Insper

Ciência dos Dados

Engenharia

Vitor Grando Eller
André Emilinik de Weber
Henry Furquim Rocha

Estudo da Aplicabilidade de Modelos de Aprendizado de Máquina em Classificação de Áudios.

Projeto 3

São Paulo
2018

Vitor Grando Eller
André Emilinik de Weber
Henry Furquim Rocha

Estudo da Aplicabilidade de Modelos de Aprendizado de Máquina em Classificação de Áudios.

Projeto 3 apresentado aos professores da disciplina de Ciência dos Dados, do INSPER, como parte dos requisitos necessários para aprovação na matéria.

São Paulo
2018

Resumo

O objetivo do projeto é analisar algoritmos de classificação para aplicá-los em áudios de sons urbanos relacionados à 40 classes.. Dessa forma, utilizamos as bibliotecas, pandas, numpy, matplotlib, librosa, scipy e scikit-learn.

O dataset foi criado através da análise de cada áudio com 52 atributos estatísticos de 7 features ao todo, listadas na área de desenvolvimento na seção de “features”, e em seguida calculada a acurácia de modelos como KNeighbors, RandomForestClassifier e GradientBoostingClassifier, explicados na seção de “classificadores”.

Palavras-chave: Machine Learning. Classificação de Sons. Classificadores.

Lista de ilustrações

Lista de quadros

Lista de gráficos

Lista de diagramas

Sumário

1 Bibliotecas Usadas

1.1 Librosa

A biblioteca librosa: tem como enfoque a análise de músicas e áudios. Foi criada para ter uma alta relação com numpy, scipy e MATLAB, facilitando no uso.

A utilização dela no projeto foi crucial, pois assim carregamos os áudios a partir da função librosa.load(). Assim como a extração das features usadas, librosa.feature.feature_name(

1.2 Pandas

A biblioteca pandas foi utilizada basicamente para manusear os dataframes de treino e teste.

1.3 Numpy

A biblioteca numpy também foi de extrema importância, pois através de suas funções conseguimos fazer a transformação de Fourier [np.fft.fft(audio)] e obter uma melhor an. Os valores como média, mediana, percentil, entre outros, também foram obtidos a partir da biblioteca.

1.4 MatPlotLib

Utilizamos a extensão Pyplot da biblioteca Matplotlib para a elaboração de gráficos

1.5 Scikit-Learn

Utilizamos a biblioteca Scikit-learn para a construção dos nossos modelos preditivos, além de utilizá-la para processar nossos dados de treinamento e obter as predições dos dados de teste. Também utilizamos sua funçao de acurácia afim de obter as acurácias de nossos modelos.

2 Processo

Em primeiro lugar, encontramos nossa base de dados no Kaggle[1]. A base de dados consitia de uma coleção com 9400 arquivos de áudio urbano, cada qual pertencente a uma classe específica. Ao todo, tratavam-se de 41 classes possíveis.

Pelo fato de os arquivos de áudio ocuparem um tamanho considerável da memória e não serem passíveis de serem enviados para o GitHub, escolhemos por extrair suas features localmente (com base em um código criado por nós e disponível em nosso repositório) e salvá-los em um arquivo próprio.

Tendo esses fatos anteriores em mente, podemos analisar o processo. O processo se iniciou com o carregamento de todos áudios a partir da base de dados, obtendo então a onda e o SampleRate (dado 44100Hz previamente), que é a quantidade de amostras do sinal coletadas em função do tempo. Em seguida, foi feita a Transformada de Fourier para decompor a onda em partes de senoides, assim em vez de guardar uma alta quantidade de dados, obtém-se a frequência que será usada para recriar a onda original e fornece uma análise alternativa em função desta frequência e não do tempo.

Após a Transformada conseguimos obter as features[2] e seus valores. Extraímos informações em cima de Fourier, harmônico, percussivo, spectral centroid, spectral flatness, spectral contrast. A partir delas foram obtido os valores: mediana, média, primeiro percentil, segundo percentil, amplitude interquartil, mínimo, máximo, desvio padrão e batidas por minuto.

Depois de organizar o dataset e normalizar os dados, foram calculadas as acurácias com três modelos para comparação: KNeighbors, RandomForest, GradientBoosting.

[1] https://www.kaggle.com/c/freesound-audio-tagging/data
[2] Tais features serão descritas na próxima seção.

3 Features

Tendo que nossa base de dados consistia apenas de áudios *crus*, precisávamos tratá-los e extrair features (propriedades) do mesmo. Escolhemos trabalhar com 7 features, sendo elas:

1) **Fourier** – Informações sobre a frequência da onda original.
2) **Harmônico**[1] – Frequência de vibração com propriedade de causar ressonância. Correspondem às notas e escala musical.
3) **Percussivo**[2] – Sons e sinais obtidos através de impacto, raspagem ou agitação.
4) **Spectral Centroid** (centróide espectral) – Indica onde o centro de massa do espectro está localizado. (Calculado com a média ponderada das frequências).
5) **Spectral Flatness** (nivelamento espectral) – Medido em decibéis. Indica o quanto o som se aproxima de ser barulho puro. Quantifica "noise-like"
6) **Spectral Contrast** (contraste espectral) – Como o próprio nome já diz, representa uma distribuição relativa espectral e suas características em um áudio.
7) **Mel-frequency Cepstral Coefficients** - Uma representação do espectro de potência de curto prazo de um som, baseado em uma Transformada cosseno linear de fourier, que não envolve números complexos, de um espectro de potência de log em uma escala de mel não linear de freqüência.

[1] As características das harmônicas do áudio foram obtidas após realizar a Transformada de Fourier no áudio que continha somente as harmônicas.
[2] As características dos percussivos do áudio foram obtidas após realizar a Transformada de Fourier no áudio que continha comente os percussivos.

4 Classificadores

4.1 K Nearest Neighbors

Este classificador se baseia no conceito de vizinhança. Ou seja, classifica um novo elemento baseado em elementos 'vizinhos'. Isso se dá através da análise das características de seus vizinhos, sendo a quantidade de vizinhos definidas por nós. Após isso, a classe com mais membros na vizinhança definirá a classe desse novo elemento.

Para estabelecer uma constante ideal, é necessário observar a curva de taxa de erro em função do valor de K, baseada no Dataset de Treino, e a curva de validação de erro em função de K, baseada no Dataset de Teste.

Gráfico 1 – Curva da Taxa de Erro do Valor-K para o Dataset de Treino

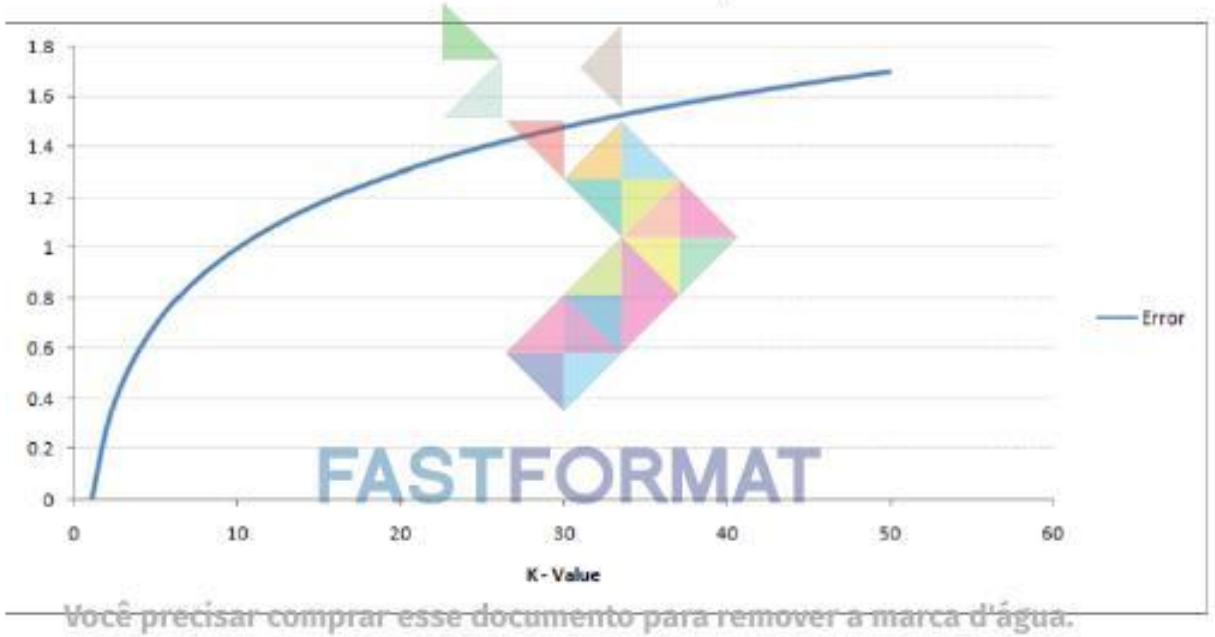
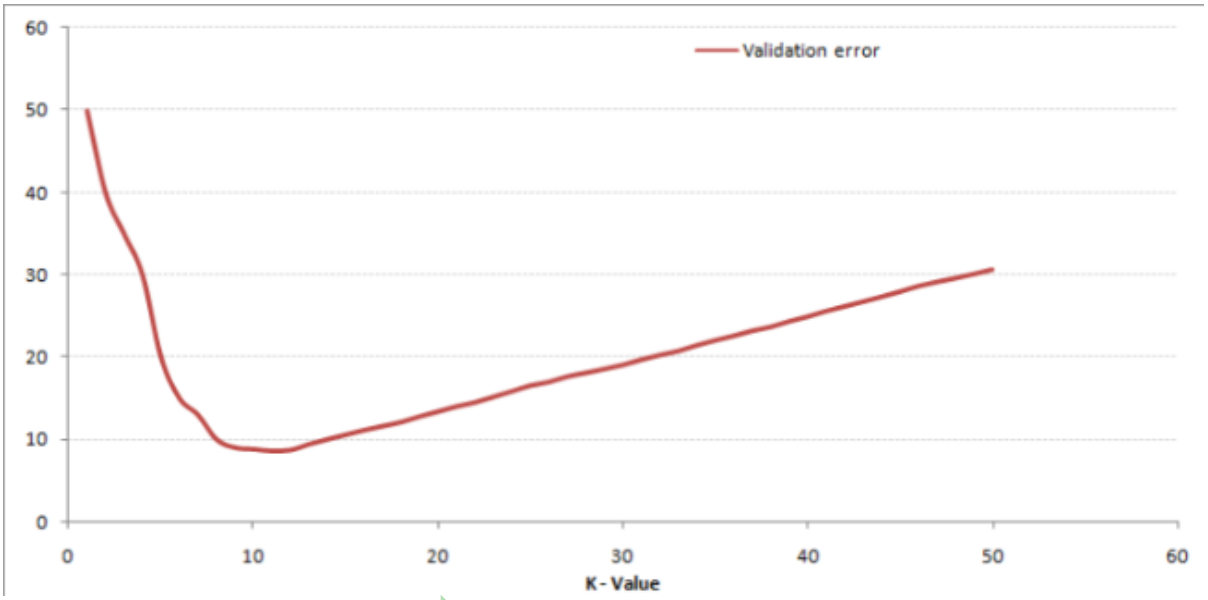


Gráfico 2 – Curva de Validação de Erro do Valor-K para o Dataset de Teste



Para o melhor K ser encontrado, utilizamos o método de 10-fold cross validation. Assim, dividimos o dataset de treino em 10 grupos semelhantes. A primeira partição é representada como validação, então o modelo é aplicado ao resto das divisões. A taxa de classificação incorreta é computada. O processo então é repetido 10 vezes, cada situação uma divisão é considerada a validação. No final, resulta em 10 estimativas de erro de teste que levam ao cálculo da média. Tal média resultou em um valor-K de 10.

Quadro 1 – Código Utilizado para o Método *Fold Cross Validation*

```
from sklearn.model_selection import cross_val_score
neighbors = list(range(1,50))

cv_scores = []

for k in neighbors:
    knn = KNeighborsClassifier(n_neighbors=k)
    scores = cross_val_score(knn, x_train, y_train, cv=10, scoring='accuracy')
    cv_scores.append(scores.mean())


MSE = [1 - x for x in cv_scores]

optimal_k = neighbors[MSE.index(min(MSE))]
print(optimal_k)
```

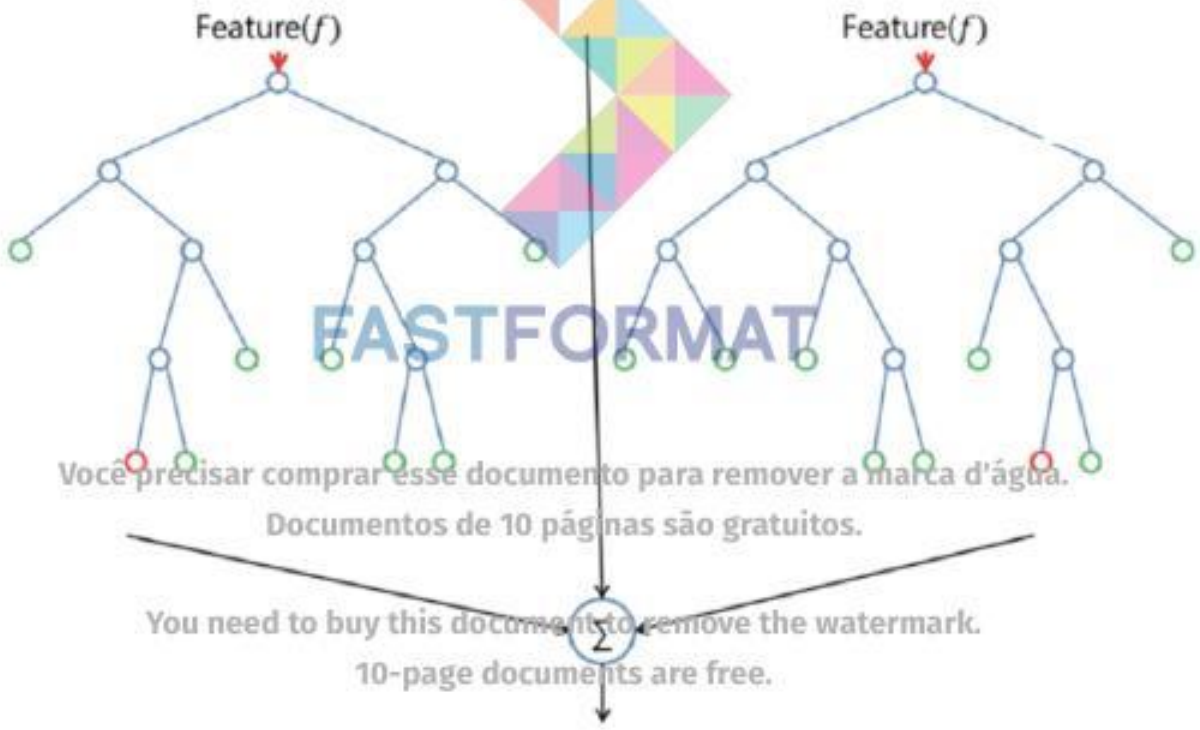
4.1.1 Parâmetros Específicos

- **n_neighbors:** O parâmetro n_neighbors define o número de vizinhos na vizinhança a ser analisada. Obtido através do método *Fold Cross Validation*.
- **weights:** O parâmetro weights define qual função será usada para ponderar as informações obtidas. Para nosso modelo, escolhemos o valor *distance*[1].

4.2 Random Forest

Este classificador basicamente se baseia em um modelo que combina inúmeras decisões por "árvore". Assim, estabelece diversas combinações com resultados como predições que são fundidos e estabelece uma predição final, que pode ser considerada como a média de todas. O algoritmo aplica uma aleatoriedade nas combinações e nas features, e posteriormente seleciona as melhores.

Diagrama 1 – Exemplo de Random Forest com Duas Árvores



4.2.1 Parâmetros Específicos

- **max-depth:** O parâmetro define a profundidade máxima de uma árvore. Para nosso modelo, utilizamos o valor 20.

[1] O valor *distance* pondera os valores baseado no inverso de suas distâncias. Dessa forma, um vizinho mais próximo tem mais influência do que um vizinho que se encontra mais distante do novo elemento.

- **n_estimators:** O parâmetro n_estimators define o número de árvores na floresta. Como possuímos diversas classes e muitas features, utilizamos um valor mais alto, de 100.

- **criterion:** O parâmetro define qual função será usada para obter a qualidade de uma divisão. Em nosso modelo, utilizamos o valor *entropy*[2].

4.3 Gradient Boosting

A lógica por trás deste classificador está em parte na regressão linear, em que a soma dos resíduos é zero. Porém, se houver diferentes de zeros, é possível que haja padrões nesses resíduos que podem ser promovidos para melhorar o modelo. Assim, o processo é feito até que todos resíduos fiquem aleatoriamente distribuídos. Chegando neste ponto, não é preciso mais modelar eles.

Basicamente o classificador produz classificações de preditivos fracos e realiza combinações por árvores de decisão e assim as combina por ensemble learning, selecionando as melhores predições.

O **Gradient Boosting** possui três elementos base: Função de Perda, Weak Learner, Additive Model.

- **Função de Perda (loss function):**
  - A função de perda depende do problema a ser resolvido. No caso de classificação, se usa perda logarítmica. Esta funçao mensura a qualidade da predição do modelo.

- **Weak Learner**
  - Combinação de Árvores de Decisão. As regressões feitas produzem valores que são adicionados permitindo também a adição da saída dos modelos subsequentes para minimizar os resíduos da predição.

- **Additive Model**
  - As árvores existentes não são alteradas e outras são adicionadas com o tempo no modelo, usando o método G*radient Descent*, minimizando a perda.

[2] O valor *entropy* define que a função utilizada avaliará o ganho de informação após cada divisão.

4.3.1 Parâmetros Específicos:

Por não conseguir entender a fundo a função dos parâmetros do Gradient Boosting, optamos por deixa-los todos com valores padrão.

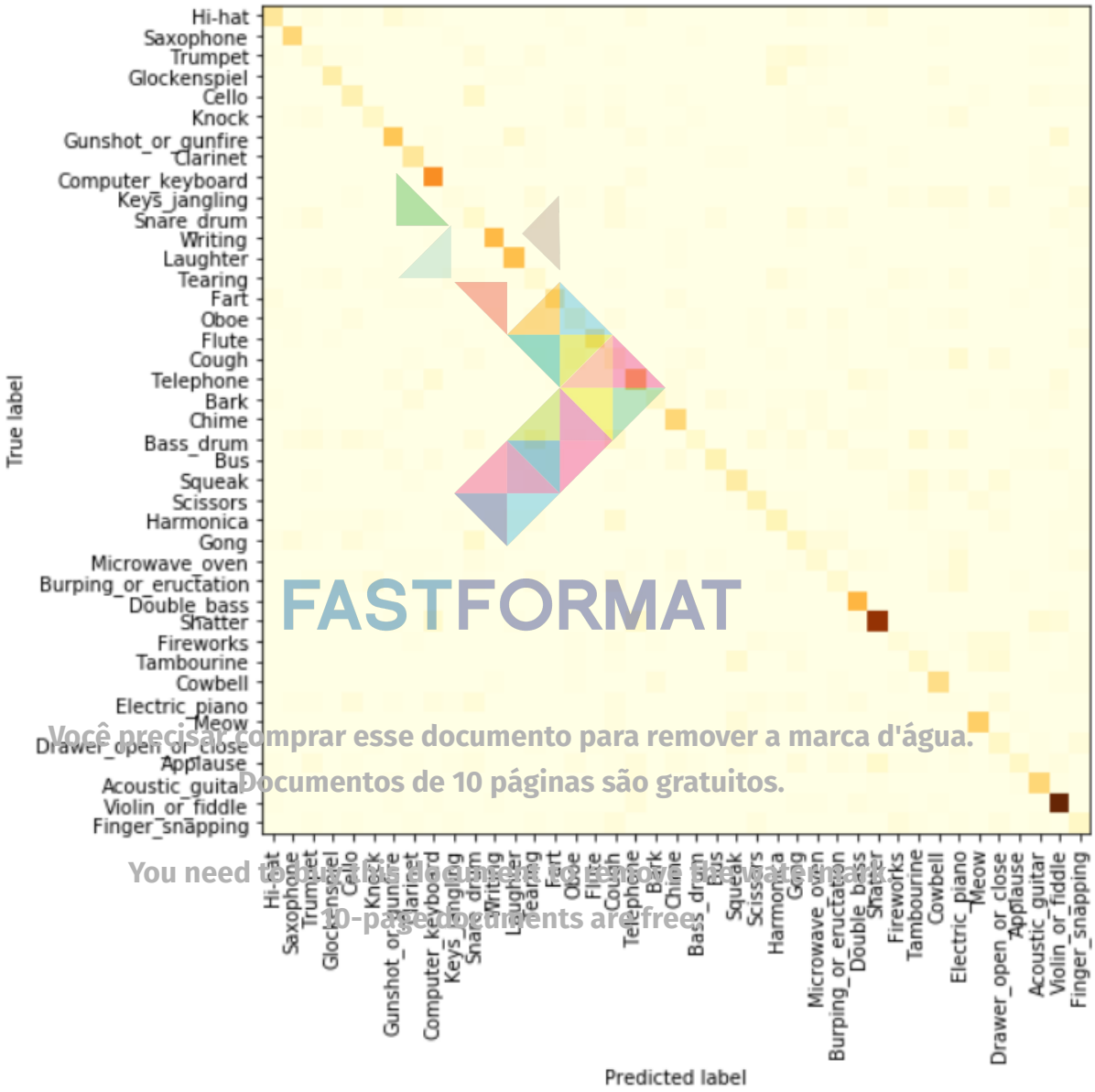
5 Resultados Obtidos

5.1 Acurácia dos modelos

5.1.1 K-Nearest

O classificador apresentou uma acurácia de aproximadamente $53,5\%$, considerando o valor-K encontrado no *10-fold cross validation.*

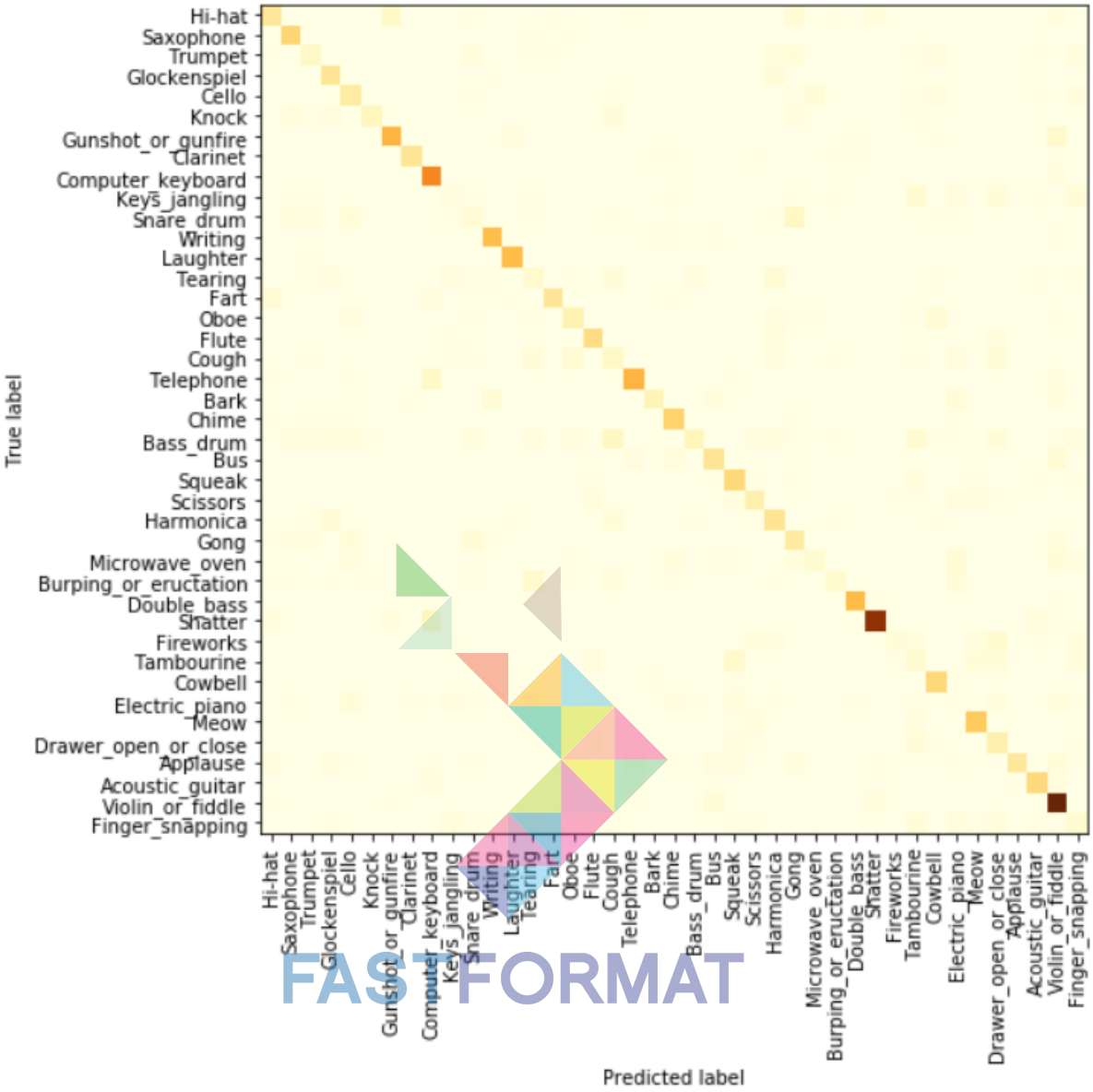
Figura 1 – Matriz de Confusão Obtida a Partir das Predições do Modelo K-Nearest



5.1.2 Random Forest

O classificador apresentou uma acurácia de 61.87%.

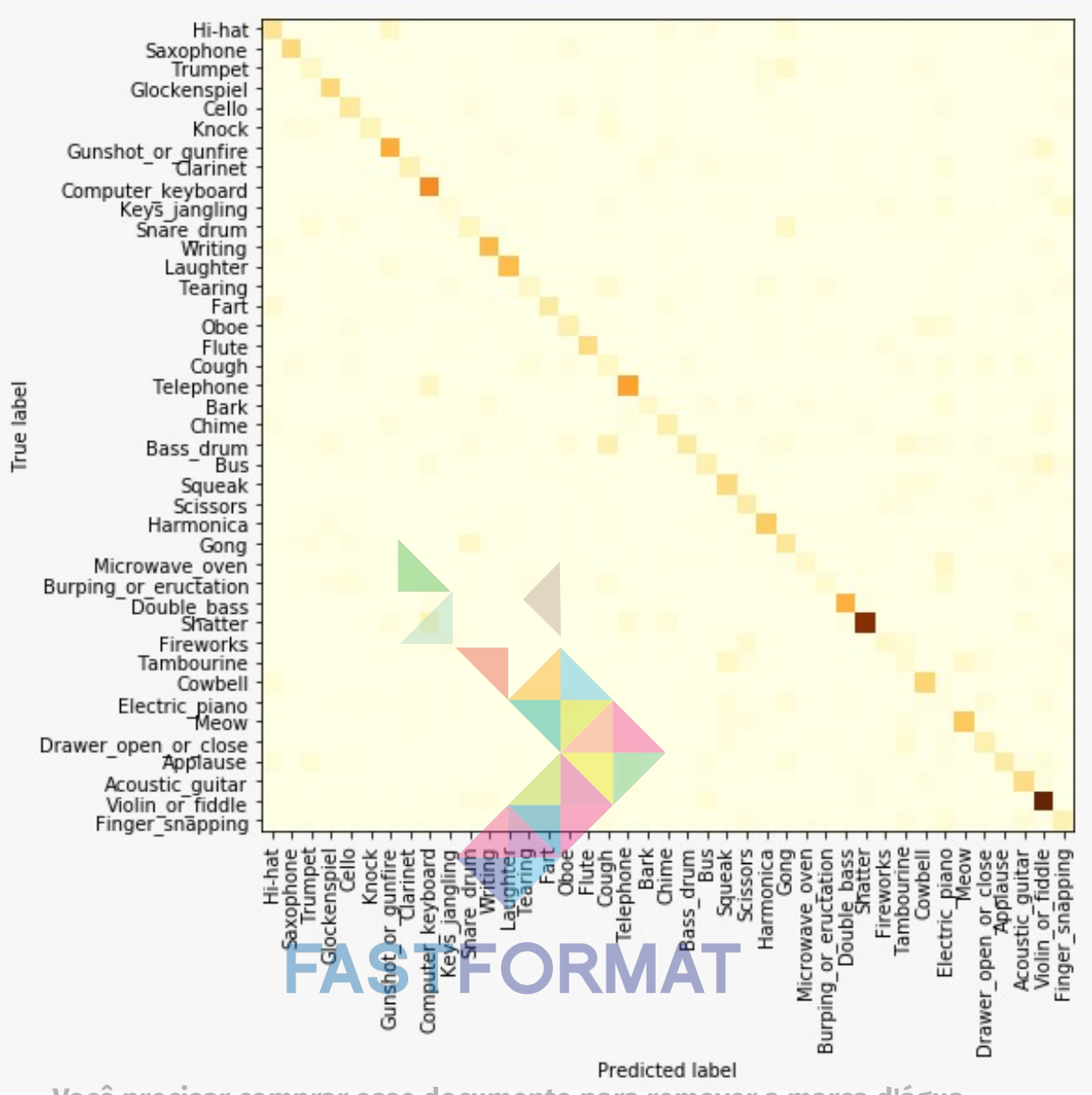
Figura 2 – Matriz de Confusão Obtida a Partir das Predições do Modelo Random Forest



5.1.3 Gradient Boosting

O classificador apresentou uma acurácia de $57,49\%$.

Figura 3 – Matriz de Confusão Obtida a Partir das Predições do Modelo Gradient Boosting



5.2 Resultado

Considerando a quantidade de features utilizadas nos primeiros modelos (58 atributos obtidos a partir de 7 features), o grupo considerou o desempenho abaixo do esperado. Portanto, resolveu realizar uma análise para averiguar se existia um overfitting no modelo.

O esperado para um modelo de *Machine Learning* é que ele não seja 100% certo quanto à classificação do Dataset de Treinamento, já que isso significaria que o modelo 'decorou' as características do Dataset de Treinamento.

Com isso, averiguamos qual seria a acurácia do modelo quando predizendo nossas amostras de treinamento.

Quadro 2 – Teste de Acurácia da Predição das Amostras de Treinamento

```
y_rf_train = rf_model.predict(x_train)
rf_acc_train = acc(y_train, y_rf_train)
rf_acc_train
```

```
1.0
```

A partir do teste, pudemos perceber que sim, houve um overfit de features. Com isso, o grupo decidiu por realisar um processo de *Feature Selection* para visualizar a possibilidade de melhora do classificador.

5.3 Feature Selection

Para o processo, decidimos por estudar somente o classificador *Random Forest*, dado que foi o modelo com melhor acurácia nos estudos anteriores.

O processo de *Feature Selection* visa estudar qual a importância de cada feature quando da utilização do modelo. A partir disso, podemos escolher as features com maior importância e recriar o modelo apenas com elas, buscando uma melhora no classificador.

Para realizar o processo, criamos um regressor logístico que averiguava quais features possuiam importância maior que zero quando utilizadas para classificar apenas uma classe dentre as 40 possíveis. Feito isso, escolhemos os regressores que obtiveram acurácia acima dos 40%.

Tendo tais regressores disponíveis, pudemos obter quais as features eles utilizavam, e captá-las para nosso novo modelo preditivo, agora com menos features. Esse processo resultou em uma redução de quase 50% de nossas features[1]. Apesar disso, a acurácia do classificador foi reduzida para $59,56\%$.

O grupo não foi capaz de entender o por que do fato ter acontecido.

[1] Features foram reduzidas de 58 para 27.

6 Próximos Passos

Para o futuro do nosso modelo, o grupo acredita que seria interessante realizar uma análise mais profunda dos atributos de cada modelo, podendo então aperfeiçoa-lo. Além disso, pensamos que seria benéfico identificar features melhores e mais amplas para extrair melhores informações de cada áudio.

Um último, e mais avançado, ponto, seria a implementação de um modelo de Deep Learning. O grupo resolveu por não seguir em frente com essa opção pois acreditou que não teria conhecimento suficiente para entender e conseguir explicar o funcionamento do modelo.

Apêndices

APÊNDICE A – Classes Possíveis para os Áudios

1) Violão Acústico
2) Aplausos
3) Latido
4) Bumbo
5) Arroto
6) Ônibus
7) Violoncelo
8) Clarineta
9) Teclas de Computador
10) Tossido
11) Sino de Vaca
12) Abertura de Porta
13) Teclado Musical
14) Flatulência
15) Estalar de Dedos
16) Fogos de Artifício
17) Flauta
18) Cantoria
19) Contrabaixo
20) Glockenspiel
21) Gongo
22) Arma de Fogo
23) Harmonica
24) Chacoalhar de Chaves
25) Batida em Porta

26) Risada

27) Miado

28) Microondas / Forno

29) Oboe

30) Saxofone

31) Tesouras

32) Tamborim

33) Choro

34) Telefone

35) Trompeta

36) Violino

37) Escrita

38) Tarola

39) Estilhaçar de Vidros

40) Hi-Hat (equipamento de Bateria)

41) Rangi

APÊNDICE B – Referências

Stava, Tavish. Introduction to k-Nearest Neighbors: Simplified (with implementation in Python). Disponível em: <www.analyticsvidhya.com/blog/2018/03/introduction-k-neighbours-algorithm-clustering/>. Acesso em: 26 Mar. 2018.

Donges, Niklas. The Random Forest Algorithm. Disponível em: <www.towardsdatascience.c random-forest-algorithm-d457d499ffcd/> . Acesso em: 22 Fev. 2018.

Zakka, Kevin. A Complete Guide to K-Nearest-Neighbors with Applications in Python and R. Disponível em: <www.kevinzakka.github.io/2016/07/13/k-nearest-neighbor/>. Acesso em: 13 Jul. 2016.

Stava, Tavish. Introduction to Random forest – Simplified. Disponível em: <www.analyticsvidh random-forest-simplified/> Acesso em: 10 Jun. 2014.

Grover, Prince. Gradient Boosting from scratch. Disponível em: <www.medium.com/mlreview boosting-from-scratch-1e317ae4587d/>. Acesso em: 9 Dez. 2017.

Gorman, Ben. A Kaggle Master Explains Gradient Boosting. Disponível em: <www.blog.kaggle.com/2017/01/23/a-kaggle-master-explains-gradient-boosting/>. Acesso em: 23 Jan. 2017.

Brownlee, Jason. A Gentle Introduction to the Gradient Boosting Algorithm for Machine Learning. Disponível em: <www.machinelearningmastery.com/gentle-introduction-gradient-boosting-algorithm-machine-learning/>. Acesso em: 9 Set. 2016.

Rogozhnikov, Alex. Gradient Boosting explained [demonstration]. Disponível em: <www.http://arogozhnikov.github.io/2016/06/24/gradient_boosting_explained.html>. Acesso em: 24 Jun. 2016.

Martins, Calors. O que é a transformada de Fourier?. Disponível em: <www.abertoatedemad que-e-transformada-de-fourier.html>. Acesso em: 7 Dez. 2013.

An Interactive Guide To The Fourier Transform. Disponível em: <www.betterexplained.com/a interactive-guide-to-the-fourier-transform/>.

Nam, Unjung, Special Area Exam Part II. 2001. 10f

Cook, John. Spectral Flatness. Disponível em: <www.johndcook.com/blog/2016/05/03/spect flatness/>. Acesso em: 2 Mai. 2016.

2002 IEEE International Conference on Multimedia and Expo. 2002. 21f. Conferência - Instituto federal de tecnologia da Suíça, Lausanne, 2002.

Mel Frequency Cepstral Coefficient (MFCC) tutorial. Disponível em: <www.practicalcryptogr learning/guide-mel-frequency-cepstral-coefficients-mfccs/>.