Projeto Final

Nanodegree Engenheiro de Machine Learning

Marcos Antônio da Costa Tanaka
13 de Agosto de 2017

Definição

Visão Geral

Uma nota musical é o nome dado a uma determinada frequência de vibração do ar. Podemos representar as notas pela convenção Do–Re–Mi–Fa–Sol–La–Si ou, de forma simplificada, com as sete primeiras letras do alfabeto latino (A, B, C, D, E, F e G). Ao tocar três ou mais notas simultaneamente, obtemos um acorde. Cada acorde possui um nome baseado na sua nota fundamental - a nota mais grave.

O objetivo deste estudo é criar um algoritmo que utilize machine learning para identificar a nota fundamental de um acorde. Para treiná-lo, gravei diversos acordes, visto que uma base de dados pronta não foi encontrada (mais detalhes nas próximas seções).

Problema

O objetivo é criar um algoritmo de classificação cuja entrada será um acorde e a saída será uma das sete notas (A, B, C, D, E, F ou G), conforme sua nota fundamental. Para atingir este objetivo, será necessário:

1. Gravar e pré-processar diversos acordes em diferentes notas.
2. Identificar e extrair dos arquivos de som (.wav) atributos que possam ser utilizados para a classificação.
3. Treinar classificadores para que consigam determinar qual a nota fundamental de um determinado acorde.
4. Avaliar a performance e escolher o melhor modelo.

Métricas

A métrica escolhida para avaliar a performance do algoritmo é a acurácia, que é a proporção de elementos corretamente classificados com relação ao número total de elementos. É uma métrica comumente utilizada em problemas de classificação multi-classe:

$$acuracia = \frac{quantidade\ acertos}{tamanho\ conjunto\ teste}$$

Para evitar que a quantidade de elementos de uma determinada classe no conjunto de teste seja desproporcional a quantidade de elementos desta classe utilizados para treinar o algoritmo, a divisão dos dados foi feita levando esta proporção em consideração.

Desta forma, temos uma métrica que representa o quão confiável é o resultado do algoritmo para um determinado acorde.

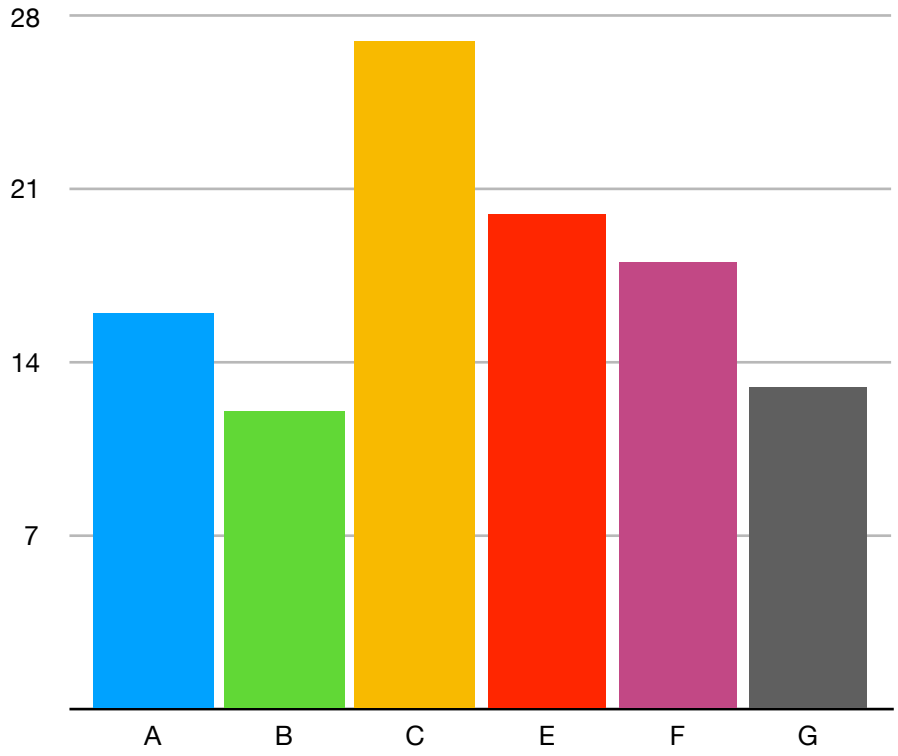
Análise

Explorando os Dados

Encontrei alguns conjuntos de dados, como o Million Song Dataset, porém eles possuem apenas músicas inteiras e não acordes específicos. Como o objetivo deste estudo é classificar acordes de acordo com sua nota fundamental, precisei criar meu próprio conjunto de dados.

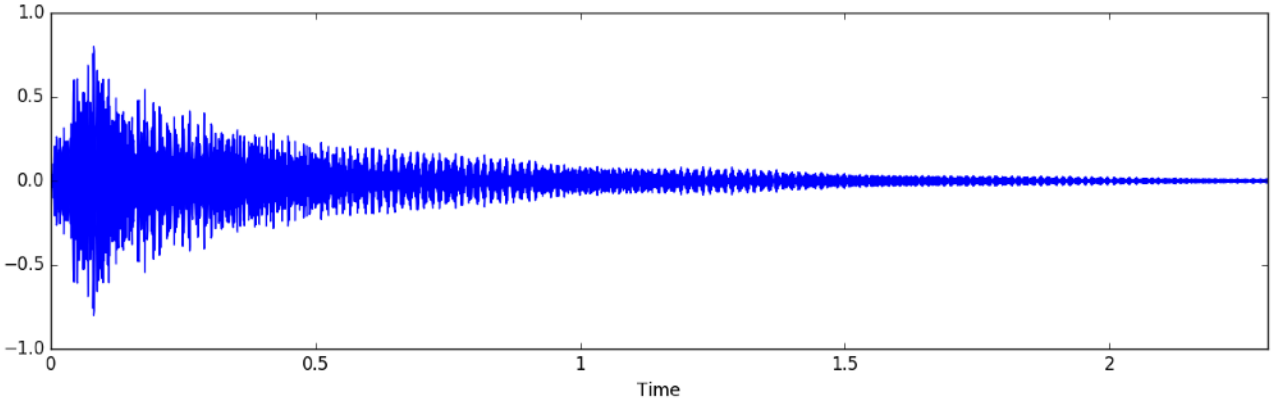
Para isto, gravei 156 acordes individualmente, utilizando um ukulele como instrumento e um smartphone como gravador. Por falhas na execução dos acordes, gravação ou edição, tive que remover 50 que não apresentaram as características corretas, ficando assim com 106 gravações.

O gráfico abaixo demonstra a distribuição de cada nota fundamental entre os acordes que compõem o conjunto de dados utilizado:

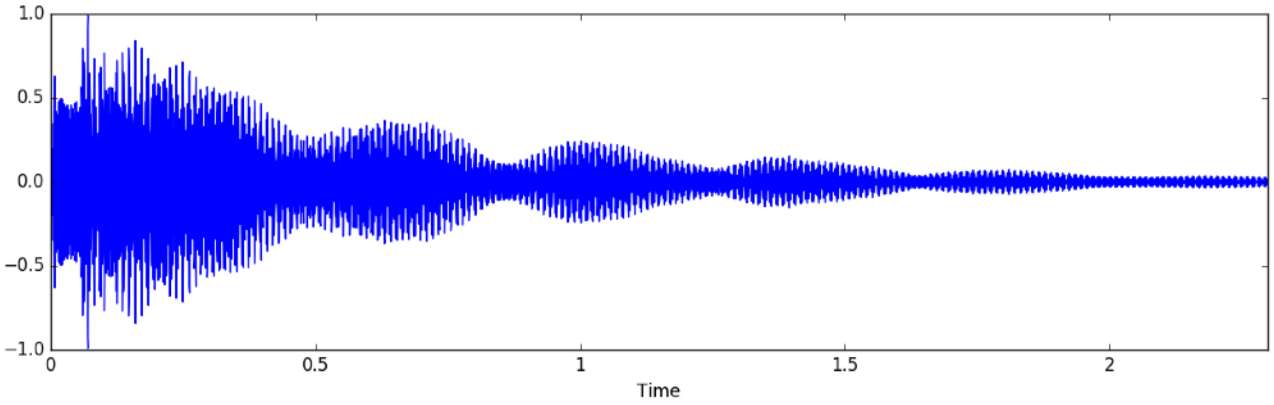


Visualização Exploratória

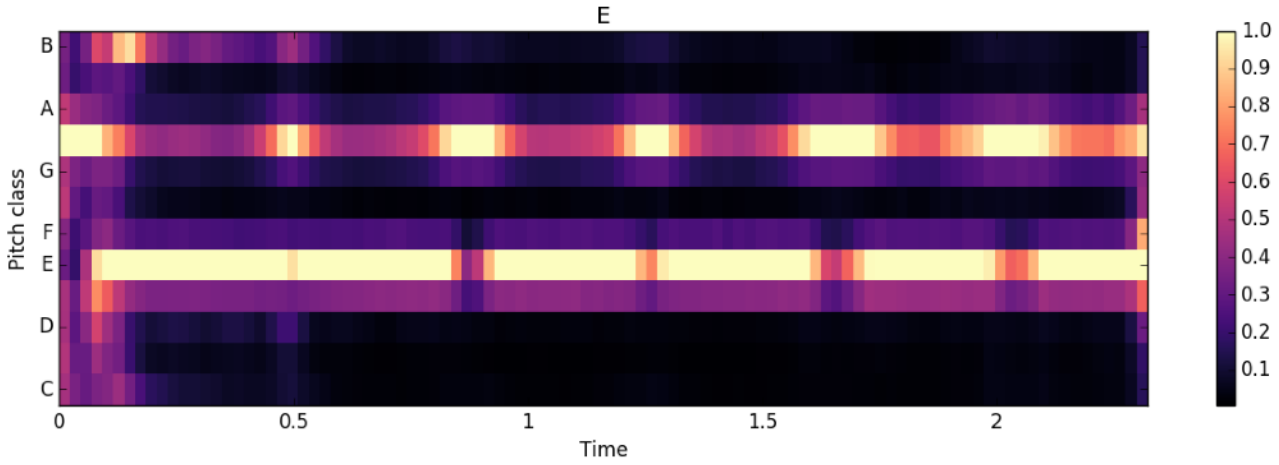
Uma das formas de visualizar os sons utilizados é por meio de um gráfico de forma de onda. Abaixo temos a forma de onda do acorde A após a edição:



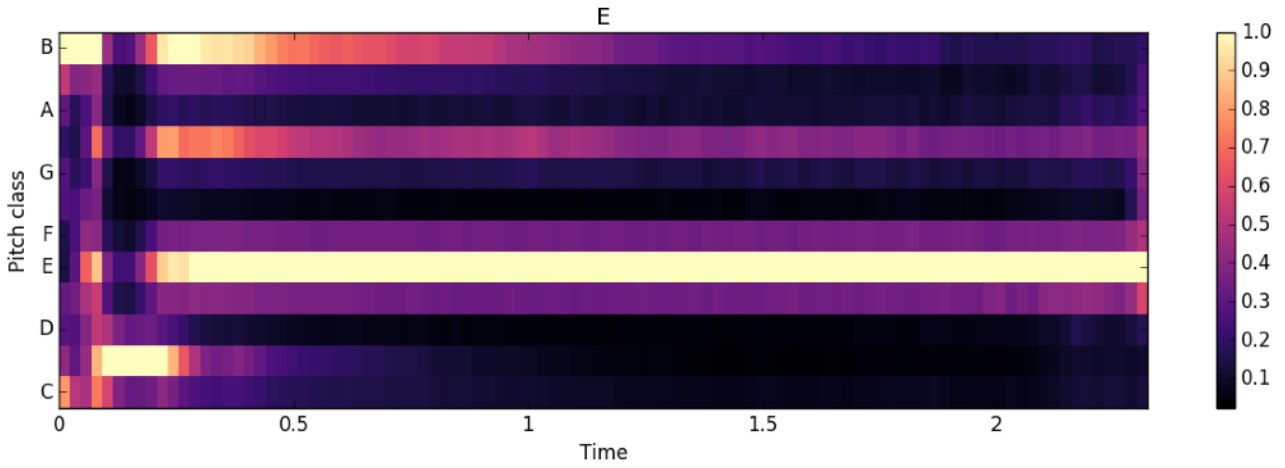
E aqui a forma de onda do acorde E após a edição:



Outro gráfico útil para visualizar os atributos que serão usados na identificação da frequência de cada acorde é o chromagrama. No eixo y temos as possíveis notas e no eixo x o tempo. A cada unidade de medida, o valor varia entre 0 e 1, indicando a quantidade de cada nota presente naquele instante. Abaixo temos um chromagrama do acorde E:



Como era de se esperar, a nota mais presente é a E. Outras notas compõem este acorde, porém



a fundamental é a E. Abaixo temos outro acorde que também possui a nota E como fundamental, o acorde E6:

Algoritmos e Técnicas

Comparei o desempenho dos algoritmos DecisionTree, SVM, GaussianNB e AdaBoost.

DecisionTree: apesar de não ser um domínio muito simples, escolhi testar como seria o desempenho do DecisionTree, por ser o mais simples dos quatro. Ele funciona tentando encontrar regras de decisão simples, cujas condições separam os dados de acordo com suas características comuns.

SVM: escolhi o SVM por se aplicar bem a domínios complicados que possuem uma margem de separação clara entre as classes. Como a diferença entre uma classe e outra está na quantidade de cada nota presente no acorde, espero que o SVM consiga encontrar esta margem e classificar corretamente os acordes. Além disso, o desempenho do SVM pode ser melhor do que os demais devido a trabalhar melhor com uma quantidade reduzida de dados de treinamento, conforme verifiquei no segundo projeto que fiz neste *nanodegree* (projeto *Construindo um Sistema de Intervenção para Estudantes*). O SVM classifica os dados procurando uma fronteira entre eles com base em suas características comuns. Ao mesmo tempo, ele certifica-se de manter uma margem de segurança entre estas fronteiras, de forma que novos dados não sejam classificados de forma indevida. Ele também funciona para determinar fronteiras não-lineares entre as classes.

GaussianNB: foi escolhido pois também apresenta um desempenho relativamente bom quando trabalha com poucos atributos. Ele utiliza conhecimentos prévios sobre o modelo, juntamente com as características do elemento que se está tentando prever a classe, para tentar descobrir a qual classe o elemento pertence.

AdaBoost: comparei também o desempenho do AdaBoost para verificar como a técnica de utilizar diversos modelos simples, focando onde os erros são maiores a cada iteração, poderia trazer de resultado para este tipo de problema.

O método GridSearch foi utilizado no DecisionTree e no SVM para encontrar os melhores hiper parâmetros.

Benchmark

Encontrei estudos similares, porém nenhum que utilizasse apenas acordes individuais como conjunto de dados. Em uma tese de mestrado[1] que utilizou arquivos de 30 segundos, com conjunto de dados relativamente maior, a acurácia ficou em torno de 75%. Outro estudo[2] conseguiu acurácia de 58,33% Com base nestes resultados, e considerando um conjunto de dados menor, o objetivo é conseguir uma acurácia em torno de 65%.

Metodologia

Pré-processamento dos Dados

Conforme explicado anteriormente, não encontrei um conjunto de dados da forma que eu queria para este estudo (contendo apenas acordes, categorizados pela nota fundamental). Assim, gravei os acordes utilizando um ukulele, afinado nas seguintes frequências:

Nota	Frequência (Hz)
G	~ 390
C	~ 255
E	~ 326
A	~ 439

Cada gravação foi editada no software Audacity, onde foram executados os seguintes passos:

1. Aplicado efeito *Noise Reduction* utilizando como base o primeiro segundo de cada gravação (reservado justamente para identificação do padrão de ruído a ser removido).
2. Aplicado efeito *Amplify* para igualar o volume de todas as gravações com base em um pico de amplitude de 0 dB (decibéis).
3. Removidos os primeiros segundos de silêncio (que foram utilizados para a redução de ruídos).
4. Removido tudo o que passou de 2,3 segundos de gravação para manter uma duração comum entre todos os acordes.

Os sons processados foram salvos em formato WAV (*WAVEform audio format*) com 2,3 segundos de duração cada.

---

[1] http://kt.era.ee/supervision/Alyanaki2011.pdf

[2] http://cs229.stanford.edu/proj2016/report/Mahieu-DetectingMusicalKeyWithSupervisedLearning-report.pdf

De cada som foi extraído o chromagrama, que serviu de entrada para os algoritmos utilizados. Sua estrutura é um *array* de 12 posições. Cada posição representa uma nota[3] e possui um número, que é a média da quantidade de presença daquela nota ao longo de todo o som. Para extrair esta informação, utilizei a biblioteca LibROSA[4]. Portanto, para o chromagrama do acorde E6 visto anteriormente, temos o seguinte array correspondente:

C	C#	D	D#	E	F	F#	G	G#	A	A#	B
0.15	0.19	0.10	0.35	0.94	0.34	0.07	0.15	0.44	0.14	0.17	0.44

É possível perceber, desta vez numericamente, que a nota mais presente no acorde E6 é a E. Este *array* é a informação extraída de cada arquivo de som do conjunto de dados utilizado neste estudo, e é com base em seus 12 atributos que os algoritmos foram treinados.

Implementação

Com o conjunto de dados pronto, comecei a implementação escrevendo os métodos responsáveis por ler os arquivos .wav e por extrair os atributos utilizando a biblioteca *LibROSA*. Em seguida, dividi o conjunto de dados em treinamento e teste utilizando a funcionalidade *train_test_split* do scikit-learn. A ideia era deixar 24% dos dados para testar os algoritmos treinados, preservando o percentual de amostras de cada classe em cada conjunto.

Ao longo do desenvolvimento, verifiquei que para obter resultados mais consistentes e confiáveis, seria interessante ter um conjunto de validação, para evitar que os algoritmos se ajustassem demais ao conjunto de testes conforme seus hiper parâmetros eram melhorados. Para não ter que diminuir ainda mais o conjunto de dados utilizado para treinamento, optei por utilizar a técnica *cross-validation*, com a estratégia *Stratified K-Folds*, preservando assim o percentual de amostras de cada classe.

Também experimentei utilizar outros atributos para descrever o conjunto de dados. No início não estava claro para mim que apenas o chromagrama seria suficiente. A biblioteca *LibROSA* fornece diversos outros métodos que extraem diferentes características de um arquivo sonoro, então testei o desempenho dos algoritmos utilizando características como *mfccs*, *melspectogram* e *spectral contrast*. Este conjunto de características gerou quase 150 atributos. Tentei reduzir a dimensionalidade utilizando PCA, o que resultou em 20 atributos que descreviam 97,69% dos dados. Nesta configuração, alguns algoritmos se ajustaram demais aos dados de treinamento. É o caso do SVM, que acertou 100% ao ser testado contra este conjunto. Apesar disto, obteve menos de 35% no conjunto de teste. Ao comparar o desempenho dos algoritmos com diversas combinações de atributos, juntamente com estudos sobre os mesmos, foi verificado que utilizar apenas o chromagrama traria os melhores resultados.

---

[3] O método *chroma_stft* da biblioteca de processamento de áudio *LibROSA* considera tanto os tons quanto os semitons, por isso retorna 12 notas (A, A#, B, C, C#, D, D#, E, F, F#, G e G#). Neste estudo estou trabalhando apenas com tons, portanto, 8 notas.

[4] http://librosa.github.io/librosa/

Medi a acurácia dos modelos com diferentes quantidades de subconjuntos para a validação cruzada, e o melhor resultado veio ao utilizar 9 subconjuntos.

Funções auxiliares como a geração de gráficos dos arquivos sonoros foram implementadas utilizando exemplos vistos em artigos e nas documentações das respectivas bibliotecas.

Refinamento

Inicialmente os resultados estavam inconsistentes. Ao mudar um pouco do conjunto de treinamento ou teste, os resultados mudavam abruptamente. Com o SVM, por exemplo, consegui taxas de acerto variando de 20% a 60%. Após implementar a validação com *cross-validation*, a consistência dos resultados melhorou muito.

Uma medida tomada para melhorar a acurácia foi testar o parâmetro do número de subconjuntos que o cross-validation deveria gerar. Os melhores resultados foram obtidos com 9 subconjuntos.

Outra medida foi o uso de *GridSearch* para encontrar os melhores parâmetros para os algoritmos. Sem ele, a taxa de acerto do SVM no conjunto de teste foi de 40%. Com ele, o acerto foi de 63%.

Para a DecisionTree, não houve grandes diferenças no uso do *GridSearch*, indo de 46% para 49% de acurácia no conjunto de teste.

Abaixo temos os parâmetros hiper parâmetros explorados pelo *GridSearch* para o SVM:

Tentativa/ Parâmetros	kernel	C	gamma	coef0
1	linear	De 0.1 a 1.0, somando 0.1 a cada tentativa	-	-
2	rbf		De 0.1 a 1.0, somando 0.1 a cada tentativa	-
3	sigmoid			-
4	-	-	-	De 0 a 5, somando 1 a cada tentativa

E aqui os explorados para a DecisionTree:

Tentativa/ Parâmetros	max_depth
1	De 2 a 10, somando 1 a cada tentativa

Resultados

Avaliação e Validação do Modelo

O modelo escolhido foi o SVM. Ele foi escolhido, juntamente com a configuração de parâmetros, pois foi o que apresentou melhor desempenho para este problema. A avaliação foi feita com *cross-validation* utilizando 9 subconjuntos, o que trouxe robustez aos resultados por diminuir o ajuste do algoritmo aos dados de treinamento e teste. A divisão dos dados foi feita preservando o percentual de amostras de cada classe, para que seja possível medir a performance de acordo com o que foi aprendido.

A tabela abaixo descreve o resultado final obtido com os quatro algoritmos testados:

AdaBoostClassifier	DecisionTreeClassifier	GaussianNB	SVM
34%	46%	52%	63%

Justificativa

Comparando a meta definida no benchmark com a acurácia obtida, vemos que o modelo não atingiu os resultados propostos. Apesar disto, a diferença foi pequena, 2% menor do que a meta definida. Comparando com o primeiro estudo de referência, obtive acurácia 12% menor. Já com o segundo estudo, a acurácia aqui foi 4,67% maior.

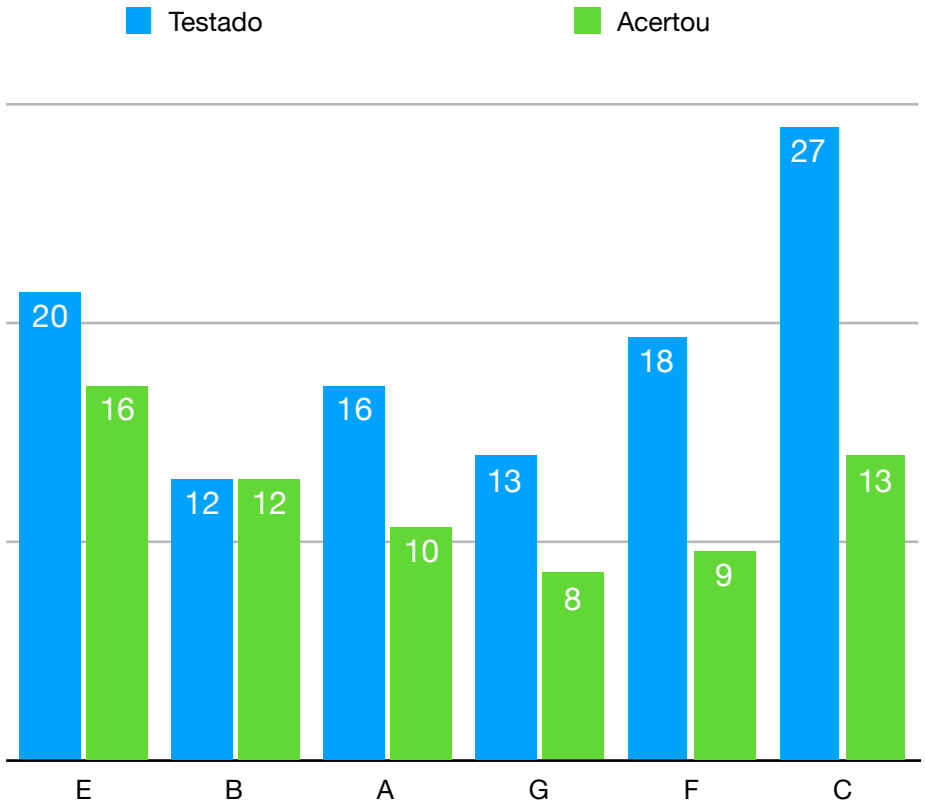
Conceitualmente, os atributos do chromagrama são a melhor opção de entrada para treinar os algoritmos. Isto foi verificado comparando o desempenho de diversos atributos diferentes, e também com pesquisas em estudos similares, conforme explicado anteriormente. O resultado final foi obtido após um processo de validação cruzada e estratificação, para assegurar que o modelo não se ajuste demais as características dos dados de treinamento e teste, resultando em uma melhor generalização do que foi aprendido.

Comparando os resultados obtidos neste estudo com outros similares, vejo que este pode ser um bom modelo para resolver o problema. Para obter uma acurácia melhor, algumas ações devem ser tomadas (ver seção *Melhorias*).

Conclusão

Visualização

O gráfico abaixo mostra a performance do algoritmo para cada nota. Podemos ver que algumas são mais difíceis de classificar do que outras. Isto pode ser um reflexo das diferentes composições que cada nota possui, e pode ser utilizado para identificar onde focar esforços na tentativa de melhorar a acurácia do modelo.



Reflexão

O processo utilizado neste estudo foi:

1. Escolher o problema que seria resolvido com *machine learning* e estudar seu domínio.
2. Procurar conjuntos de dados para treinar e testar o modelo.
3. Como não foi encontrado um conjunto de dados com as características desejadas, foi preciso criar um, gravando diversos acordes dentre os conjuntos de notas que seriam classificadas.
4. Pré-processar os arquivos gravados para padronizá-los e remover características indesejadas.
5. Definir um benchmark com base em estudos similares.
6. Escolher algoritmos adequados para a tarefa e treiná-los utilizando diversas técnicas, com o objetivo de aumentar a acurácia obtida.
7. Avaliar o desempenho obtido e escolher o melhor modelo, conjunto de parâmetros e técnicas.

Foi um projeto desafiador e legal de desenvolver. No processo, testei e melhorei muitos conceitos e técnicas que aprendi durante o *nanodegree*. Sobre a escolha do tema, um dos requisitos que defini era explorar domínios fora do meu conhecimento. Isto trouxe um desafio a mais, pois precisei estudar um pouco sobre teoria musical e o que extrair dos dados para treinar os algoritmos.

Encontrar o melhor algoritmo e seus parâmetros foi interessante, considerando também o conjunto de dados reduzido que foi utilizado, o que requereu algumas técnicas para tentar amenizar esta situação.

Melhorias

Alguns pontos de melhoria foram identificados:

- Utilizar mais dados para treinar os modelos, focando principalmente nas notas que tiveram menor acurácia.
- Aplicar técnicas mais avançadas de pré-processamento dos dados, de forma a remover características indesejadas. Como não tenho muita experiência em lidar com áudio, pode ser que existam formas melhores de pré-processar as entradas, resultando em dados de melhor qualidade para treinar os modelos.
- Utilizar melhores equipamentos de produção e captura de áudio, visto que utilizei o que tinha disponível e, certamente, gravações realizadas em estúdios apropriados com melhores equipamentos resultariam em um aumento na qualidade dos dados.
- Tentar utilizar deep learning para verificar se o modelo se aplicaria melhor ao problema quando comparado com os modelos utilizados neste estudo. Para aplicar este modelo, eu teria que estudá-lo mais a fundo, e provavelmente precisaria de mais dados para treinamento.