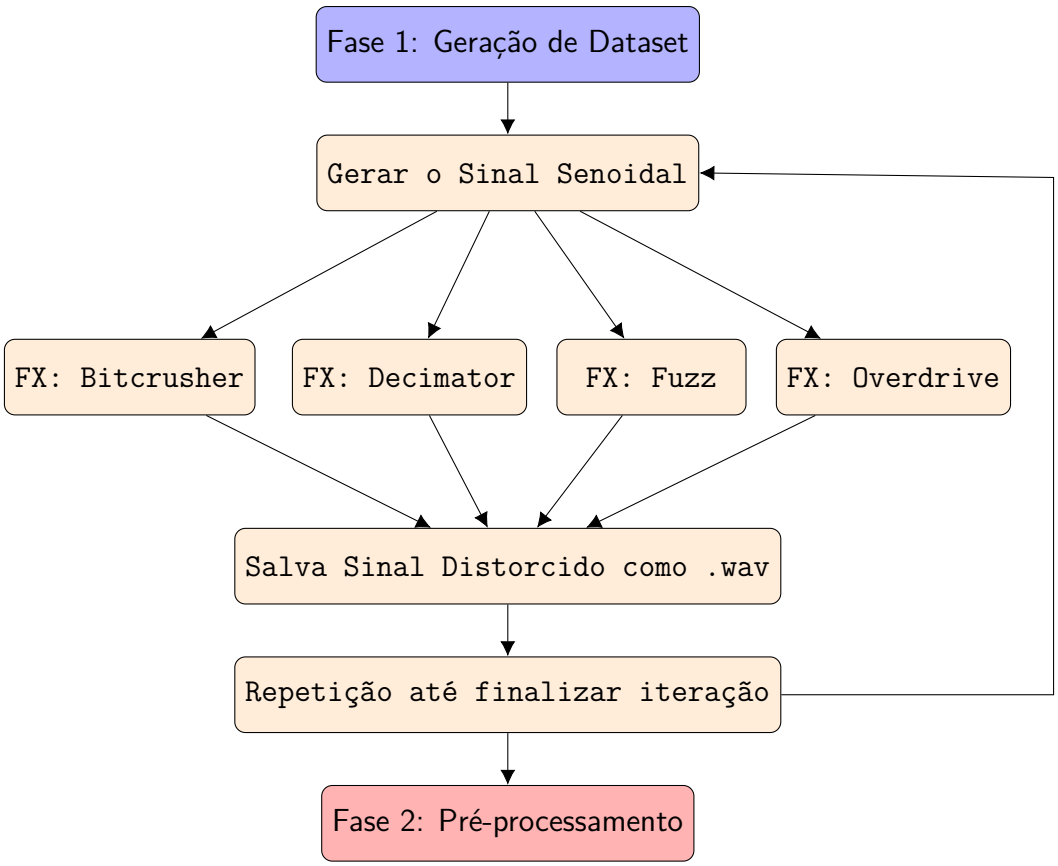

Figura 3.1: Diagrama de como o dataset foi gerado.

O diagrama 3.9 mostra como foi o processo da criação de dataset. Os seguintes passos foram necessários:

Gerar o Sinal Senoidal de entrada

O sinal de entrada foi criado de uma forma bem simples: a taxa de amostragem *fs* é o número de amostras que serão criadas no espaço linear declarado na variável *t*. A frequência foi definida arbitrariamente como 440Hz. E então, a função *sin* foi usada para gerar o sinal senoidal.

Aplicando os Efeitos de Distorção

Os algoritmos de cada efeito demonstrado na seção anterior foram aplicados considerando cada parâmetro em cada um deles. Como era necessário incrementar os valores decimalmente, foi necessário implementar a função *drange2*, que foi aplicada nos loops de *for*. A finalidade disso foi para facilitar a automação e gerar o dataset. Para o caso dos parâme-

tros *bit depth* e *gain*, a função *drange2(start, stop, step)* não foi necessária. Isto porque o tipo de dado deles é o inteiro.

Os valores de início e fim são definidos previamente para cada loop de *for*. Isto foi feito tendo em mente um sinal resultante mais discrepante em relação ao sinal de entrada.

Guardando cada dado como arquivo .wav

E, a função *write* da biblioteca *soundfile* foi utilizada para salvar como arquivo .wav cada forma de onda alterada pelo efeito de distorção. A taxa de amostragem selecionada foi de 44.1 KHz, com profundidade de bit de 24 com um encoder de modulação por código de pulso (PCM).

```
1 sf.write(file_name, sig_out, 44100, 'PCM_24')
```

Esta linha de código aparece após cada aplicação de efeito, como mostrado acima. Os parâmetros para esta função são os seguintes:

- **file_name**: Nome do arquivo que guardará o sinal gerado. Para cada amostra criada, tem-se um padrão para guardar cada um dos valores dos parâmetros que foram utilizados para gerar este sinal resultante nos próprios nomes de arquivo .wav. Observa-se abaixo que:

Fuzz: "fuzz___gain_<valor de gain>.wav"
Bitcrusher: "bitcrusher___bitdepth_<valor de bitdepth>_rate_<valor de rate>.wav"
Overdrive: "overdrive___threshold_<valor de threshold>.wav"
Decimator: "decimator___bitdepth_<valor de bitdepth>_rate_<valor de rate>.wav"

  Em suma, o padrão é o seguinte: ***<efeito>___<nome do parâmetro 1>_<valor do parâmetro 1>_<nome do parâmetro 2>_<valor do parâmetro 2>_...***;

- **sig_out**: Sinal resultante da aplicação do efeito de distorção na senóide;
- **44100**: Taxa de amostragem desejada para o arquivo .wav;
- **'PCM_24'**: salva o arquivo .wav de profundidade de bit de 24 bits para cada frame da amostra.

E cada arquivo é guardado em seu respectivo diretório, de acordo com o efeito que foi aplicado, como demonstrado na figura abaixo.

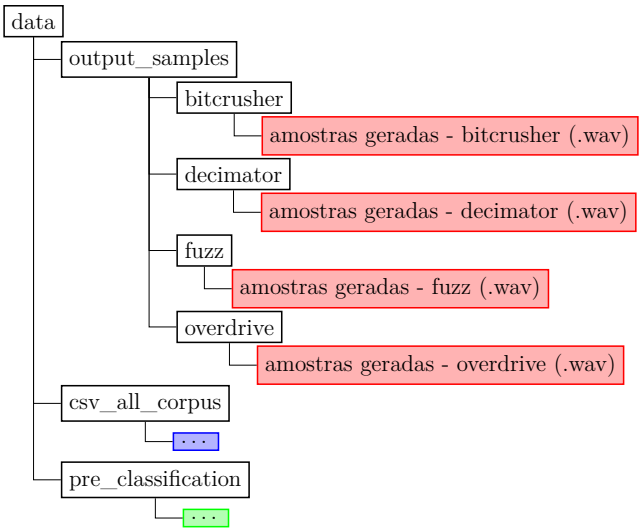

Figura 3.2: Organização de diretório dos dados gerados e utilizados.

3.2 Fase 2: Pré-processamento - Redução de Dimensionalidade e Categorização de Classes

A etapa de pré-processamento consiste na preparação dos dados disponíveis para adequá-los ao formato pedido pelos redutores de dimensionalidade e classificadores. Por isso, técnicas de redução de dimensionalidade foram utilizadas para aprofundar mais o experimento. E também, para evitar problemas na fase de classificação, foi necessário categorizar as classes apresentadas nos parâmetros.

3.2.1 Ferramentas e Bibliotecas Utilizadas

Nesta fase, as bibliotecas utilizadas foram:

- **Pré-processamento**:
  - ***os, os.path, Path***: para tratar casos de diretório nas funções;
  - ***libROSA***: para capturar as classes que serão utilizados neste trabalho;
  - ***pandas***: para gerenciar os dados, isto é, salvar/carregar o dataset convertido para corpus e agilizar o processamento dos dados.
- **Redução de Dimensionalidade**: da biblioteca scikit-learn (***sklearn***),

- ***MinMaxScaler***: para normalização do corpus. Este passo é necessário para adequar corpus para a redução da dimensionalidade do dataset;
- ***t-SNE, PCA***: técnicas de redução de dimensionalidade do corpus.

3.2.2 Características Selecionadas

A biblioteca libROSA tem capacidade de fornecer várias características a partir de um arquivo de áudio, de natureza temporal ou espectral. Delas, foram selecionadas três aspectos espectrais:

- **Roll-off Espectral**: é a frequência mais abaixo do intervalo no qual se passa pelo menos 85% da distribuição da magnitude. Isto é computado para cada frame. Roll-off é a inclinação de uma função de transferência em relação à uma frequência.
- **Largura de Banda Espectral**: é o intervalo de banda no qual a amplitude espectral irradiada não é menos que a metade do seu valor máximo. Quanto maior o intervalo, mais ruído a amostra de áudio tem.
- **Coeficientes Cepstrais de Mel-Frequência[10, 11]**: são coeficientes criados para representar sucintamente a periodicidade do espectro de um sinal de áudio. Esta feature geralmente é utilizada na área de reconhecimento de voz como, por exemplo, conseguir representar a textura vocal de uma pessoa.

Estas características foram selecionadas porque o conjunto delas representam bem a textura e comportamento do sinal distorcido. Assim, espera-se diferenciar os valores aplicados nos parâmetros que são buscados na classificação.

3.2.3 Redutores de Dimensionalidade

Para finalidade de comparação, dois métodos de redução de dimensionalidade foram inseridos ao código do experimento.

PCA: Principal Component Analysis - Análise de Componente Principal [12]

Uma das formas mais comuns de diminuir o tempo de execução de um algoritmo de aprendizado de máquina é usando o método PCA, ou Análise de Componente Principal. É um método no qual se extraem variáveis importantes (chamados de componentes) a partir de um grande set de dados em um dataset utilizando transformação linear. Com ele, é interessante encontrar componentes que maximizam a variância no dataset. Ou seja, via PCA, um dataset completo é projetado em um subespaço diferente. Para rápido referenciamento, seguem alguns pontos-chave importantes antes de seguirmos:

- Um componente é a combinação linear das variáveis originais.
- Componentes principais são extraídas de tal forma que o primeiro componente principal explicita a máxima variância no dataset. O segundo componente principal tenta expor o restante da variância no dataset e não é correlacionado com o primeiro componente. O terceiro componente tenta expor a variância que não é exposta pelos dois primeiros componentes, etc.

Este algoritmo de redução de dimensionalidade tem 7 passos:

1. Selecionar o dataset completo, e desconsiderar as labels das características;
2. Computar o vetor médio d-dimensional;
3. Calcular a matriz de dispersão e a matriz de covariância;
4. Computar autovetores e autovalores correspondentes;
5. Ordenar os autovetores diminuindo o valor dos autovalores;
6. Escolher k autovetores com os maiores autovalores;
7. Transformar o dataset no novo subespaço.

É importante ressaltar que, como o PCA é um algoritmo linear, ele não vai ser capaz de interpretar relações polinomiais complexas.

t-SNE: t-distributed Stochastic Neighbor Embedding - Incorporação de Vizinhos Estocásticos t-distribuídos [13]

Já o t-SNE é um algoritmo de redução de dimensionalidade não-linear que encontra padrões no dataset a partir da identificação de agrupamentos (clusteres) observados baseados na similaridade de pontos dos dados com múltiplas características.

Como este procedimento é feito?

Este algoritmo de redução de dimensionalidade tem 4 passos:

1. Converter as distâncias euclideanas de alta-dimensão entre os pontos para probabilidades condicionais que representam similaridades. A similiaridade do ponto x_i ao ponto x_j é a probabilidade condicional. Para pontos próximos, a probabilidade condicional é relativamente alta.
2. Para os pontos homólogos de baixa dimensão y_i e y_j dos pontos de alta dimensão x_i e x_j, é possível computar um probabilidade condicional similar. Em termos simplificados, os passos 1 e 2 são para calcular as probabilidades condicionais em alta e baixa dimensão.

3. Medir a minimização das somas das diferenças das probabilidades condicionais.
4. Computar as variâncias σ_i da distribuição de t-Student centralizado em cada ponto de alta dimensão x_i.

É importante lembrar que, mesmo que t-SNE identifique agrupamentos, ele não é um algoritmo para clustering, mas sim para redução de dimensionalidade. Isso porque ele mapeia o dado multi-dimensional em um espaço de menor dimensão; isso faz com que as características de entrada não sejam mais identificáveis. Por isso, não é possível fazer inferência baseado somente na saída do t-SNE.

3.2.4 Procedimentos da Fase

Nesta fase segue-se o diagrama abaixo:

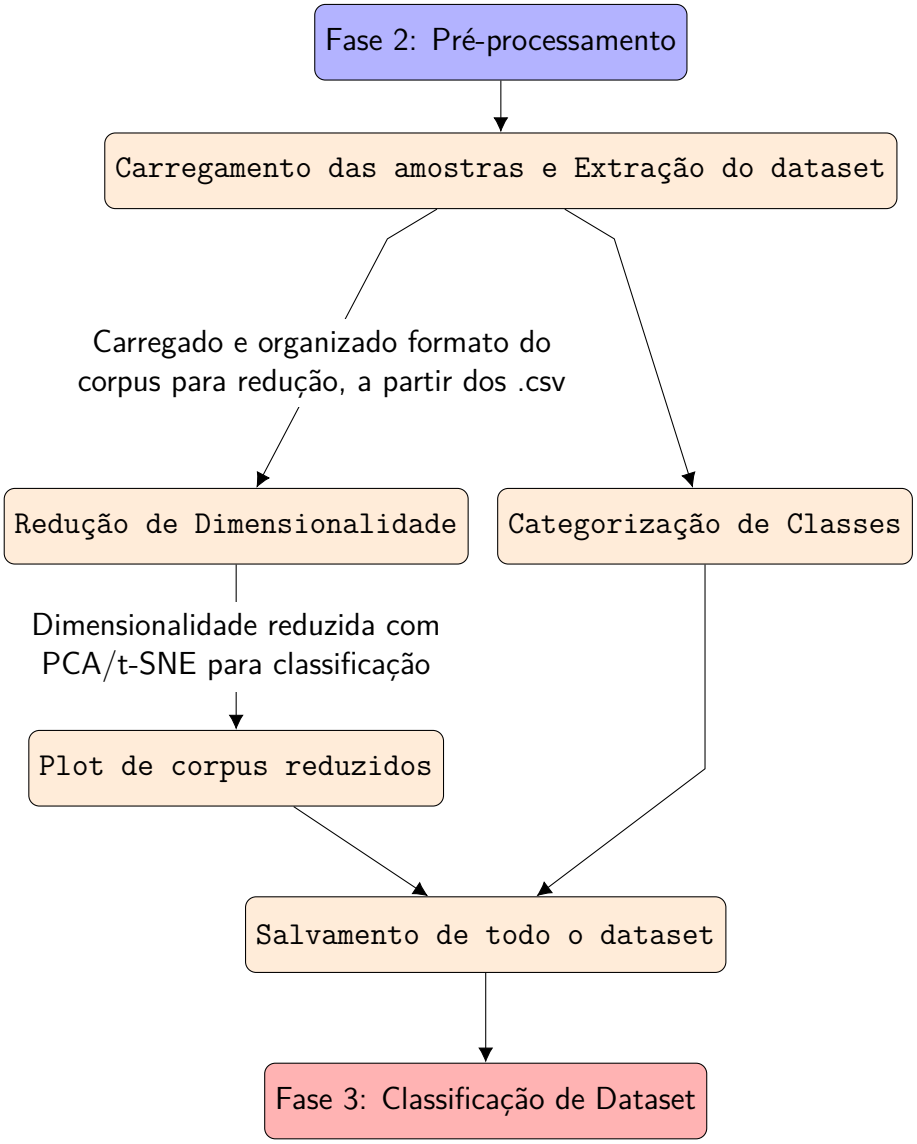

Figura 3.3: Diagrama de pré-processamento - Redução de Dimensionalidade e Categorização de Classes

Carregamento das amostras e Extração do dataset

Para carregar e extrair o dataset, algumas funções precisaram ser criadas previamente

- **get_target_params_from_filename**

  Esta função tem como entrada uma string e retorna os parâmetros utilizados no algoritmo de efeito de distorção para uma amostra, ou seja, cada parâmetro extraído da string é o que será utilizado como classe.

- **get_features_from_wav**

  Esta é uma das principais funções do pré-processador. Ela tem 3 argumentos de entrada, que são strings: **folderpath**, que é o diretório onde estão as pastas com os arquivos .wav; **folder**, que é a pasta selecionada para a execução da função e; **file_name**, que é o arquivo selecionado. E, como saída, ela retorna um array com a média e o desvio-padrão para o Roll-off Espectral, a Largura de Banda Espectral e 13 MFCCs (Coeficientes Cepstrais de Mel-Frequência), 30 características no total. É importante ressaltar que, ao calcular a média e desvio-padrão, as informações amostra-a-amostra são perdidas.

- **load_from_preselected_folders e load_from_all_folders** Esta é a principal função do Pré-processador. Ela utiliza as duas funções acima para adquirir cada amostra que compõe o array, e as classes de cada amostra, que é adicionada no array *target_parameters_all*. Ela tem 2 argumentos de entrada: uma string **folderpath**, que é o diretório onde estão as pastas com os arquivos .wav e; uma lista de strings **folders**, que é a lista de pastas que contêm estes arquivos. E, como saída, ela retorna um dicionário com todas as informações referentes a um efeito ou conjunto de efeitos pré-selecionados no segundo argumento *folders*.

Com isso, 16 conjuntos de dados diferentes foram gerados. Para cada efeito, quatro datasets: **data**, que são as características espectrais extraídas de cada amostra gerada; **target_fx_names** e **target_fx**, que são respectivamente o nome aplicado de efeito de distorção e um número identificador correspondente e; **target_params**, que são os parâmetros para cada amostra específica que queremos utilizar como classe.. Dos 16 arrays, os que foram usados são os que correspondem à **data** e a **target_params**, que são 8 deles. E todos eles foram guardados em formato de arquivo .csv, de acordo com a figura 3.4. Isto foi feito para que toda a função de extração das características espectrais não precise ser executada toda vez.

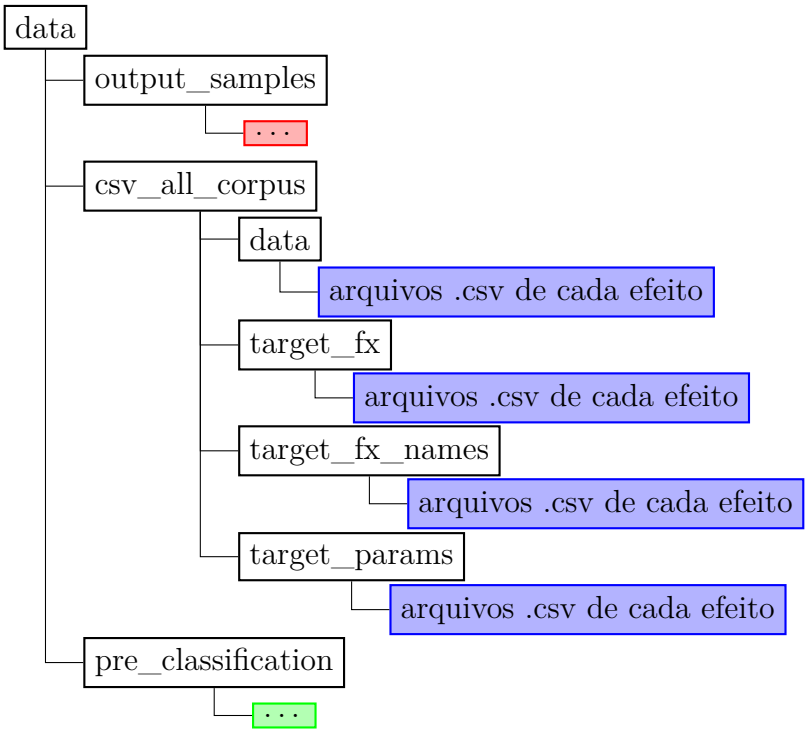

Figura 3.4: Organização de diretório dos datasets gerados e guardados em arquivos .csv.

A média e o desvio-padrão de todas as características das amostras que compõem os quatro corpus serão gerados e separados de acordo com o padrão de classificação do modelo selecionado. Principalmente os dados na pasta *data* e na pasta *target_params* serão utilizados, como já foi dito. Para finalidade de agilizar processamento, foi necessário utilizar a biblioteca *pandas* para salvar e carregar os corpus já computados. E, para cada efeito, tem-se a mesma lógica de código. Caso o corpus ainda não esteja computado, a função *load_from_preselected_folders* é chamada e o que foi gerado é salvo no diretório correspondente à cada palavra-chave do dicionário.

Redução de Dimensionalidade e Plot de Corpus Reduzidos

Esta etapa consiste somente em normalizar os dados por coluna para ter média zero e desvio-padrão um, e aplicar as técnicas de redução de dimensionalidade nos corpus normalizados, além de imprimir na tela os gráficos correspondentes. Padronizar os corpus é um requisito necessário para que não haja discrepância no formato de dados pedidos pelas técnicas de redução de dimensionalidade PCA e t-SNE, especificados pelo sklearn. Então, os corpus normalizados são submetidos a estas técnicas, que foram definidas para reduzir os dados para 2 componentes.

Categorização de Classes

Na geração de dataset percebe-se que, como loops de *for* foram usados para incrementar decimalmente quase todos os parâmetros de efeitos de distorção, tem-se uma quantidade muito grande de classes ao fim do processamento. Por isso, a categorização simples é uma solução que garante uma separação em termos de intervalo para cada parâmetro como, por exemplo, a divisão por categorias de 10 em 10, no parãmetro *gain* ou; de 4 em 4 no parâmetro *bit depth*.

Salvamento de todo o dataset

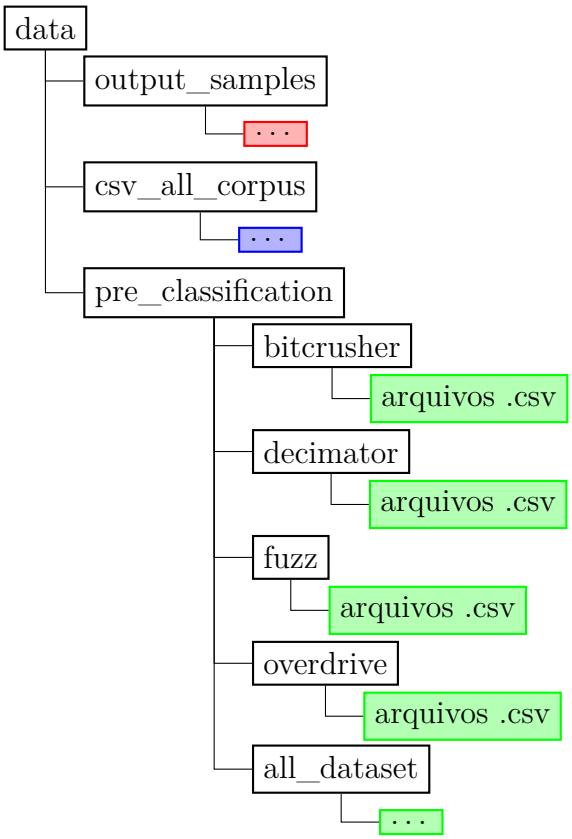

Figura 3.5: Organização de diretório dos datasets pré-processados, salvos em arquivos .csv.

Por fim, todos os dados já pré-processados são salvos, como mostrado na figura 3.5. Cada uma das subpastas com nome do efeito de distorção contém os arquivos que correspondem aos dados tratados. No caso, os arquivos .csv que posteriormente serão usados são os que têm os finais **preclass_data_no_reduction**, **preclass_data_pca**, **preclass_data_t-SNE** e **preclass_target_params**.

3.3 Fase 3: Classificação de Dataset

Foi proposto neste trabalho se é possível simular o processo de busca de parâmetros de efeitos de distorção a partir de modelos de aprendizado de máquina. Nesta fase, com os dados tratados, técnicas de classificação são aplicadas para adquirir o resultado. A finalidade é se o uso dos modelos SVM, KNN e Random Forest propostos também têm resultados satisfatórios para classificação de efeitos de distorção, utilizando os datasets com as características reduzidas por PCA e T-SNE e o dataset sem redução de dimensionalidade. E, a partir daí, é feita a avaliação dos resultados a partir dos plots das matrizes de confusão e de calor resultantes. Com isto, é possível estender o estudo para análise com modelos mais complexos e elaborados.

3.3.1 Ferramentas e Bibliotecas

Nesta fase, as bibliotecas utilizadas foram:

- **Carregamento de Dados**:
  - ***os, os.path***: para tratar casos de diretório nas funções;
  - ***pandas***: para gerenciar os dados, isto é, salvar/carregar o dataset convertido para corpus.
- **Aprendizado de Máquina**: da biblioteca scikit-learn (***sklearn***),
  - ***svm, KNeighborsClassifier, RandomForestClassifier***: Classificadores SVM, KNN e Random Forest.
  - ***StratifiedShuffleSplit, ShuffleSplit e GridSearchCV***: Para validação e classificação cruzada do corpus.

3.3.2 Classificador e a Técnica de Validação Cruzada

Quanto ao modelo de classificação, o modelo de Support Vector Machines e K-Nearest Neighbors foram usados. Isto foi decidido por influência da literatura do artigo publicado relacionada à área referente à classificação de gêneros musicais [2] e de parâmetros de efeito de reverb [4]. Também estes resultados foram comparados com o do modelo Random Forest. Então, foi possível analisar e comparar o comportamento de modelos mais simples considerando especificamente parâmetros para efeitos de distorção como classes.

A Validação Cruzada de Pesquisa por Grade (Grid Search Cross Validation) com iterador por Permutação Randômica (Shuffle Split) de divisão 80/20 (80% dados de treino e 20% dados de teste), em repetição de cinco vezes, foi utilizada para automar o experimento e adquirir hiperparâmetros mais específicos para cada um dos modelos.

SVM - Support Vector Machines

Se trata de um algoritmo de classificação que se apoia na intuição inicial de que, quando as classes podem ser separadas por um hiper-plano, uma análise da fronteira entre as classes pode ser efetiva para se gerar vetores de suporte que proporcionem o cálculo de tal hiperplano. A consequência disto é que, mesmo com uma grande quantidade de dados, o algoritmo poderia realizar os cálculos mais rapidamente, pois este só analisaria um sub-conjunto das observações, ou seja, só aquelas necessárias para o cálculo dos vetores de suporte.

Uma característica importante são as margens do separador. O algoritmo que inspirou o SVM, o Classificador de Margem Máxima, como o nome sugere, utilizava o raciocínio simples de que seria mais eficiente colocar seu hiperplano exatamente no meio da distância entre as duas classes. Ou seja, o mais longe da fronteira de cada uma delas.

O SVM linear estende este conceito para uma margem de tolerância, chamado parâmetro C. No caso do SVM configurado com kernels diferentes, como a de função de base radial, além do parâmetro C, existe o parâmetro gamma, que é o grau de tolerância da influência de cada uma das características no separador. Quanto mais alto o valor, mais atenuada a curva de separação fica.

Basicamente o classificador estabelece um limite em que amostras de uma classe podem aparecer do outro lado. Isso dá maior robustez ao algoritmo, que passa a conseguir classificar com boa taxa de precisão, mesmo os dados que não sejam tão uniformemente separáveis. Isto possibilita aplicações eficientes para uma maior quantidade de datasets, mesmo aqueles muito grandes e não tão bem distribuídos.

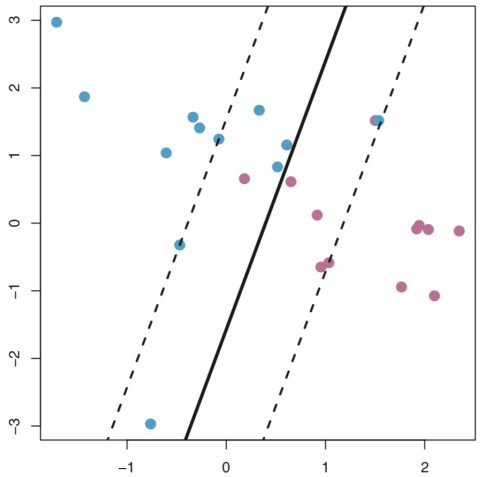

Figura 3.6: Funcionamento do parâmetro de tolerância entre a fronteira do separador e a característica mais próxima - o parâmetro C.

KNN - K-Nearest Neighbors

O algoritmo K-Nearest Neighbors [14, 15] é um modelo de aprendizado competitivo que se baseia no contexto de similaridade entre os dados submetidos, que estão espalhados em um plano n-dimensional, com n sendo o número de características adotadas para os corpus. Esta similaridade é calculada considerando uma *métrica* de distância e a escolha de um número k de vizinhos mais próximos a um dado. Os elementos do modelo influenciam a saída do sistema quando à medida de similaridade é calculada para cada novo dado de entrada. Por isso, o KNN é um modelo de aprendizado competitivo. Na perspectiva de classificação, a saída gerada pelo KNN consiste em tomar um voto majoritário dos k vizinhos mais próximos. Então, quanto maior o valor de k, maior a influência entre eles no resultado e menor o f1-score final, especialmente no caso de classificação para várias classes.

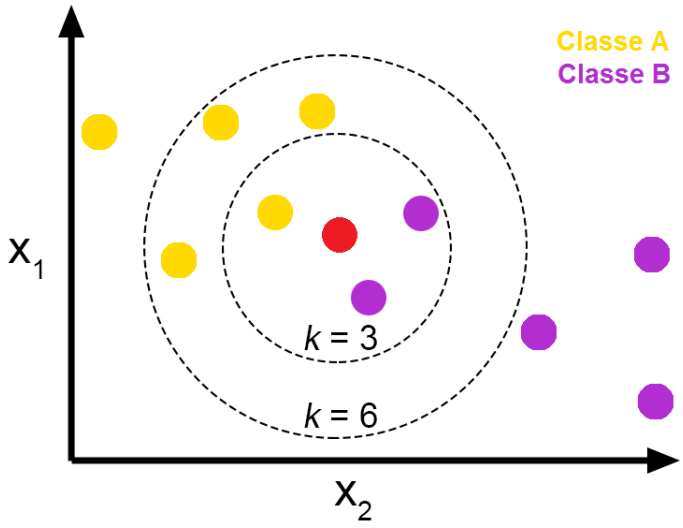

Figura 3.7: Representação do modelo KNN em um plano. O ponto rosa pode pertencer à classe B com k=3, ou à classe A com k-6.

O KNN funciona da seguinte forma:

1. Recebe um dado não classificado;
2. Mede a distância (por métrica Euclideana, Manhattan, Minkowski ou ponderada) do dado selecionado em relacão a todos os outros dados já classificados;
3. Computa o resultado do passo anterior e obtém as k menores distâncias;
4. Verifica e conta a quantidade em que aparece cada classe, de cada um dos dados que tiveram a menor distância e toma como resultado a classe que mais apareceu;
5. Classifica o novo dado com a classe tomada como resultado da classificação.

Random Forest

Antes de falar de Random Forest, é importante mencionar sobre bagging, boosting e árvore de decisão. Bagging é o algoritmo que permite o ajuste de modelos com diferentes subconjuntos e a combinação das predições deles. E boosting refere-se ao algoritmo que treina um conjunto de modelos de forma sequencial. E cada modelo aprende com os erros feitos pelo modelo anterior. O processo de bagging visa reduzir a complexidade de modelos que sobreajustam os dados de treinamento (overfitting). Já boosting é uma abordagem que visa aumentar a complexidade dos modelos que sofrem com uma alta tendência, isto é, que subajustam os dados de treinamento (underfitting).

Já Árvore de Decisão é um mapa dos possíveis resultados de uma série de escolhas relacionadas. Ela geralmente começa com um único nó, que se divide em possíveis resultados, e pode ser usada para mapear um algoritmo que prevê matematicamente a melhor escolha.

Random Forest é um método de aprendizagem supervisionada em conjunto, baseado no método de bagging, que consiste em criar uma combinação randômica de árvores de decisão não-correlacionadas a partir de subconjuntos dos dados de treinamento, e treinar combinando-as para obter uma predição acurada e estável. E este modelo pode ser utilizado tanto para classificação quanto para regressão.

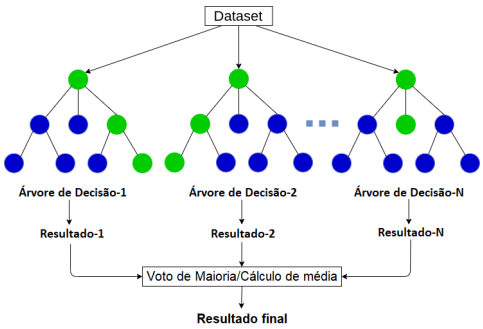

Figura 3.8: Representação do modelo Random Forest. Observe que cada árvore decide seu próprio resultado. Logo depois o resultado final é determinado como voto de maioria.

O Random Forest consiste nos seguintes passos[16]:

1. O modelo seleciona n subconjuntos a partir dos dados separados para treinamento;
2. Faz-se o treinamento de n árvores de decisão, onde:
    - Um subconjunto randômico é usado para treinar uma árvore de decisão.
    - as divisões ótimas dos dados para cada árvore de decisão são baseadas em um subconjunto aleatório de características. Por exemplo, de 10 características, 5 são selecionadas aleatoriamente para a divisão;

3. Cada árvore de decisão prediz independentemente as classes com o conjunto de dados para teste;

4. Faz a predição final, por "voto de maioria".

Pesquisa em Grade, com Validação Cruzada por Permutação Randômica

Para explicar sobre Pesquisa em Grade, é necessário falar primeiro sobre **Hiperparâmetros**. Hiperparâmetros são parâmetros de modelos preditivos. Para este trabalho, é importante determinar qual é a configuração mais otimizada para cada um dos modelos. Como os modelos utilizados têm vários parâmetros, alguns deles foram selecionados para este ajuste. Todos os que estão aqui abaixo estão disponíveis na biblioteca sklearn. Do modelo SVM foram:

- ***C:*** o parâmetro de tolerância entre a fronteira do separador e a característica mais próxima.
- ***Kernel:*** neste, pode-se definir o parâmetro para linear, poly, rbf, sigmoid, pré-computado ou prover o nosso próprio kernel. Para este trabalho, utilizamos os quatro primeiros somente.
- ***Degree:*** neste, um grau específico é definido para o kernel polinomial.
- ***Gamma:*** é o parâmetro que define o grau de tolerância da influência de cada uma das características no separador.
- ***Max_Iter:*** é o número máximo de iterações para o solucionador.

Já do modelo KNN foram:

- ***n_neighbors:*** é o número de vizinhos para usar como padrão, para cada iteração de K-Neighbors.
- ***weights:*** é a função de peso usada na predição. Ela pode ser uma função de peso uniforme ou de peso influenciado pela distância dos vizinhos do dado (quanto mais perto do dado, mais influência no dado).
- ***algorithm:*** é o algoritmo utilizado para computar os vizinhos mais próximos. Os algoritmos Kd-Tree, Ball-Tree e de força bruta foram utilizados para este trabalho.
- ***metric:*** é a métrica de distância utilizada para o KNN. Ela pode ser Euclideana, de Manhattan, de Minkowski ou ponderada. Para este trabalho utilizamos a Euclideana e a de Manhattan.

E, do modelo Random Forest foram:

- ***n_estimators:*** é o número de árvores na floresta randômica.
- ***criterion:*** aqui é a função que mede a qualidade de um *split*. No caso, os critérios utilizados foram o da impureza de Gini e o de ganho de informação.
- ***max_features:*** é o número de características a serem consideradas ao procurar pelo melhor *split*. Para este trabalho, estes números foram a raíz quadrada e o log2 deste número de características.

E são estes alguns dos hiperparâmetros que garantem o grau de precisão quando usamos o modelo. Por isso, é aí que a Pesquisa em Grade entra.

Pesquisa em Grade é a técnica de ajuste que procura encontrar os hiperparâmetros mais otimizados para um modelo. É o processo de busca exaustiva sobre os parâmetros específicos do modelo. Ela testa todas as possibilidades previamente determinadas pelo usuário e garante uma forma mais elegante para buscar um resultado mais otimizado ao projeto. A busca exaustiva pode ser determinada tratando um ou vários hiperparâmetros de um modelo. Por exemplo, os parâmetros adotados para os modelos SVM, KNN e Random Forest foram o ***C***, o ***n_neighbors*** e ***n_estimators***, respectivamente.

Já a **Validação Cruzada por Permutação Randômica** é o processo no qual há a separação do dataset por porcentagem entre treino e teste (geralmente é 70/30 ou 80/20) de forma aleatória. Isto serve para treinar o modelo de forma mais automatizada, e também permite iteração para divisão.

Agora, considere que é possível separar parte do dataset que queremos utilizar para treino do modelo e outra para teste n vezes e selecionar o resultado com maior acurácia, *para cada* teste de hiperparâmetros específicos do modelo.

3.3.3 Procedimentos da Fase

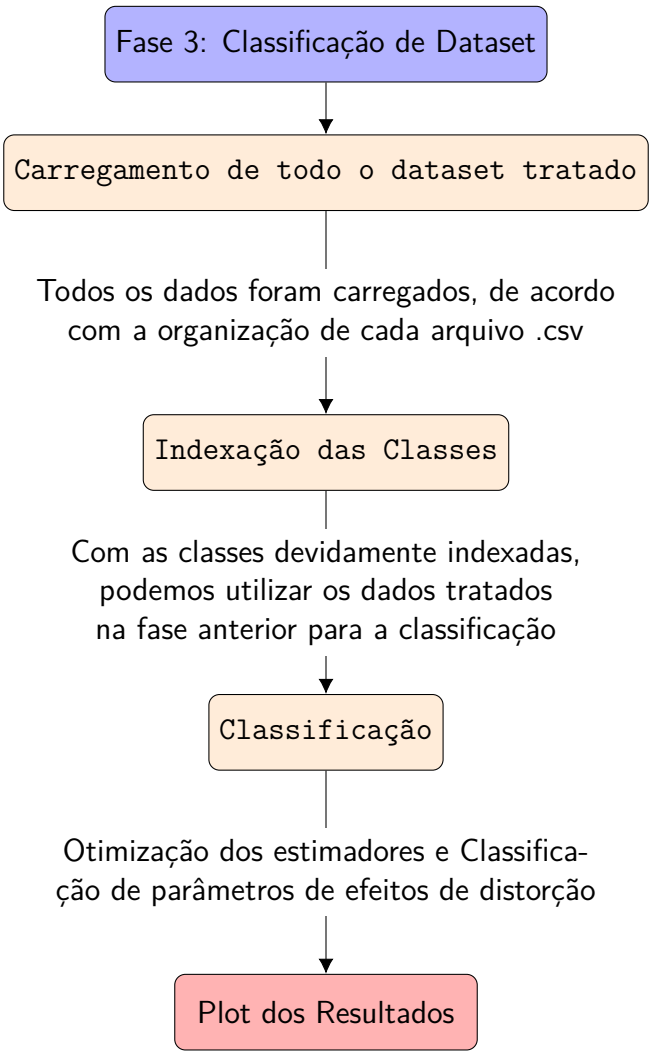

Figura 3.9: Diagrama da etapa de classificação.

Carregamento de todo o dataset tratado

Todos os dados tratados na fase anterior são carregados aqui. A biblioteca *pandas* é utilizada para extrair os dados assim como foram pré-processados. Ao fim, cada um dos dados são guardados em arrays.

Indexação das Classes

Os únicos arrays que deverão ser tratados mais uma vez serão aqueles que têm ao final "**_target_params**". Isto porque o tipo de dado deles é *string*. E como o mecanismo de classificação cruzada do corpus exige que as classes categorizadas estejam em *inteiro*, foi necessário indexar cada classe correspondente a cada parâmetro. E isto foi feito na fase

de Classificação para manusear os resultados com mais facilidade, sem sempre precisar carregar o dado e o seu índice.

Classificação

Por fim, cada classe indexada é aplicada junto a todos os corpus, seja sem redução ou reduzidos por PCA ou T-SNE, encontrando a melhor configuração para cada um dos três estimadores com validação cruzada por pesquisa em grade e iteração por divisão cruzada de 80/20 de cinco vezes. Os estimadores, com seus parâmetros, são:

- SVM. com quatro diferentes kernels (linear, de função de base radial, sigmóide e polinomial) e seus respectivos valores de gamma;
- KNN, com dois pesos (uniforme e por distância), três algoritmos para computação do vizinhos mais próximos (kd-tree, ball-tree e força bruta) e duas méticas (Euclideana e de Manhattan) e;
- Random Forest, com dois diferentes critérios (de Gini e por Entropia) e dois max_ features (raíz quadrada e logarítmica com base 2).

Após encontrar e salvar a melhor configuração para cada estimador, a classificação é feita de forma direta com esta configuração encontrada, para adquirir e salvar os plots dos resultados. Assim, descobrimos se o que foi proposto e especificado neste trabalho teve bons resultados.

Capítulo 4

Resultados e Análise

Com os métodos devidamente aplicados, é possível analisar se de fato os modelos escolhidos são abordagens interessantes para classificação de parâmetros de efeitos de distorção, especificamente. Para isso, as matrizes de confusão dos melhores casos foram extraídas para os modelos SVM, KNN e Random Forest, por cada parâmetro específico. A análise consistirá em observar os resultados de cada modelo e fazer uma comparação entre eles por parâmetro e depois por efeito de distorção.

4.1 Fuzz

Para este efeito sonoro, somente um parâmetro foi utilizado para fazer esta classificação. O parâmetro gain determina o grau de distorção aplicado pelo algoritmo no sinal, de forma direta.

4.1.1 Análise por parâmetro: gain

Este parâmetro tem o intervalo entre 1 e 100, divididos de 10 em 10 e categorizados numericamente por classes:

- Classe 0: categoria de ganho 1 até 9.99;
- Classe 1: categoria de ganho 10 até 19.99;
- Classe 2: categoria de ganho 20 até 29.99;
- Classe 3: categoria de ganho 30 até 39.99;
- Classe 4: categoria de ganho 40 até 49.99;
- Classe 5: categoria de ganho 50 até 59.99;
- Classe 6: categoria de ganho 60 até 69.99;

- Classe 7: categoria de ganho 70 até 79.99;
- Classe 8: categoria de ganho 80 até 89.99;
- Classe 9: categoria de ganho 90 até 100.

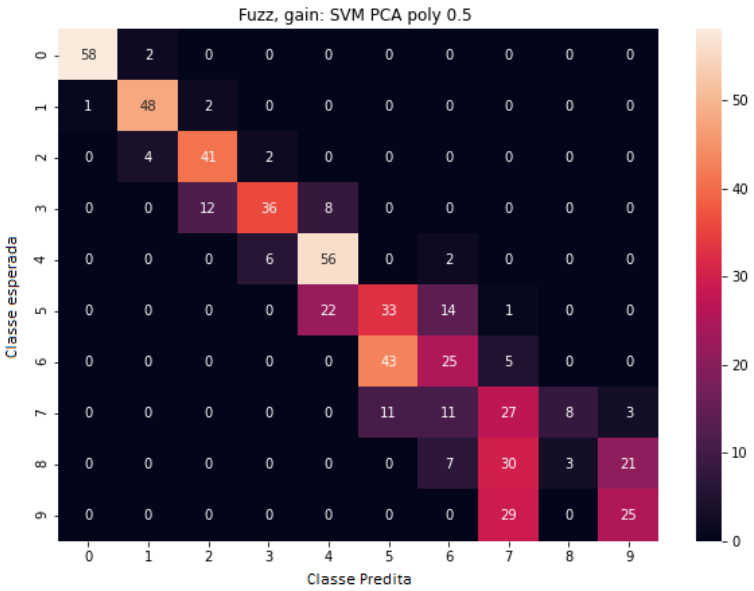

Figura 4.1: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo SVM, com corpus reduzido por PCA, kernel polinomial e gamma 0.5

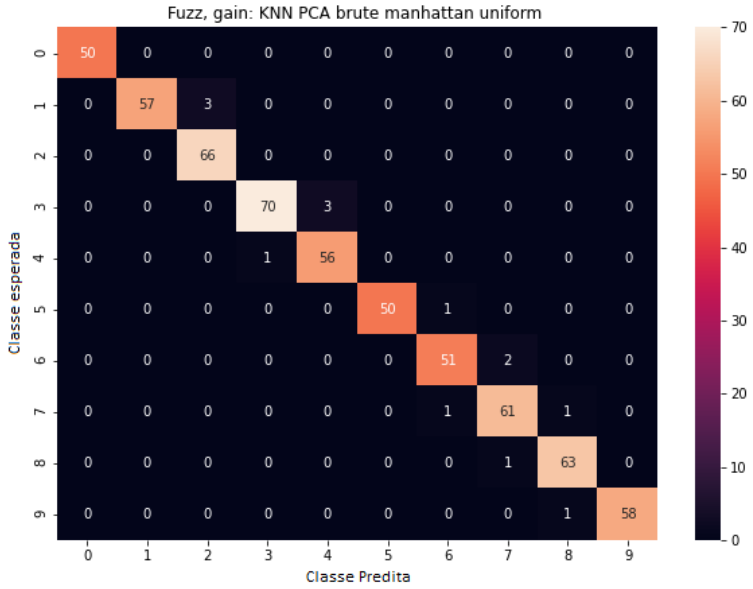

Figura 4.2: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo KNN, com corpus reduzido por PCA, algoritmo brute, métrica manhattan e peso uniforme

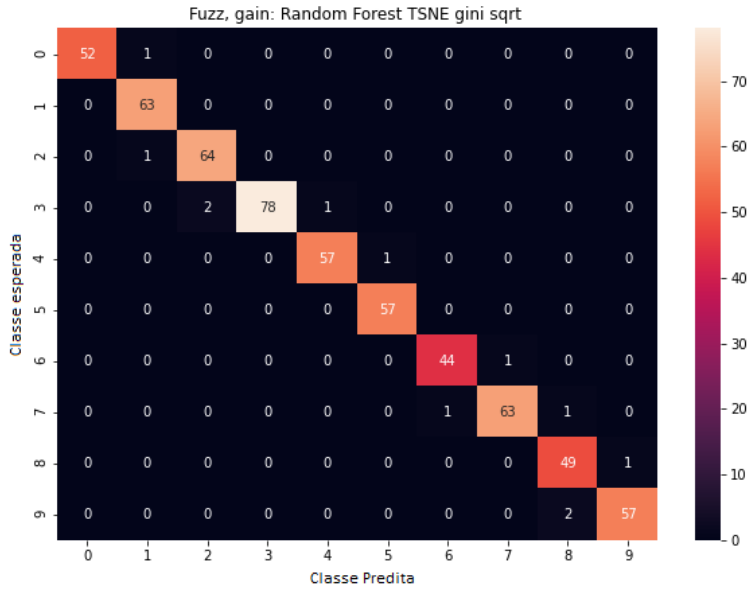

Figura 4.3: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo Random Forest, com corpus reduzido por t-SNE, critério gini e padrão de raiz quadrada

De acordo com a matriz resultante do modelo SVM na figura 4.1, pode-se perceber que o modelo não consegue discernir corretamente as classes 5 a 9, principalmente a

classe 7. Isto deve-se também ao número de componentes usados para a redução de dimensionalidade no t-SNE. Não há uma discrepância muito aparente na maioria das classes. Mas, a quantidade de erros maior que acertos nas 5 últimas classes ressalta a dificuldade deste modelo de predizer corretamente para ganhos maiores. Podemos concluir em geral que o modelo SVM não é muito apropriado para este parãmetro, já que há uma baixa acurácia na metade das classes propostas.

Na matriz resultante do modelo KNN na figura 4.2 visando o parâmetro gain, de Fuzz, um caso bem acurado e preciso é observado, para todas as classes dadas. Há um ou outro dado que não foi corretamente predito, mas, pra este caso, não influencia tanto assim, pelo fato do intervalo do erro ser menor que 19,9.

Por fim, na matriz resultante do modelo Random Forest na figura 4.3, observa-se o mesmo caso bem acurado e preciso, como no modelo anterior. E, da mesma forma, podemos ver uma ou outra discrepância mínima na predição dos dados relacionados às classes.

Fuzz: Parâmetro Gain			
Modelo	SVM	KNN	Random Forest
f1-score (macro)	65.69%	98.23%	98.27%
Tempo médio de fit (em ms)	11.8361	2.6909	3271.735

Tabela 4.1: Probabilidade de predição e tempo médio de fit (em milissegundos) para o parâmetro gain, do efeito Fuzz.

Como pode-se observar na tabela 4.1, os resultados referentes a f1-score também recomendam o Random Forest e o KNN para classificação do parãmetro gain do efeito Fuzz. O que mais vai diferenciar entre os dois é o tempo de fit médio extenso ao custo de um pouco mais de score geral. Isso ocorre por causa do custo computacional exigido pelo Random Forest. Por isso, neste caso, os dois modelos são apropriados para a estimação deste parâmetro, com preferência ao KNN.

4.1.2 Análise geral para este efeito de distorção

Como somente um parâmetro foi avaliado por causa do algoritmo selecionado na geração de dataset, o que foi analisado na seção anterior vale totalmente para esta análise geral.

4.2 Bitcrusher

No algoritmo do efeito de bitcrusher há dois parâmetros que serão avaliados: Bit depth e Rate. Bit depth é a quantidade ou profundidade de bits que estão definidos para todos os frames de cada arquivo de áudio. E rate é a taxa em que o efeito de bitcrusher é aplicado.

É importante ressaltar que, ao contrário do parâmetro rate, para o parâmetro bitdepth espera-se uma grande discrepância na predição em relação à categoria esperada, exceto para os casos de maior valor. Isto porque, quanto maior a distorção, o som fica mais próximo de um ruído.

4.2.1 Análise por parâmetro: bitdepth

O parâmetro bitdepth foi dividido em 4 categorias:

- Classe 0: categoria de bitdepth 01 até 04
- Classe 1: categoria de bitdepth 05 até 08
- Classe 2: categoria de bitdepth 09 até 12
- Classe 3: categoria de bitdepth 13 até 16

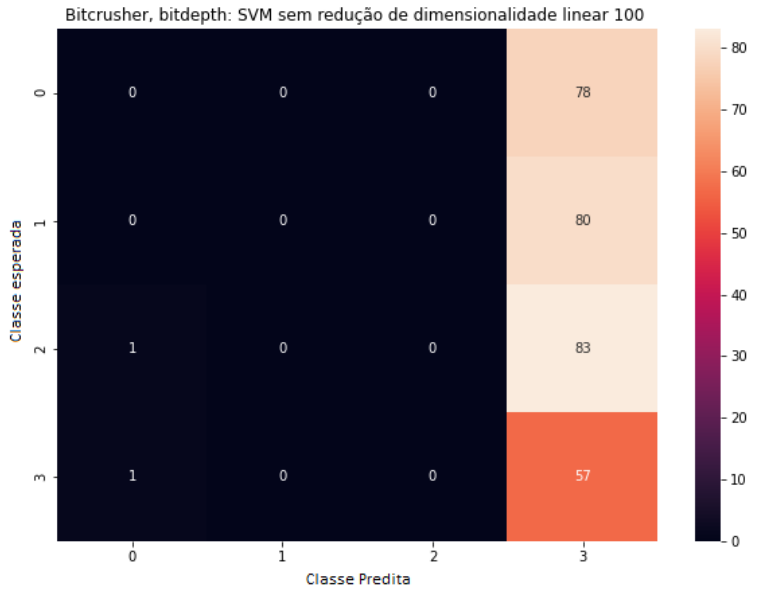

Figura 4.4: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo SVM, com corpus sem redução de dimensionalidade, kernel linear e gamma 100

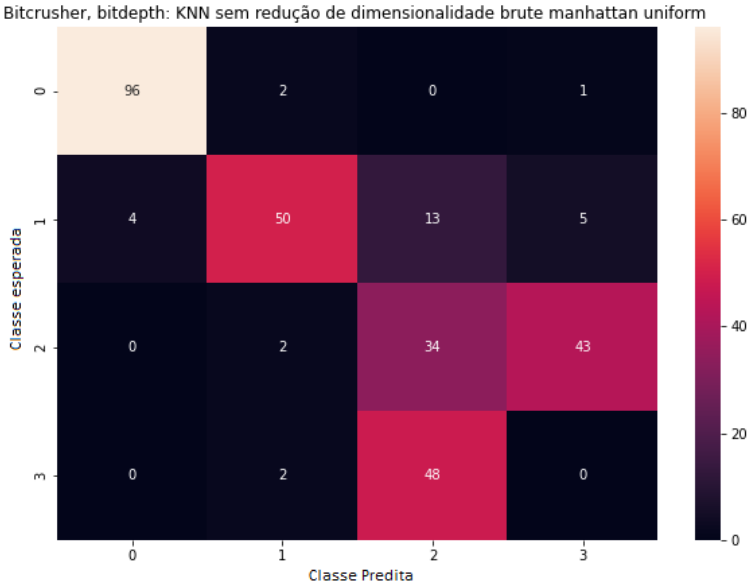

Figura 4.5: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo KNN, com corpus sem redução de dimensionalidade, algoritmo brute, métrica manhattan e peso uniforme

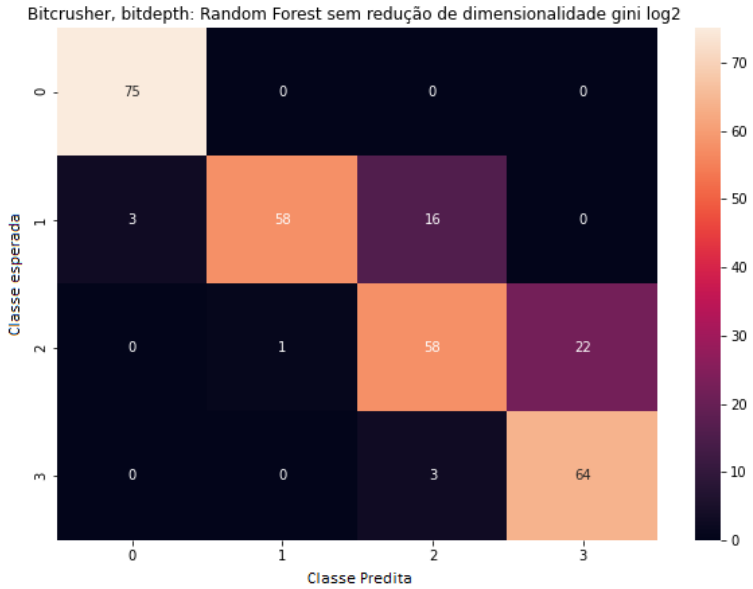

Figura 4.6: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo Random Forest, com corpus sem redução de dimensionalidade, critério gini e padrão logarítmico de base 2

Neste caso mostrado na figura 4.4, o modelo SVM é inapropriado para a predição deste parâmetro. Podemos ver que, das 4 classes, somente uma há acertos. E, da que tem

acertos, os acertos em si são menos da metade das tentativas de predição.

Já para o modelo KNN, na figura 4.5, percebem-se acertos significativos nas classes 0 e 1. Já a classe 2, há menos da metade de acerto e, da classe 3, temos ausência de acertos. Isto significa que o modelo KNN não é muito bom para o intervalo de 9 a 16 do parâmetro bitdepth.

Por fim, na figura 4.6, a matriz resultante para o modelo Random Forest mostra o melhor resultado entre os três modelos testados. As classes 0 e 1 tem um bom grau de acerto. E, as classes 2 e 3 tem maioria de acertos para cada caso. Mas, ainda assim, o modelo apresenta dificuldades em conseguir predizer as classes 2 e 3 corretamente, com erro de aproximadamente 1 a cada 4 predições.

Bitcrusher: Parâmetro Bitdepth			
Modelo	SVM	KNN	Random Forest
f1-score (macro)	46.95%	84.18%	86.58%
Tempo médio de fit (em ms)	3.4248	2.2892	1905.0227

Tabela 4.2: Probabilidade de predição e tempo médio de fit (em milissegundos) para o parâmetro bitdepth, do efeito Bitcrusher.

A tabela 4.2 acima confirma, de forma geral, os resultados analisados a partir das matrizes. Mas é importante observar que, para este parâmetro, o melhor caso não passa de 90% de no f1-score.

4.2.2 Análise por parâmetro: rate

O parâmetro rate foi dividido em 10 categorias:

- Classe 0: categoria de taxa (rate) 0.01 - 0.99
- Classe 1: categoria de taxa (rate) 0.1 - 0.199
- Classe 2: categoria de taxa (rate) 0.2 - 0.299
- Classe 3: categoria de taxa (rate) 0.3 - 0.399
- Classe 4: categoria de taxa (rate) 0.4 - 0.499
- Classe 5: categoria de taxa (rate) 0.5 - 0.599
- Classe 6: categoria de taxa (rate) 0.6 - 0.699
- Classe 7: categoria de taxa (rate) 0.7 - 0.799
- Classe 8: categoria de taxa (rate) 0.8 - 0.899
- Classe 9: categoria de taxa (rate) 0.9 - 1.0

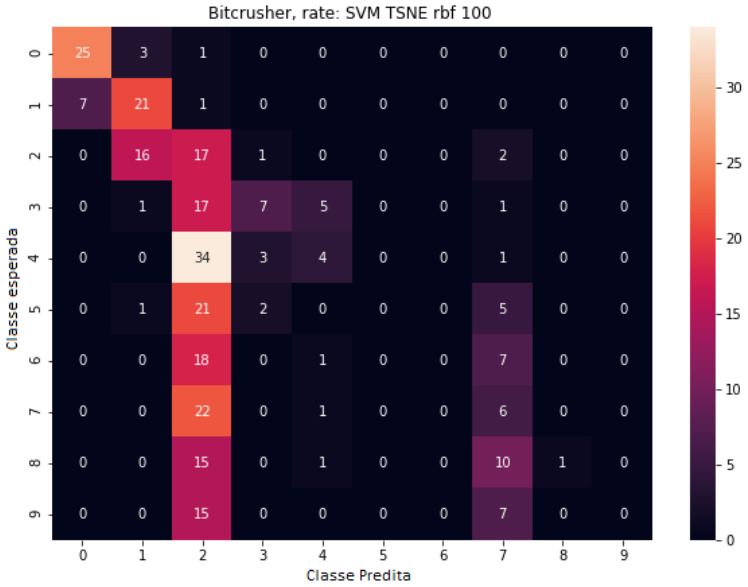

Figura 4.7: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo SVM, com corpus reduzido por t-SNE, kernel FBR e gamma 100

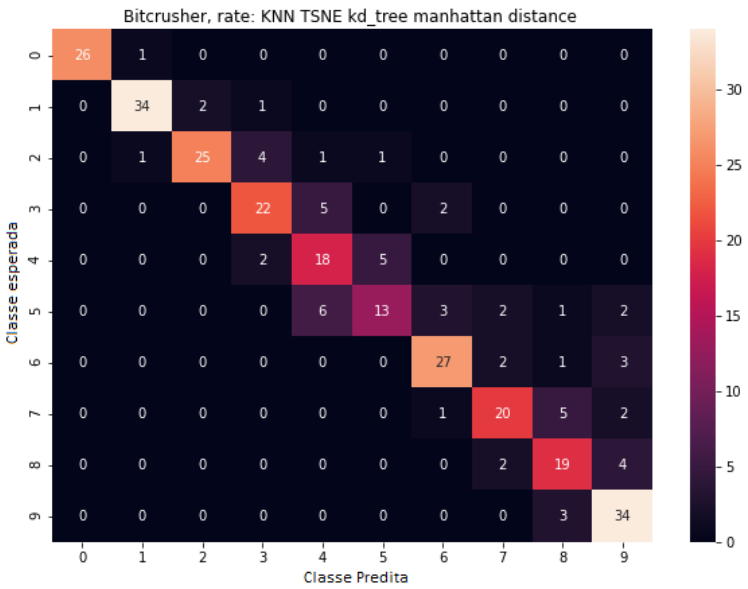

Figura 4.8: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo KNN, com corpus reduzido com t-SNE, algoritmo kd-tree, métrica manhattan e peso distance

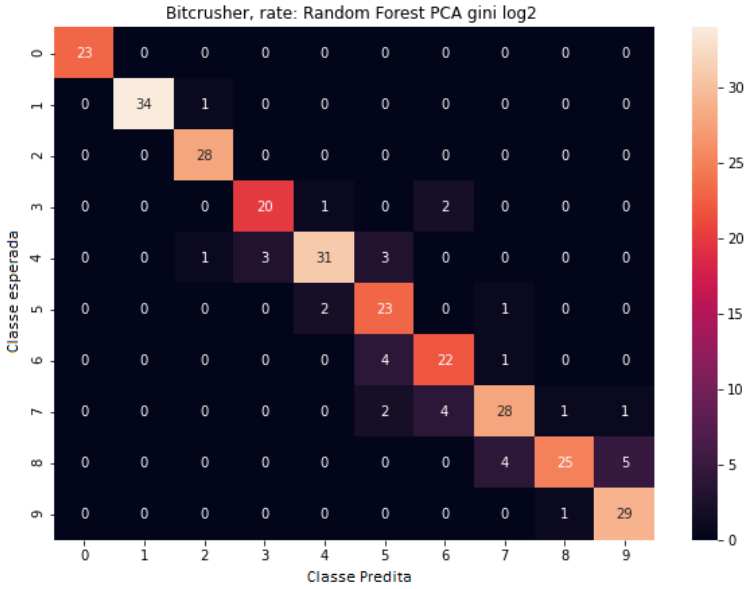

Figura 4.9: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo Random Forest, com corpus reduzido por PCA, critério gini e padrão logarítmico de base 2

Como mostrado na figura 4.7 para o modelo SVM, não há um bom resultado em geral. Em duas das classes a predição é muito espalhada e errônea, e em pelo menos três classes temos nenhum acerto na predição. Isto deve-se também ao número de componentes usados para a redução de dimensionalidade no t-SNE. O modelo SVM não é apropriado para classificar este parâmetro.

Já para o modelo KNN, mostrado na figura 4.8, tem-se um resultado muito melhor que o anterior: o mapa de calor mostra uma diagonal próxima do ideal. As únicas discrepâncias que ocorreram na predição foram no caso das classes 7 e 9, em que poucos casos de alta discrepância entre o resultado esperado e o resultado predito são observados. Isto é, um intervalo maior que 0.199 entre o valor do parâmetro que está na categoria em que se espera e o resultado retornado. Por fim, conclui-se que o modelo KNN é favorável para classificação deste parâmetro.

E, no resultado obtido usando o modelo Random Forest, como mostrado na figura 4.9, encontra-se também um mapa de calor em que se passa uma diagonal pela matriz. Aqui, comparado ao resultado do modelo KNN, há menos discrepâncias ainda. Mas, ainda assim há discrepâncias nas classes 5 e 6, mesmo que sejam 2 e 1 amostras, respectivamente.

Bitcrusher: Parâmetro Rate			
Modelo	SVM	KNN	Random Forest
f1-score (macro)	28.19%	79.87%	85.78%
Tempo médio de fit (em ms)	22.003	3.1787	2875.5375

Tabela 4.3: Probabilidade de predição e tempo médio de fit (em milissegundos) para o parâmetro rate, do efeito Bitcrusher.

A partir de comparação direta com os dados na tabela 4.3 acima, o melhor modelo é o Random Forest, seguido do KNN. Mas ainda assim, todos estes modelos não são os ótimos para este caso. Isto é evidenciado no f1-score para cada um dos modelos testados, que são menores que 85%.

4.2.3 Análise geral para este efeito de distorção

Bitcrusher: Parâmetro Bitdepth e Rate			
Modelo	SVM	KNN	Random Forest
f1-score (bitdepth)	46.95%	84.18%	86.58%
f1-score (rate)	28.19%	79.87%	85.78%
f1-score (bitdepth * rate)	13.23%	67.24%	74.27%

Tabela 4.4: Probabilidade de predição para casos bitdepth, rate e o caso conjunto dos dois parâmetros.

Para o caso do Bitcrusher, há dois parâmetros: bitdepth e rate. Estes dois têm como melhor caso o modelo Random Forest, separadamente. Por cálculo de probabilidade condicional, para o caso de o modelo classifica corretamente todos os parâmetros, o melhor modelo ainda é o Random Forest, com chance em torno de 3 em 4 predições, como mostrada na tabela 4.4 acima.

4.3 Overdrive

Para este efeito sonoro, somente um parâmetro foi utilizado para fazer esta classificação. O parâmetro threshold determina a limiar de ganho para a ação do efeito de distorção aplicado pelo algoritmo no sinal.

4.3.1 Análise por parâmetro: threshold

O parâmetro threshold foi categorizado da seguinte forma:

- Classe 0: categoria de limiar (threshold) 0 - 0.099

- Classe 1: categoria de limiar (threshold) 0.1 - 0.199
- Classe 2: categoria de limiar (threshold) 0.2 - 0.299
- Classe 3: categoria de limiar (threshold) 0.3 - 0.399
- Classe 4: categoria de limiar (threshold) 0.4 - 0.499
- Classe 5: categoria de limiar (threshold) 0.5 - 0.599
- Classe 6: categoria de limiar (threshold) 0.6 - 0.699
- Classe 7: categoria de limiar (threshold) 0.7 - 0.799
- Classe 8: categoria de limiar (threshold) 0.8 - 0.899
- Classe 9: categoria de limiar (threshold) 0.9 - 1.0

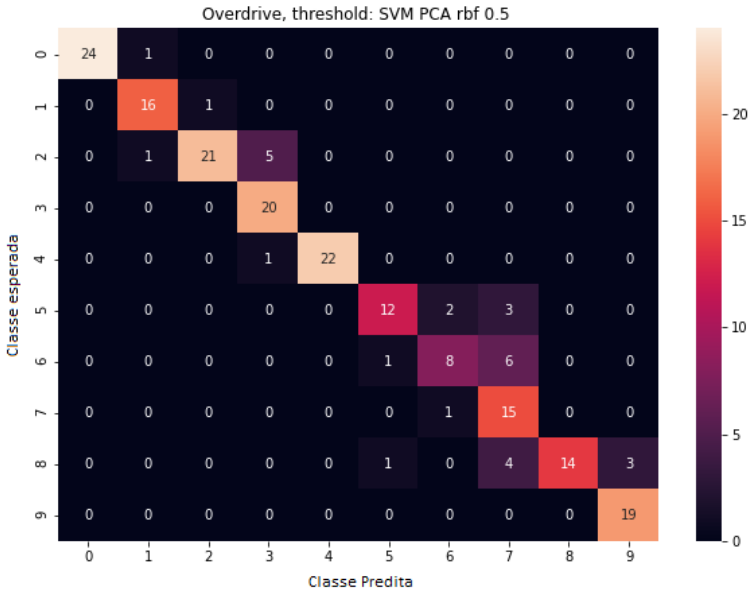

Figura 4.10: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo SVM, com corpus reduzido por PCA, kernel FBR e gamma 0.5

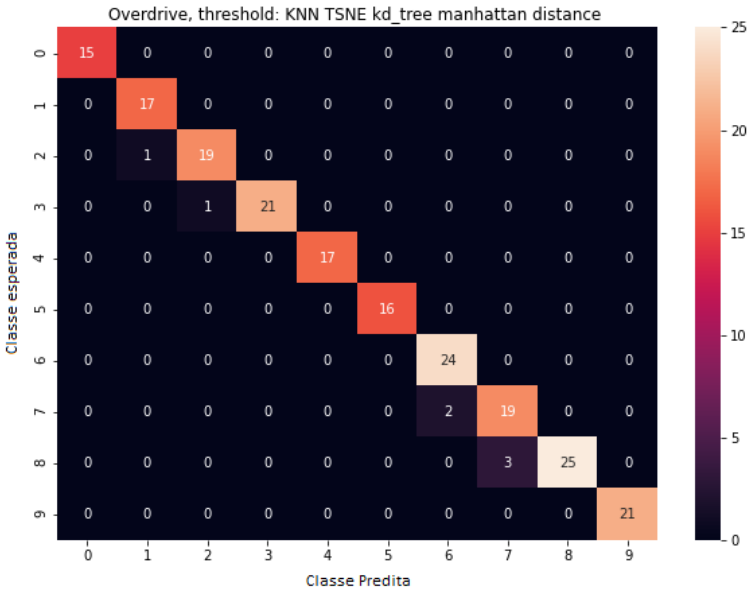

Figura 4.11: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo KNN, com corpus reduzido com t-SNE, algoritmo kd-tree, métrica manhattan e peso distance

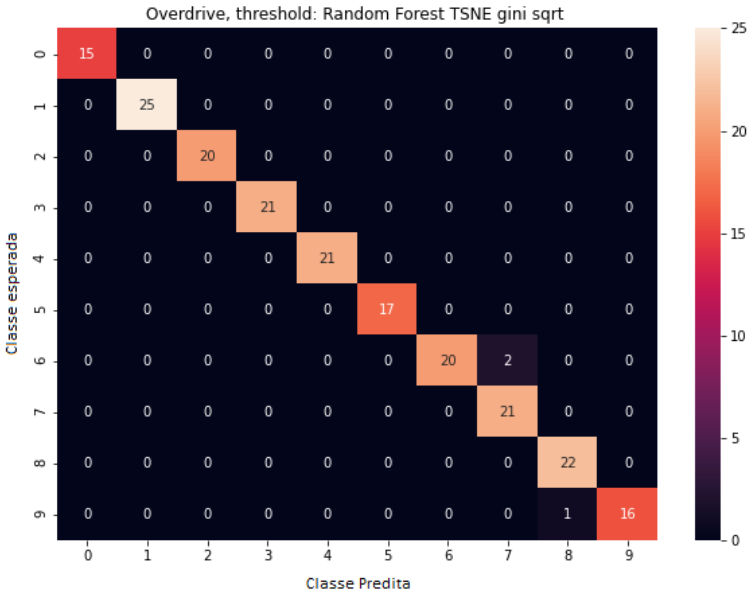

Figura 4.12: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo Random Forest, com corpus reduzido por PCA, critério gini e padrão de raíz quadrada

Para este caso mostrado na figura 4.10, o modelo SVM consegue ter predições corretas, em geral, mesmo com o número baixo de componentes usados para a redução de

dimensionalidade no PCA. A única classe onde ocorre uma discrepância maior é a 7. Para este parâmetro, o modelo SVM pode ser interessante para uso.

O caso do modelo KNN da figura 4.11 pode ser claramente utilizado. Pode-se observar uma predição muito próxima das classes esperadas.

E pela figura 4.12, o resultado de Random Forest é mais satisfatório que o do KNN. As predições se aproximam mais ainda das classes esperadas na grande maioria das amostras testadas.

Overdrive: Parâmetro Threshold			
Modelo	SVM	KNN	Random Forest
f1-score (macro)	85.30%	97.35%	99.04%
Tempo médio de fit (em ms)	7.5997	2.0483	1128.864

Tabela 4.5: Probabilidade de predição e tempo médio de fit (em milissegundos) para o parâmetro threshold, do efeito Overdrive.

De acordo com a tabela 4.5 acima, os dois melhores modelos são o Random Forest seguido do KNN, e são comparativamente efetivos para a predição deste parâmetro.

4.3.2 Análise geral para este efeito de distorção

Como somente um parâmetro foi avaliado no algoritmo selecionado, o que foi analisado na seção anterior vale totalmente para esta análise geral.

4.4 Decimator

No algoritmo do efeito de decimator há dois parâmetros que serão avaliados: Bit depth e Rate. Bit depth é a quantidade ou profundidade de bits que estão definidos para todos os frames de cada arquivo de áudio. E rate é a taxa em que o efeito de decimator é aplicado.

4.4.1 Análise por parâmetro: bitdepth

O parâmetro bitdepth foi dividido em 4 categorias:

- Classe 0: categoria de bitdepth 01 até 04
- Classe 1: categoria de bitdepth 05 até 08
- Classe 2: categoria de bitdepth 09 até 12
- Classe 3: categoria de bitdepth 13 até 16

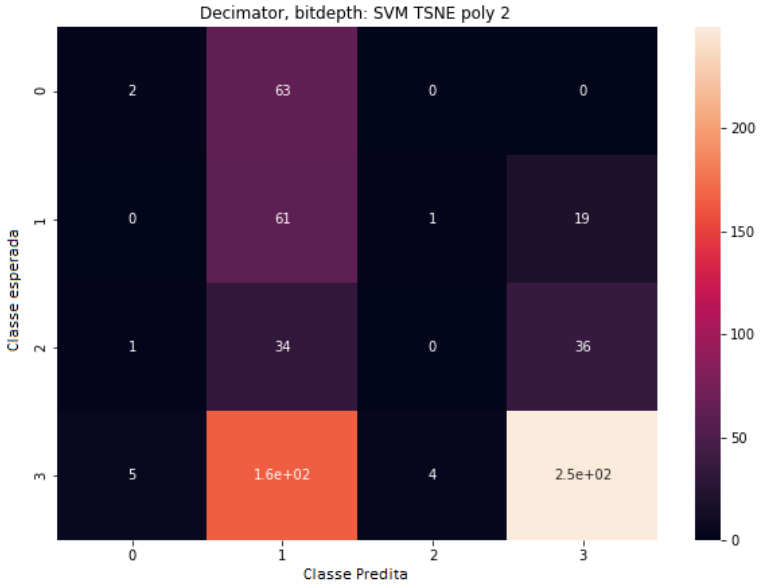

Figura 4.13: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo SVM, com corpus reduzido por t-SNE, kernel polinomial e gamma 2

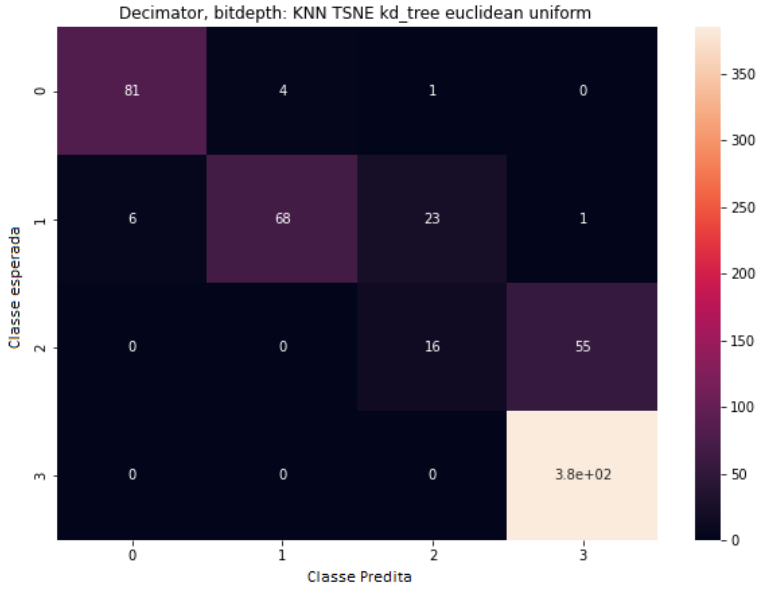

Figura 4.14: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo KNN, com corpus reduzido com t-SNE, algoritmo kd-tree, métrica euclideana e peso uniforme

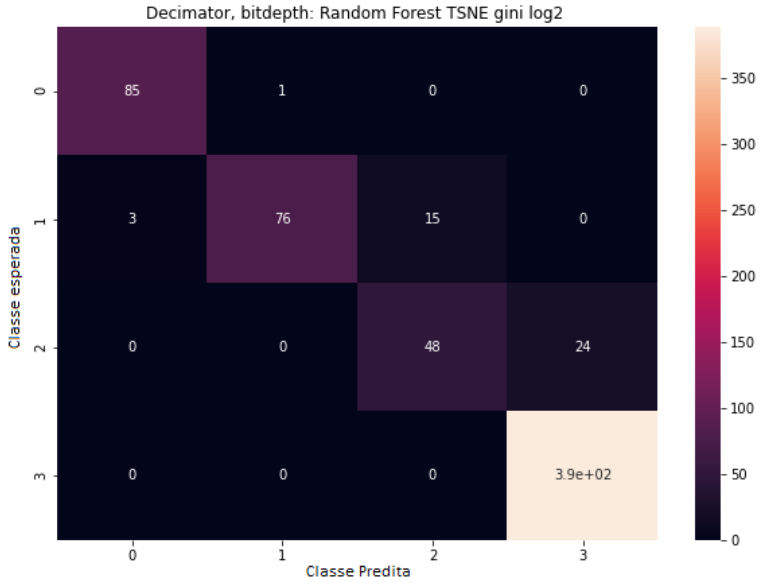

Figura 4.15: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo Random Forest, com corpus reduzido por t-SNE, critério gini e padrão de raíz logarítmica de base 2

O resultado na figura 4.13 mostra mais uma vez que, para o parâmetro bitdepth de decimator, o modelo SVM não tem boa precisão e acurácia. Há nenhum acerto para as classes 0 e 2. E dos que houveram algumas amostras em que se houve acerto, falta precisão. Isto deve-se também ao número de componentes usados para a redução de dimensionalidade no t-SNE. Por isso, o modelo SVM não é apropriado para este parâmetro.

Já o resultado na figura 4.14 do modelo KNN mostra a dificuldade de conseguir discernir corretamente as amostras de classe 2. Por isso, o modelo KNN não é muito apropriado para casos de valores de bitdepth neste intervalo.

E, no caso do resultado do Random Forest mostrado na figura 4.15, para este parâmetro, também existem discrepâncias na classe 3, mesmo que minoria. Mas, na classe 2, de 63 amostras preditas, 15 foram classificadas de forma errônea. Isto equivale a quase 1 a cada 4 predições.

Decimator: Parâmetro Bitdepth			
Modelo	SVM	KNN	Random Forest
f1-score (macro)	50.35%	88.65%	90.97%
Tempo médio de fit (em ms)	6.6655	10.2469	2679.7074

Tabela 4.6: Probabilidade de predição e tempo médio de fit (em milissegundos) para o parâmetro bitdepth, do efeito Decimator.

De acordo com a tabela 4.6, para este parâmetro, o único modelo que tem um escore acima de 90% é o Random Forest. E, na maioria dos casos, este modelo é o mais apropriado. Mas ele não é muito confiável se o intervalo para predição for entre 9 e 12 (classe 2). Ainda assim este modelo é o melhor para este intervalo específico.

4.4.2 Análise por parâmetro: rate

O parâmetro rate foi dividido em 10 categorias:

- Classe 0: categoria de taxa (rate) 0.01 - 0.99
- Classe 1: categoria de taxa (rate) 0.1 - 0.199
- Classe 2: categoria de taxa (rate) 0.2 - 0.299
- Classe 3: categoria de taxa (rate) 0.3 - 0.399
- Classe 4: categoria de taxa (rate) 0.4 - 0.499
- Classe 5: categoria de taxa (rate) 0.5 - 0.599
- Classe 6: categoria de taxa (rate) 0.6 - 0.699
- Classe 7: categoria de taxa (rate) 0.7 - 0.799
- Classe 8: categoria de taxa (rate) 0.8 - 0.899
- Classe 9: categoria de taxa (rate) 0.9 - 1.0

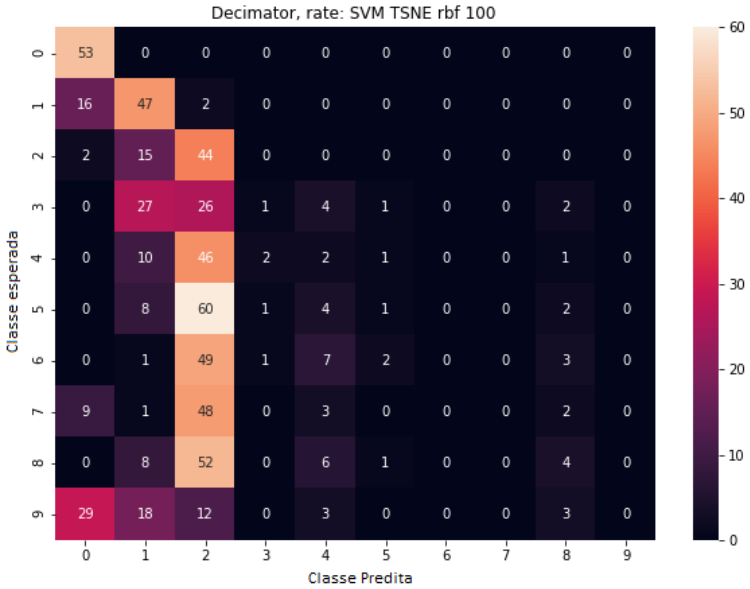

Figura 4.16: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo SVM, com corpus reduzido por t-SNE, kernel FBR e gamma 100

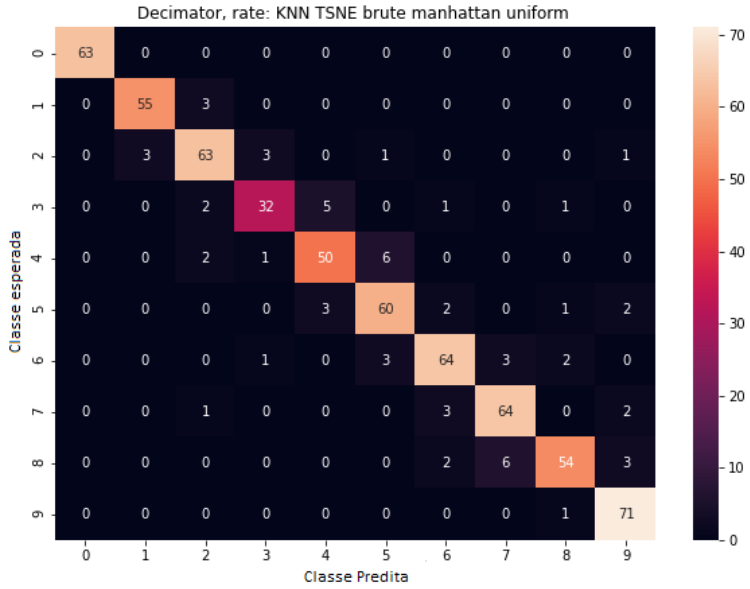

Figura 4.17: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo KNN, com corpus reduzido com t-SNE, algoritmo brute, métrica manhattan e peso uniforme

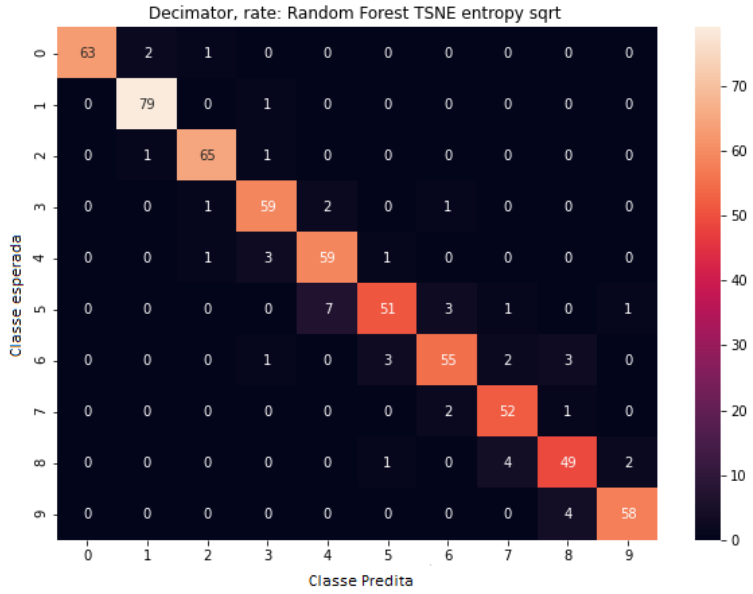

Figura 4.18: Matriz de Confusão para o melhor caso avaliado para o parâmetro em questão: modelo Random Forest, com corpus reduzido por TSNE, critério por entropia e padrão de raíz quadrada

Para este parâmetro, o modelo SVM não é apropriado, como mostrado na figura 4.16. Para todas as classes é possível observar os erros que ocorrem, seja por precisão ou por acurácia. Isto deve-se ao número de componentes usados para a redução de dimensionalidade no t-SNE.

Ao contrário do modelo SVM, o KNN tem um resultado muito mais próximo do esperado, como visto na figura 4.17. Mesmo que existam algumas amostras que foram classificadas de forma muito discrepante, os casos são poucos. Por isso, o modelo KNN é apropriado para este parâmetro.

Se o KNN é apropriado para uso, tendo rate como parâmetro, o Random Forest é mais ainda porque menos casos discrepantes são observados na figura 4.18, para predição das amostras.

Decimator: Parâmetro Rate			
Modelo	SVM	KNN	Random Forest
f1-score (macro)	31.10%	91.86%	94.47%
Tempo médio de fit (em ms)	40.8108	2.7964	7526.6629

Tabela 4.7: Probabilidade de predição e tempo médio de fit (em milissegundos) para o parâmetro rate, do efeito Decimator.

Pode-se concluir pela tabela 4.7 acima que, para o parâmetro rate, os modelos Random

Forest e KNN são os melhores, respectivamente. Isto por causa dos altos graus de acerto na predição.

4.4.3 Análise geral para este efeito de distorção

Decimator: Parâmetro Bitdepth e Rate			
Modelo	SVM	KNN	Random Forest
f1-score (bitdepth)	50.35%	88.65%	90.97%
f1-score (rate)	31.10%	91.86%	94.47%
f1-score (bitdepth * rate)	15.66%	81.43%	85.94%

Tabela 4.8: Probabilidade de predição para casos bitdepth, rate e o caso conjunto dos dois parâmetros.

Para o caso do Decimator, há dois parâmetros: bitdepth e rate. Cada um destes dois separadamente, têm o modelo Random Forest como melhor estimador. Por cálculo de probabilidade, onde o modelo classifica corretamente cada parâmetro, o melhor modelo ainda é o Random Forest, com f1-score de mais de 85%, como é observado na tabela 4.8 acima.

4.5 Análise Final

KNN e Random Forest (RF): f1-score por efeito de distorção			
Efeito de Distorção	f1-score KNN	f1-score RF	Redução do dataset do melhor caso
Fuzz (gain)	**98.23%**	**98.27%**	t-SNE
Bitcrusher (bitdepth)	84.18%	86.58%	sem redução
Bitcrusher (rate)	79.87%	85.78%	PCA
Bitcrusher (bitdepth e rate)	*67.24%*	*74.27%*	—
Overdrive (threshold)	97.35%	**99.04%**	PCA
Decimator (bitdepth)	88.65%	**90.97%**	t-SNE
Decimator (rate)	91.86%	**94.47%**	t-SNE
Decimator (bitdepth e rate)	*81.43%*	*85.94%*	*t-SNE*

Tabela 4.9: Resumo de valores de f1-score por melhores efeitos de distorção neste trabalho (modelos KNN e Random Forest). Observe que os melhores resultados se encontram no modelo Random Forest.

Com os resultados mostrados na tabela 4.9, conclui-se que o modelo SVM não é bom para classificar parâmetros de efeitos de distorção a partir das características espectrais adotadas neste trabalho, ao contrário de gêneros musicais e parâmetros para efeitos de

reverb. Isto deve-se também ao número de componentes usados para a redução de dimensionalidade nos algoritmos de redução de dimensionalidade. Por outro lado, o modelo Random Forest se saiu muito bem em todos os testes, tendo mais de 85% de f1-score, representando alta acurácia e precisão nas predições das amostras.

Mas, um possível problema dele é o tempo muito alto de fitting, devido à configuração otimizada dos seus hiperparâmetros.

Já o modelo KNN é também uma opção muito válida por ter um resultado tão bom quanto o do Random Forest. É um tempo de fitting bem mais rápido, com o custo de menor f1-score.

Capítulo 5

Conclusão e Trabalhos Futuros

A partir das literaturas estudadas, foi observado que o modelo SVM é apropriado para classificação de gêneros musicais e parâmetros de efeitos de reverb. Neste trabalho foi proposto se a premissa é a mesma para o caso de parâmetros de efeitos de distorção, comparando os resultados que o SVM trouxe com os resultados de dois outros modelos: o KNN e o Random Forest. E também esperava-se um resultado de f1-score acima de 90%, para cada caso testado.

Dentre os estimadores testados, o Random Forest foi o que teve melhor desempenho, com mais de 90% de f1-score para 4 de 6 predições de parâmetros individuais. E, desses 4 resultados de melhor desempenho, 3 foram com o dataset reduzido por t-SNE como visto na tabela 4.9, da página 48.

É interessante mencionar também que o KNN teve um resultado mais viável que o Random Forest no parâmetro gain do efeito Fuzz, por causa do tempo de fit e do custo computacional do seu algoritmo.

Ao final deste trabalho, conclui-se que um modelo como o proposto consegue buscar os parâmetros de efeitos de distorção de forma apropriada, mesmo com as limitações impostas: foi necessário criar um dataset a partir do processo de geração de sinal mais básico, que é a onda senóide, e aplicação de algoritmos de distorção simplificados. Além disso, as características espectrais foram extraídas diretamente do sinal de saída dos algoritmos de distorção. E, o resultado da extração como média e desvio-padrão fez com que as informações amostra-a-amostra sejam perdidas. Outra limitação foi a necessidade de categorizar as classes em intervalos maiores, uma vez que haviam classes demais por causa do intervalo estabelecido para cada parâmetro na fase da geração de dataset.

Portanto, é possível observar que os trabalhos futuros deverão ser mais focados no polimento das fases de geração e pré-processamento do dataset. As formas mais relevantes para se fazer isso são usar a resposta ao impulso como dado para extração das características espectrais (ao invés do sinal de saída) e estudar uma forma para explicitar a

correlação das características espectrais considerando a perspectiva de amostra-a-amostra. É interessante também detalhar seu comportamento visando estudar modelos que possam aproveitar mais deste dataset.

A partir daí, é possível estender este estudo procurando classificar estes efeitos de distorção, parametrizando-os de forma dinâmica e aplicando codificação linear preditiva [5]. É possível estender para estudar outros efeitos sonoros não necessariamente não-lineares, ou até para elementos da síntese sonora, como classificação de parâmetros de síntese sonora (como a síntese FM, por exemplo) para replicação de timbres a partir de áudio digital.

Referências

[1] Davies, Hugh: *A history of sampling.* Organised Sound, 1:3–11, 1996. vii, 1, 3

[2] Li Guo, Zhiwei Gu, Tianchi Liu: *Music genre classification via machine learning.* 2017. `http://cs229.stanford.edu/proj2017/final-reports/5244969.pdf`. vii, 6, 22

[3] Chang-Hsing Lee, Jau-Ling Shih, Kun Ming Yu e Jung Mau Su: *Automatic music genre classification using modulation spectral contrast feature.* 2007. vii, 7

[4] Chourdakis, E. T. e J. D. Reiss: *A machine learning approach to application of intelligent artificial reverberation.* J. Audio Eng. Soc., 65:56–65, Janeiro/Fevereiro 2017. vii, 7, 22

[5] Lacrimioara Grama, Corneliu Rusu: *Audio signal classification using linear predictive coding and random forests.* páginas 1–9, 2017. vii, 8, 51

[6] Francisco J. Martinez-Murcia, Juan M. Górriz, Javier Ramírez: *Feature extraction.* Wiley Encyclopedia of Electrical and Electronics Engineering, 2017. vii, 8

[7] *Music information retrieval.* `https://musicinformationretrieval.com/index.html`. 8

[8] Zolzer, Udo: *DAFX.* John Wiley and Sons .Ltd, 2002. 10, 11

[9] MusicDSP, University of Columbia: *Musicdsp.org documentation.* `https://www.musicdsp.org/_/downloads/en/latest/pdf/`. 10, 11

[10] Mermelstein, P.: *Distance measures for speech recognition, psychological and instrumental.* Pattern Recognition and Artificial Intelligence, páginas 374–388, 1976. 15

[11] Suksri, S. e T. Yingthawornsuk: *Speech recognition using mfcc.* 2012. 15

[12] Jolliffe, Ian: *Principal component analysis 2nd ed.* Springer Verlag, New York, 2002. 15

[13] Maaten, Laurens van der e Geoffrey Hinton: *Visualizing data using t-sne.* Journal of Machine Learning Research, 9:2579–2605, 2008. 16

[14] Cover, T. e P. Hart: *Nearest neighbor pattern classification.* 13:21 – 27, 1967, ISSN 0018-9448. 24

[15] Samworth, Richard J.: *Optimal weighted nearest neighbour classifiers.* The Annals of Statistics, 40(5):2733 – 2763, 2012. 24

[16] Breiman, Leo: *Random forests.* Machine Learning, 45(1):5–32, 2001, ISSN 0885-6125. 25

Apêndice A

Apêndice

A.1 Código-fonte

O código-fonte completo se encontra no seguinte link: `https://github.com/arthurveiga/fx_param_classification_from_audio`. Ele consiste em 3 arquivos:

- **TG2_1_criador_dataset.ipynb**;
- **TG2_2_preprocessador.ipynb** e;
- **TG2_3_classificador.ipynb**.

O passo a passo foi explicado no capítulo 3.

A.2 Plot de Datasets com Redução de Dimensionalidade

A.2.1 por PCA

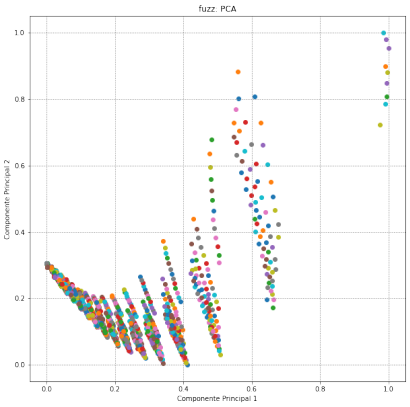

Figura A.1: Fuzz reduzido com PCA.

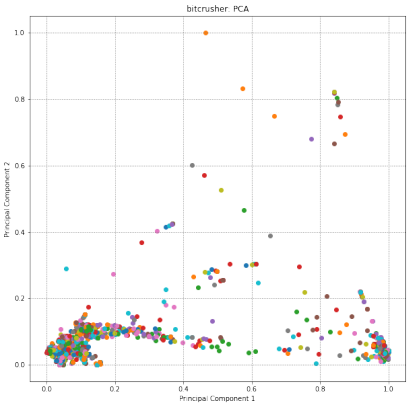

Figura A.2: Bitcrusher reduzido com PCA.

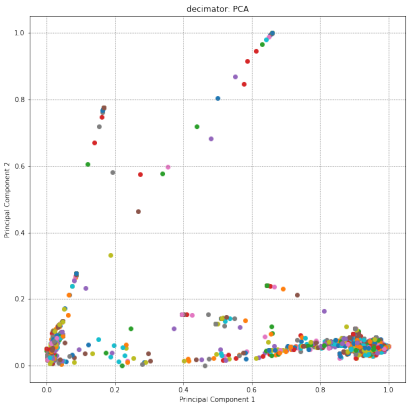

Figura A.3: Decimator reduzido com PCA.

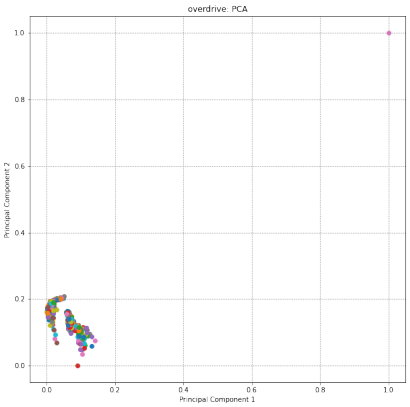

Figura A.4: Overdrive reduzido com PCA.

A.2.2 por t-SNE

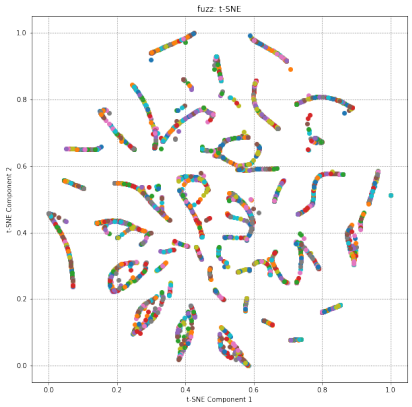

Figura A.5: Fuzz reduzido com t-SNE

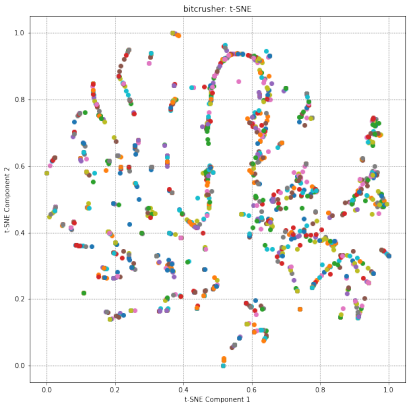

Figura A.6: Bitcrusher reduzido com t-SNE

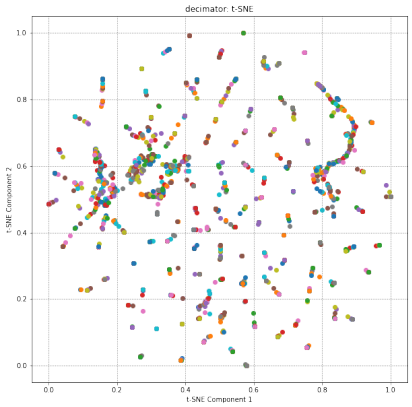

Figura A.7: Decimator reduzido com t-SNE

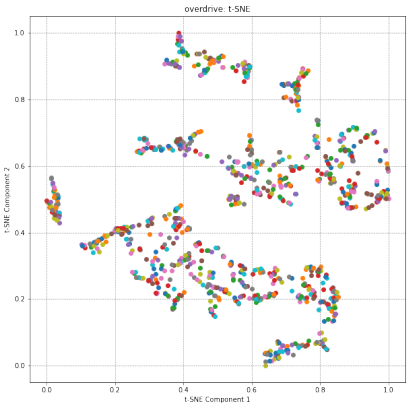

Figura A.8: Overdrive reduzido com t-SNE