NANODEGREE ENGENHEIRO DE MACHINE LEARNING

Projeto Final

Izabela Paulino Fonseca

3 de maio de 2018

# Sumário

# Capítulo 1

# Definição

## 1.1 Visão geral do projeto

No início da existência dos serviços de *streaming* durante a década de 90, a internet ainda era extremamente custosa e possuía uma performance muito menor do que a que temos acesso hoje em nossos próprios celulares. Com o surgimento dos serviços de banda larga e internet móvel, o mercado de *streaming* tomou ainda mais força e vem crescendo cada dia mais.

Pesquisas e dados divulgados pela Nielsen Music no ano de 2017 explicitam que, em média, os americanos escutam mais de 32 horas de música por semana, onde a reprodução da maior parcela dessas músicas é através dos serviços de *streaming*.

Todo o sucesso e aumento do *market share* desta indústria trouxe consigo um crescimento das plataformas de *streaming*, o que tem como consequência uma competição cada vez mais acirrada para a captação e fidelização de seus clientes. Dessa forma, muitas empresas têm feito uso de *Machine Learning* principalmente para definir o perfil musical de seus clientes e fazer recomendações, onde a empresa mais bem sucedida nesse empreitada é o *Spotfy*.

## 1.2 Descrição do problema

Dado o contexto introduzido acima, o problema proposto consiste em criar um modelo capaz de prever como cada usuário classificaria, como "Gostei"ou "Não Gostei", músicas nunca antes ouvidas com base em seus históricos, visando aumentar a assertividade de tais recomendações e torná-las cada vez

mais personalizadas e eficientes. Desta forma, implementaremos e testaremos dois algoritmos de classificação de aprendizado supervisionado, onde, após diversos testes e buscas por melhorias de seus resultados, escolheremos o que melhor performar.

A base de dados que será utilizada neste projeto foi retirada da competição *Spotify Song Attributes* [1], existente no *Kaggle*, e corresponde ao histórico de um único usuário. Tal base consiste em um único arquivo contendo uma tabela com diversas músicas classificadas por este usuário e suas respectivas características técnicas, além do nome da música e de seu artista. Seus atributos serão detalhadamente explorados explicados nas seções que seguem.

Iniciaremos o projeto com a análise dos dados e o teste de diferentes métodos de transformação dos dados categóricos encontrados na base, bem como diferentes pré-processadores. Com os melhores métodos em mãos, tentaremos melhorar o resultado buscando a remoção de possíveis *outliers* e a utilização da técnica *Grid Search* para encontrarmos os melhores valores para os parâmetros de cada algoritmo.

## 1.3 Métricas

Como métricas de avaliação, será considerada a mais trivial de todas para que tenhamos rápidas percepções dos modelos gerados, e para uma análise mais profunda do desempenho dos modelos em cada classe, o *F1 Score*, sendo esta última extremamente adequada ao problema de classificação que queremos solucionar.

Ambas as métricas citadas acima são obtidas através da avaliação da taxa de acerto da base de treinamento, onde a acurácia contabiliza os acertos em geral do modelo nessa base e o F1 Score nos fornece a tabela de confusão, apresentando os falsos positivos, falsos negativos e acertos positivos e negativos.

# Capítulo 2

# Análise

## 2.1 Exploração dos dados

Neste projeto, foi utilizada a base de dados de um usuário do *Spotify*, encontrada em uma competição do *Kaggle* [1] chamada *Sportify Song Attributes*, contendo diversas informações técnicas sobre as músicas ouvidas e classificadas por tal usuário como "gostei", indicado na base de dados como 1, e "não gostei", indicado na base como 0, além do nome da música e artista.

Explorando os dados no início da implementação do código obtivemos as informações do tamanho da base, que neste caso possui 17 colunas e 2017 exemplos de músicas classificadas pelo usuário, sendo a primeira coluna **id** que tem apenas os índices das linhas e, por tanto, será descartada. Dessas 16 colunas restantes, 15 são consideradas dados de entrada e uma detém os dados de saída.

A descrição completa de cada coluna está em [2] e no notebook enviado junto ao relatório, e como são 16 colunas, citaremos apenas aquelas que identificamos como as mais relevantes para esta etapa. A maioria das variáveis presentes nesta base são do tipo *float* ou *int*. Apenas duas delas são do tipo *string*, sendo estas a *song_title* e a *artist*. Estas são consideradas variáveis categóricas, necessitando, assim de uma transformação para que possamos aplicar os algoritmos, os quais aceitam apenas dados numéricos. Finalmente, coluna *target* possui os dados de saída, sendo esta a nossa variável de interesse.

Analisando o resumo estatístico obtido como código, a coluna *speechiness* chamou atenção pelo fato do seu valor máximo estar consideravelmente dis-

tante do valor médio, sendo isso uma consequência de possíveis *outliers* na base de dados. Também foi possível identificar que a base de dados não possui dados faltantes, o que é uma característica muito boa para o desempenho do modelo. Analisando também a coluna *target*, a qual divide o nosso problema em duas classes, vimos que existem 1020 exemplos positivos e 997 negativos, o que nos indica que a quantidade de exemplos das classes está em equilíbrio.

## 2.2 Visualização exploratória

Como os dados se tratam de características técnicas de várias músicas, ou seja, várias dimensões(características técnicas) e vários pontos nesse espaço multidimensional (músicas), uma forma de visualizar esses dados é utilizando o PCA para reduzir o número de dimensões. Abaixo, podemos ver a imagem de um PCA feito com duas componentes principais e um PCA em 3 dimensões.

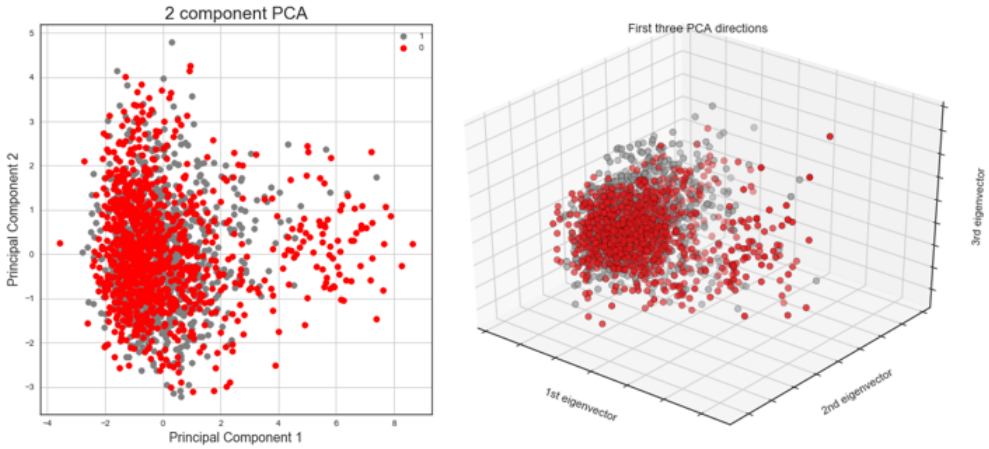


Figura 2.1: PCA com duas e três componentes principais

A variância apresentada pelas componentes do PCA de duas dimensões foi de 21,89% e 11,73%, respectivamente, dando um total de 33,72% da variância total dos dados. Já o PCA de 3 dimensões apresentou para cada uma de suas componentes variâncias de 21,89%, 11,73% e 10,11% dando, agora, um total de 43,83%.

Essas informações e as imagens geradas através da implementação de PCA's de duas e três dimensões, nas figuras 2.1, mostram que aparentemente os dados estão bem misturados, levando-nos a crer que provavelmente todas

as *features* da base de dados trazem informações relevantes pro modelo. Dessa forma, não foi considerado interessante a retirada de *features* neste caso.

Outra maneira de visualizar as informações da base de dados é plotando as informações das colunas duas a duas, em relação à coluna *target*, o que nos permite ver a dispersão dos dados em relação à variável de interesse. Na imagem 2.2, pode-se observar três gráficos, onde o primeiro mostra as colunas *acousticness* versus *danceability* e *danceability* versus *energy*, ambos em relação ao *target*.

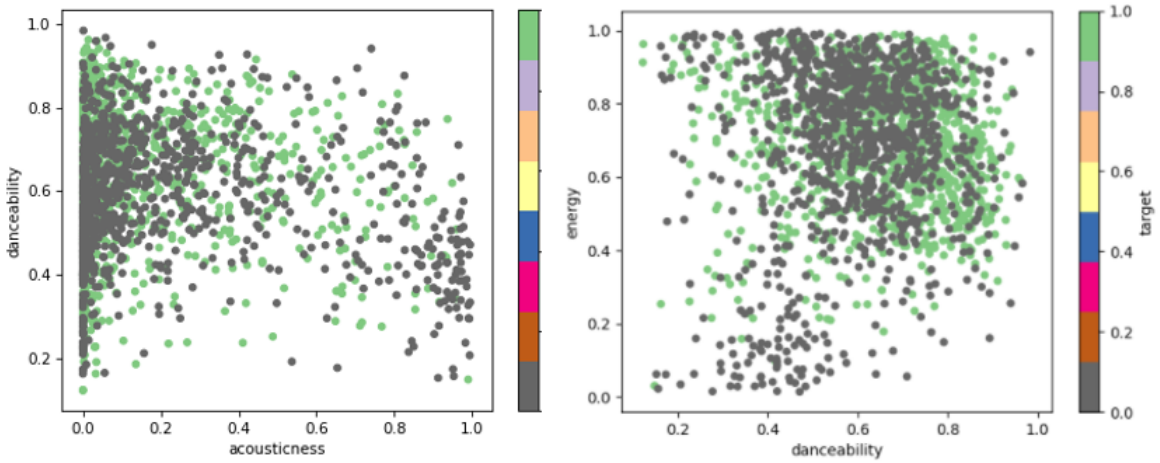


Figura 2.2: Variáveis plotadas duas a duas em relação à *feature target*

Os três gráficos da figura 2.2 exemplificam novamente que não há uma separação muito clara entre os dados, apenas que em algumas regiões parece ser mais provável que o usuário goste ou não de uma música. Por exemplo, na gráfico da extremidade esquerda, na para valores de *danceability* acima de 0.8 e de *energy* acima de 0.4, parecem existir mais pontos que indique que o usuário gosta de músicas com essas características.

Outro fato interessante de se notar pelos gráficos mostrados em 2.1 e 2.2 é a existência e pontos que podem ser considerados *outliers*. Isto nos mostra que é interessante abordar como uma tentativa de melhoria a remoção de *outliers*.

## 2.3 Algoritmos e técnicas

Buscando solucionar o problema proposto, foram implementados e testados dois algoritmos: Classificador *Suport Vector Machine* SVM e o *Gradient Bo-*

*osting Classifier*. Ambos são adequados para aprendizado supervisionado e classificação.

O algoritmo *Suport Vector Machine* (SVM) de aprendizado supervisionado, como o próprio nome já sugere, utiliza vetores de suporte como pontos de referência para efetuar a separação dos dados em classes [3], através dos quais ele constrói um hiperplano ou um grupo de hiperplanos em um espaço multidimensional. A melhor separação considerada é aquela que possui o(s) hiperplano(s) o mais distante possível dos pontos que estão na fronteira das classes, o que faz com que esse hiperplano represente a maior separação possível entre as duas classes.

Já o *Gradient Boosting Classifier* é considerado um método de *ensenble*, que consiste no treinamento de diversos modelos que tem seus resultados combinados (através da média por exemplo) para chegar à um resultado final. O primeiro exemplo de *ensemble* e mais simples se chama *Bagging*, e consiste em construir diversos modelos independentes e combinar seus resultados usando alguma técnica de média (ex.: média ponderada, moda, média comum). O outro é o *Boosting*, que é uma técnica de *ensemble* onde os preditores não são feitos independentemente, e sim sequencialmente [4], sendo este último o tipo em que se encaixa este classificador.

Normalmente, este classificador utiliza árvores de decisão. Inicialmente, é construída uma árvore de decisão com um único *split*, calcula-se o seu erro e cria-se um corretor. Daí, a árvore seguinte é criada utilizando este corretor, e assim esse ciclo se repente, reduzindo gradativamente o erro de cada nova árvore me relação à anterior. Estes algoritmos normalmente geram melhores resultados, mas não há garantias, é importante testas ambas as abordagens.

## 2.4 Benchmark

Após algumas pesquisas, foi possível ver a aplicação de diversos métodos de classificação de aprendizado supervisionado nesta mesma base, uma vez que este é um desafio do *Kaggle* [1]. Dentre eles estão: árvore de decisão, *Random Forest*, *GaussianNB* e *XGBoost*, onde os que atingiram melhores resultados foram *Random Forest* e SVM classifier, sendo este último encontrado em um tutorial de Rodrigo Santana [3]. Os valores de acurácia atingido por cada um foi em torno de 72%.

# Capítulo 3

# Projeto do Sistema

## 3.1 Pré-processamento de dados

Analisando os dados categóricos, citados na seção anterior, fa coluna ***song_title*** foi considerada irrelevante, já que praticamente quase todas as suas células são únicas. O nome do artista, entretanto, deve ser relevante, pois se um usuário gosta de uma música, ou algumas, de um artista específico, provavelmente gostará de outras deste mesmo artista. Assim, decidiu-se testar nesca coluna dus técnicas de pré-processamento: *One Hot Encoder* e *Label Encoder*

Primeiramente, aplicamos o *One Hot Encoder* que consiste em converter os valores categóricos em valores binários, dando como resultado uma matriz de presença. Isto é, se temos três artistas: *The Beatles*, *Beyoncé* e *Interpol*, estes se tornarão colunas da base de dados, ou seja, se a linha 1 refere-se à uma música dos *The Beatles*, as células dessas novas *features* ficarão preenchidas com 1, 0 e 0, respectivamente.A figura 3.1, lado esquerdo, exemplifica bem esse processo.

O problema deste método é que, como existe uma grande variedade de artistas, com sua utilização, foram inseridos todos os diferentes artistas como novas *features*. Isso tornou o modelo extremamente espaço, o que geralmente prejudica o desempenho do modelo. Dessa forma, também foi testada a técnica *Label Encoder* para transformar a coluna *artist* para compararmos o desempenho do modelo inicialmente implementado. A melhor delas também foi aplicada no segundo modelo.

A transformação *Label Encoder* consiste, então, em transformar a coluna *artist* em números inteiros, onde cada artista corresponderia à um código,

sendo este um valor inteiro. O problema deste método é que ele pode enviesar o modelo já que, como os códigos são números ordinais, podem ser colocados em ordem crescente e o modelo pode entender que o valor 449 é maior que 222 ao invés de apenas identificar que são diferentes. Isso é mostrado na figura 3.1, no lado direito.

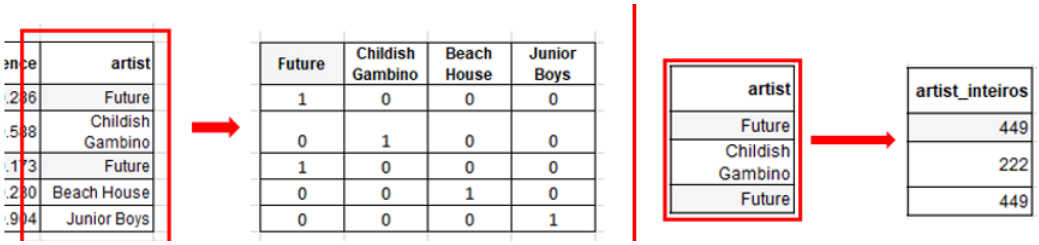


Figura 3.1: Resultado da aplicação do *Label Encoder*

Cada uma das técnicas utilizadas acima foi testada com outras 5 funções de pré-processamento da biblioteca *sklearn.preprocessing*, a qual fornece várias funções úteis comuns de transformação de classes para mudar os vetores linhas das *features* em uma representação que seja mais adequada pra os estimadores [5]. Os pré-processadores usados são:sendo estas: *StandardScaler*, *MinMaxScaler*, *MaxAbsScaler*, *RobustScaler* e *Normalizer*.

Dentre os 5 pré-processadores, foi escolhido o que foi capaz de proporcionar o melhor desempenho ao modelo junto a um dos dos outros dois pré-processadores de variáveis categóricas citados no início desta seção.

## 3.2 Implementação

Parte da implementação do algoritmo do classificador SVM foi baseado em [3], a partir do qual foram feitos vários melhoramentos e modificações para que os resultados obtidos com esse classificador fossem feitos de maneira correta e mais eficiente. O algoritmo *Gradient Boosting Classifier* foi implementado com base em [6], seguindo os mesmo procedimentos que foram realizados para o classificador SVM.

## 3.3 Refinamento

Após os testes com os diferentes pré-processadores, foram feitas testes com diferentes *kernels* e tentativas para melhorar o desempenho dos modelos através da aplicação de técnicas para a remoção de *outliers*, tomando o cuidado de testar a quantidade ótima de *outliers* para serem retirados. Além disso, também foram testados diversos valores dos principais parâmetros de cada algoritmo classificador através do *GridSearchCV*, que também pertence à biblioteca *sklearn*.

# Capítulo 4

# Resultado

## 4.1 Modelo de avaliação e validação

### 4.1.1 Implementando SVM e os pré-processadores

Iniciamos a implementação do modelo do classificador SVM, testando primeiramente a transformação da coluna *artist* utilizando o *One Hot Encoder*, além dos outros pré-processadores obtidos através da biblioteca *sklearn.preprocessing*. A implementação do classificador junto a cada pré-processador da biblioteca *sklearn.preprocessing* foi feita utilizando pipelines da biblioteca *sklearn.pipeline*.

A base de dados foi dividida em treino e teste. Os dados mostrados a seguir são resultado da acurácia do processo de validação cruzada utilizando apenas a base de treinamento, e só depois de definir os melhores pré-processamentos e refinamento, o modelo será aplicado na base de teste.

| SVM com *OneHotEncoder* | |
|---|---|
| **Pré-processador** | **Acurácia** |
| StandardScaler | 68.26% |
| MinMaxScaler | 51.07% |
| MaxAbsScaler | 51.07% |
| RobustScaler | 62.58% |
| Normalizer | 51.07% |

| SVM com Label*Encoder* | |
|---|---|
| **Pré-processador** | **Acurácia** |
| StandardScaler | 72.72% |
| MinMaxScaler | 67.05% |
| MaxAbsScaler | 66.67% |
| RobustScaler | 74.16% |
| Normalizer | 51.07% |

Figura 4.1: Resultado do classificador SVM com aplicação do *One Hot Encoder*

É possível observar pelos resultados mostrados na figura 4.1, lado es-

querdo, que a melhor de acurácia foi de 68,26%, que podemos considerar ainda um valor relativamente baixo. Possíveis razões para obtermos esses valores são: a inserção de uma grande quantidade de *features* na base de dados, deixando a matriz muito esparsa, a quantidade de dados (músicas classificadas) da matriz e a presença de *outliers*, por exemplo.

Tentando melhorar o resultado apresentado pelo primeiro método, e considerando a grande alteração que fizemos na base de dados ao inserirmos uma enorme quantidade de *features* através da aplicação do *One Hot Encoder*, tentou-se classificar os dados utilizando apenas a transformação do *Label Encoder*.É importante lembrar que tal método pode enviesar o modelo, uma vez que este pode identificar uma ordem entre os códigos únicos gerados para cada artista, como já dito anteriormente.

Vendo a imagem 4.1, lado esquerdo, constatamos que o resultado melhorou consideravelmente o nosso resultado consideravelmente, saindo de uma acurácia de 68,26% para 74,15% e, obviamente, tivemos uma redução do tempo de processamento. Além disso, o pré- processador que teve o melhor desempenho anteriormente foi o *StandardScaler*, enquanto que, ao usamos apenas o *Label Encode* para transformar os dados da coluna categórica *artist*, o melhor pré-processador passou a ser o *RobustScaler*.

Após estes testes, concluímos então que o melhor método de transformação da coluna *artist* foi o *LableEnconde*, utilizando o pré-processador *RobustScaler*.

### 4.1.2 Refinamento dos resultados do SVM usando *Kelnel's*

Após os processos e testes realizados acima, vamos agora utilizar os dois melhores pré-processadores, *Label Encode* e *RobustScaler*, para testarmos os diferentes tipos de *kelnel's* e constatamos que o que teve melhor desempenho foi o ***rbf***, mostado na figura 4.2, sendo este o default do SVM, o que justifica o mesmo resultado da melhor acurácia apresentada anteriormente.

### 4.1.3 Refinamento dos resultados removendo *outliers*

Utilizando o mesmo processo feito no projeto 3 para retirar os *outliers*, testamos essa técnica para tentar melhorar o nosso resultado ainda mais. É

| SVM com diferentes *kernel's* | |
|---|---|
| **Kernel** | **Acurácia** |
| rbf | 74.16% |
| poly | 65.23% |
| linear | 66.22% |
| sigmoid | 51.57% |

Figura 4.2: Resultado do classificador SVM com diferentes *kernel's*

importante lembrar que ainda estávamos trabalhando apenas com a base de treino. Encontraremos, então, os *outliers* de cada *featurs* utilizando o *interquartile range* e usaremos o SVM com os pré-processadores e *kernel* já eleitos como os melhores, testando a retirada dos *outliers* de acordo com a frequência que aparecem entre as *features*.

Por exemplo, se eu tenho o mesmo *outlier* pra duas *features*, isso significa que a frequência desse *outlier* é 2 . Dessa forma, testaremos a acurácia retirando os todos os *outliers*, tirando os que aparecem 2 vezes ou mais, 3 vezes ou mais e assim por diante.

| SVM com remoção dos *outliers* | |
|---|---|
| **A partir da frequência** | **Acurácia** |
| >=1 | 71.68% |
| >1 | 73.22% |
| >2 | 74.38% |
| >3 | 74.27% |

Figura 4.3: Resultado do classificador SVM retirando *utliers*

A imagem 4.3 nos mostra, então, que quando retiramos os *outliers* que aparecem 3 vezes ou mais nos deu o melhor resultado, melhorando a acurácia do nosso modelo que antes estava com 74.16% para 74.38%, o que mostra que de fato a retirada de *outliers* beneficiou o desempenho do modelo. Assim, tendo encontrado a melhor frequência para retirar os *outliers*, este mesmo processo tem que ser feito com a base de teste para que possamos ter uma base de comparação correta.

### 4.1.4 Teste de diferentes valores de C e Gamma usando *GridSearchCV*

Outra tentativa de melhorar o resultado do classificador SVM foi aplicando o *GridSearch*, para que pudéssemos testar diferentes valores dos parâmetros C e Gamma com os pré processadores de melhor desempenho. Fizemos tal teste com a base sem retirar os *outliers* e a base com a retirada destes. Assim, criamos uma lista com os diferentes possíveis valores para cada parâmetros, e aplicamos o *GridSearch*. Foram usados 10 *folders* na validação cruzada. Todos os cálculo de acurácia também estão sendo feitos com validação cruzada, para que possamos garantir que o modelo não irá sofrer *overfit*.

Através deste teste, os melhores valores encontrados dos parâmetros foram C = 1 e Gamma = 0.1, além do melhor score deste grid ter sido de 74.93%. Com isso em mãos, treinamos, sem validação cruzada, estes parâmetros e pré-processadores na base sem remoção dos outliers e aplicamos na base de treino, obtendo 88,72% de acurácia, e em seguida na base de teste, obtendo 76,73% de acurácia.Este valor da acurácia de no treino é maior do que a obtida com a validação cruzada pois utilizamos todo a base de treino para treinar o modelo. Podemos ver que o resultado obtido na base de teste neste caso foi de 76,73%, o que nos mostra um resultado satisfatório.

Fazendo o mesmo processo para de *GridSearchCV* com a base sem os *outliers*, os valores ótimos dos parâmetros C e Gamma ficam da mesma forma que os valores anteriores. Entretanto, o valor do melhor *score* deste grid foi de 75,08%, comprovando a melhoria que a remoção dos *outliers* trouxe para o desempenho do modelo.

Treinando o modelo com os melhores parâmetros e pré-processadores e remoção dos outliers, e aplicando na base de treino tivemos uma acurácia de 81,50%, e aplicando à base de teste conseguimos uma acurácia de 76,96%. Este é um resultado extremamente satisfatório, e se compararmos a melhoria que tivemos no modelo observando a acurácia que foi calculada no decorrer dos teste em cima da base de treino, saímos de 68,26% para 81,50%, o que nos indica uma ótima melhora de performance.

### 4.1.5 Implementando *GradientBoostingClassifier*

Para que pudéssemos ter uma outra perspectiva em relação aos algoritmos de classificação, testamos também o classificador *GradientBoostingClassifier* já explicado anteriormente. Dessa forma, utilizamos nesse algoritmo os pré-

processadores que geraram os melhores resultados durante a implementação do classificador SVM (*RobustScaler* e *LabelEncoder*).

Testamos inicialmente de forma simples, sem utilização do *GridSearchCV* para verificar o desempenho que teríamos sem melhorias e, utilizando os parâmetros iguais a *n_estimators* = 100, *learning_rate* = 1.0 e *max_depth* = 1 já conseguimos treinar um modelo que conseguisse uma acurácia na base de teste de 76,24%, que é um valor bem maior do que o teste simples feito com o SVM.

Depis, buscamos os valores ótimos para os parâmetros *n_estimators*, *learning_rate* e *max_depth* através do *GridSearchCV*. Testamos com a base sem a retirada dos *outliers* e com a retirada dos *outliers* para vermos se isso provocava alguma diferença neste classificador. Assim, os valores dos parâmetros encontrados pelo *GridSearchCV* foram os mesmos sem e com a remoção dos outliers, sendo iguais a *learning_rate* = 0.1, textitmax_depth = 4 e *n_estimators* = 500. O score máximo atingido pelo *GridSearchCV* sem a remoção dos *outliers* foi 78,29%, e com a remoção dos *outliers* foi 77,88%,o que nos mostra que a estratégia de remoção dos *outliers* para este classificador não adequada.

Testando sem a validação cruzada, tivemos resultados muito bons com a utilização deste classificador. A acurário obtida aplicando o modelo na base de treino foi de 99,88%, o que é normal, já que o *GradientBoostingClassifier* tende à "overfitar"a base de treino. Os parâmetros ótimos obtidos com a validação cruzada foram importantes neste aspecto, para que mesmo dando *overtit* na base de treino, o classificador pudesse ter um bom desempenho em novas bases. Já a acurácia obtida aplicando o modelo na base de teste foi de 82,67%, o que nos mostra que este algoritmo teve um excelente desempenho. Ao fazermos este mesmo teste com a base sem os outliers e aplicarmos o modelo gerado na base de teste, obtivemos uma acurária de 79,05%, o que mostra que esta não é uma boa estratégia para este algoritmo.

### 4.1.6 Comparando o F1 *score* dos dois melhores classificadores

Para compararmos de forma mais detalhada os resultados obtidos, observaremos o F1 score do melhor modelo de cada algoritmo utilizado, mesmo tendo classes com quantidades de exemplos balanceadas. Isso vai nos permitir ver

o desempenho desses modelos em cada uma das classes.

O valor de *Precision* é obtido através do número de previsões positivas (1) corretas, ou seja, tudo o que foi classificado corretamente como 1, dividido pelo número de previsões iguais a 1. Já o *Recall* corresponde ao número de previsões positivas (1) corretas dividido pela quantidade de positivos reais existentes na base.

Classificador SVM

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.78 | 0.78 | 0.78 |
| 1 | 0.76 | 0.76 | 0.76 |
| avg / total | 0.77 | 0.77 | 0.77 |

*GradientBoostingClassifier*

| | precision | recall | f1-score |
|---|---|---|---|
| 0 | 0.86 | 0.82 | 0.84 |
| 1 | 0.80 | 0.84 | 0.82 |
| avg / total | 0.83 | 0.83 | 0.83 |

Figura 4.4: F1 Score dos melhores resultados obtidos por cada classificador

Observando a figura 4.4, lado esquerdo que se refere ao melhor resultado do SVM, podemos ver que tanto para *precision* quanto para *recall*, os maiores valores foram da classe negativa, sendo ambos iguais a 78%, 2% maior que da classe positiva para os dois casos. Assim, o F1 *score* para a classe negativa foi maior nesse caso, e a média total do F1 *score* foi de 77%, o que podemos considerar um desempenho satisfatório.

A figura 4.4, lado direito, referente ao melhor resultado do *GradientBoostingClassifier*, da mesma forma como no lado esquerdo, também mostrou um F1 *score* maior para a classe negativa, apesar do valor de *recall* da classe positiva ser maior. A média total dos três valores, *precision*, *recall* e F1 *score*, foram iguais a 83%, sendo este um valor muito bom, o que demonstra que este foi o nosso melhor classificador.

## 4.2 Justificativa

Como podemos lembrar, no *benchmark* feito na seção 2.4 a melhor acurácia encontrada foi de 72%. Já os resultados obtidos com os dois modelos testados nesse projeto conseguiram uma acurácia de 76,96% com o classificador SVM e 82,23% com o *GradientBoostingClassifier*. Tais valores além de serem bem melhores que os obtidos na etapa de *benchmark* são significativos e suficientes, tendo resolvido o problema apresentado para este projeto de maneira extremamente satisfatória.

# Capítulo 5

# Conclusão

## 5.1 Foma livre de visualização

Como verificamos na seção anterior, foram obtidos resultados muito bons com a utilização do classificador *GradientBoostingClassifier* acredito que principalmente pelo fato dos dados terem classes com quantidades de exemplos muito balanceadas e nenhum valor faltante. Acredito que para bases do *Spotify*, dada a organização que eles devem ter atualmente para preencher essa base, não será um problema. Entretanto, muito provavelmente para bases que possuam vários valores faltantes e para usuários que tenham sua divisão de exemplos classificados como "gostei"e "não gostei"muito desbalanceada, podemos enfrentar uma perda de desempenho considerável.

## 5.2 Reflexão

O processo seguido para implementar este projeto pode ser descrito nos passos como segue:

- **1** Foi feita uma pesquisa para descobrir um projeto que seria de interesse;
- **2** Escolhido o projeto, buscamos a base de dados;
- **3** Decidimos se haviam colunas que deveriam ser retiradas e pré-processadas, e buscamos outros pré-processadores para testar;
- **4** Decidimos os modelos que seriam implementados e testados para podermos comparar diferentes técnicas de classificação;

- **5** Treinamos primeiramente o classificador SVM com diferentes pré-processadores e *kernel's* para elegermos os melhores e assim partir para o refinamento dos resultados;

- **6** Partindo para a melhoria, testamos os parâmetros C e Gama usando *GridSearchCV* e a remoção de *outliers* para elegermos o melhor modelo e finalizar esta etapa;

- **7** Por fim, testamos o algoritmo *GradientBoostingClassifier* inicialmente usando apenas os dados tratados com *LabelEncoder*;

- **8** Depois, fizemos também um *GridSearchCV* para encontrar os melhores valores para os parâmetros *n_estimators*, *learning_rate* e *max_depth*;

- **9** Comparamos os modelos gerados pelos dois algoritmos, e verificamos que o que teve o melhor desempenho foi o modelo gerado com *GradientBoostingClassifier*;

A maior dificuldade estava em melhorar o resultado do classificador SVM, além de utilizar os pipelines e implementar imagens em três dimensões a partir de PCA's, já que estes eram métodos inéditos para mim.

Os resultados obtidos ao final do projeto foram de extrema satisfação e superaram expectativas, e acredito que boa parte do que foi implementado possa ser utilizado de forma geral para resolver problemas semelhantes.

## 5.3 Melhorias

Após pesquisar diversos métodos para melhorar o desempenho dos modelos, foi encontrado um método chamado *Stacking*, muito utilizado em competições de machine learning, já que normalmente melhora consideravelmente o resultado dos modelos. Um início de implementação foi feita, mas como não foram encontradas informações que diziam respeito à forma de se obter um limiar ótimo para transformar as probabilidades obtidas no final de sua aplicação em 0's e 1's, essa técnica foi deixada para ser testada em uma outra oportunidade.

Esta técnica pode ser aplicada nos algoritmos usados, apesar dos resultados apresentarem já serem suficientemente bons.

# Referências Bibliográficas


[1] Spotify. Spotify song attributes. https://www.kaggle.com/geomack/spotifyclassification/data. Acessado em 2018-04-21.

[2] Spotify. Get audio features for a track. https://beta.developer.spotify.com/documentation/web-api/reference/tracks/get-audio-features/. Acessado em 2018-04-23.

[3] Rodrigo Santana. Classificando músicas do spotify com svm, com códigos python. http://minerandodados.com.br/index.php/2018/04/04/spotify-svm-python/, 2017.

[4] Prince Grover. Gradient boosting from scratch. https://medium.com/mlreview/gradient-boosting-from-scratch-1e317ae4587d, 2017.

[5] scikit-learn developers. Preprocessing data. http://scikit-learn.org/stable/modules/preprocessing.html. Acessado em 2018-04-21.

[6] scikit-learn developers. sklearn.ensemble.gradientboostingclassifier. http://scikit-learn.org/stable/modules/generated/sklearn.ensemble.GradientBoostingClassifier.html. Acessado em 2018-04-23.