Relatório do Projeto — Mineração de Dados

Nathalia Tescarollo Gonçalves e Renan Baisso

9 de dezembro de 2019

1. Introdução

Este projeto tem como objetivo desenvolver um classificador capaz de fazer predições sobre gravações de diferentes caracteres.

A base de dados, sobre a qual as análises foram feitas, é formada por arquivos de áudio (extensão .wav), sendo que cada arquivo contem uma sequência de quatro caracteres. Cada caractere foi gravado separadamente e em ordem aleatória, utilizando uma ferramenta de gravação. A duração de cada gravação individual é de dois segundos, e os caracteres possíveis são: a,b,c,d,h,m,n,x,6,7.

Os arquivos de treinamento foram extraídos do diretório "TREINAMENTO", e os arquivos de teste foram extraídos do diretório "VALIDAÇÃO", ambos disponibilizados no Tidia da disciplina Mineração de Dados. A partir dessa base, então, foram realizados a segmentação e identificação das sequências de caracteres e o tratamento dos sinais. Quanto às bibliotecas utilizadas nessa etapa, temos as seguintes:

- **librosa**, tanto para o carregamento como para a extração de *features* dos áudios (*MFCC*, *Mel Frequency Cepstral Coefficients* e *Chroma Energy Normalized Statistics*);
- **scipy**, para a transformação dos sinais de áudio (*DCT*, *FFT* e Filtro de passa baixas de Butterworth);
- **random**, para embaralhar os arquivos da base de forma aleatória;
- **pandas** e **numpy**, para cálculos e manipulação dos dados.

Após realizarmos as devidas transformações na base de dados, aplicamos algoritmos de classificação e avaliamos suas predições (mais detalhes nas seções "Metodologia"e "Resultados"). Nesta etapa, utilizamos as seguintes bibliotecas:

- **sklearn**, tanto para treinar os modelos e realizar as predições como para extrair métricas;
- **matplotlib**, para gerar gráficos das matrizes de confusão dos modelos.

2. Análise Exploratória

Para realização das análises dos áudios, primeiramente, foi feita a segmentação dos áudios seguindo, pois todos os arquivos teriam aproximadamente oito segundos e que cada letra falada estaria em uma janela de dois segundos (exemplo de áudio capturado na Figura 1). Por isso, o áudio foi segmentado em quatro partes para extração de dados que deveriam ser lidos e classificados.

Com os áudios isolados, foi possível começar com as análises necessárias. À princípio, foi pensado em apenas utilizar os áudios em sua forma original, porém, com

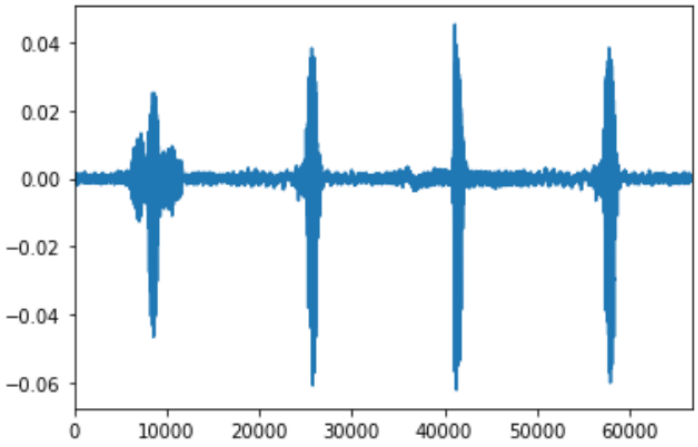

Figura 1. Exemplo de áudio capturado.

um pouco de observação sobre os dados, notou-se que os áudios nem sempre estavam centralizados nas janelas de dois segundos. Por isso, outro tipo de característica devia ser extraída, além apenas das amplitudes no tempo.

Com isso, foi necessário realizar algumas transformações para que os dados no tempo fossem analisados no domínio da frequência. Por isso, primeiramente foram realizadas as Transformadas Rápidas de Fourier (FFT) (Figura 2) e a Transformada Discreta de Cossenos (DCT) (Figura 3).

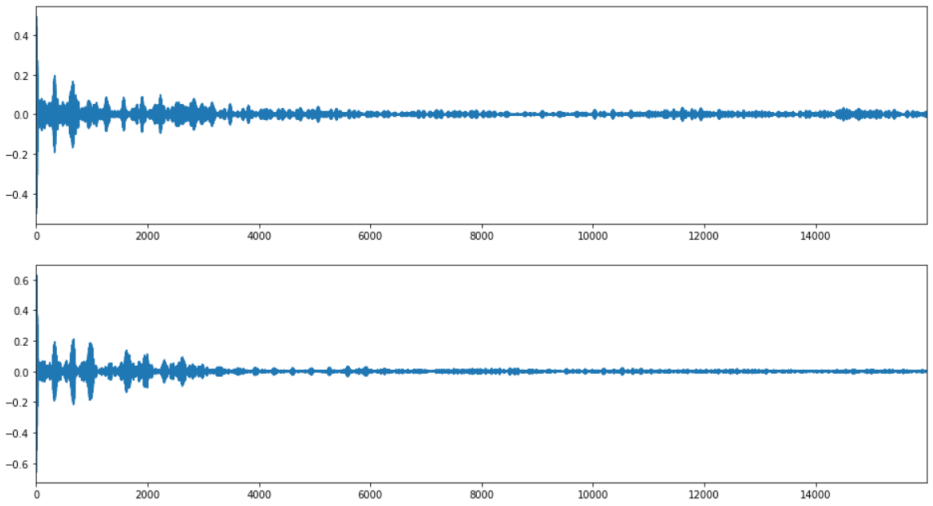

Figura 2. Espectro DCT para letras B.

À partir dos espectros gerados, foi possível verificar que os áudios de mesma letra tinham espectros semelhantes, tanto para FFT quanto para DCT, o que poderiam servir de auxílio para extração de características dos dados. Apesar disso, essas transformações são sensíveis aos ruídos captados na geração dos dados. Sendo assim, foi considerado utilizar um filtro passa baixas de Butterworth para diminuição de ruídos de alta frequência, pois os sons que os humanos naturalmente escutam estão entre 20Hz e 20kHz.

Ainda assim, a FFT e a DCT por si só não levam em conta a característica humana

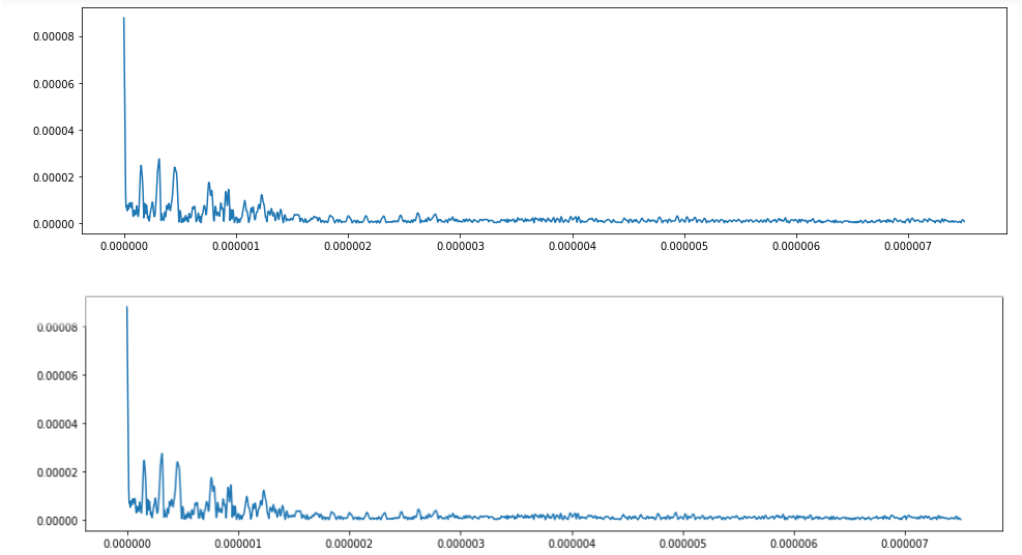

Figura 3. Espectro FFT para letras B.

para resolução de sons. Desta maneira, foram escolhidas outras três transformações de áudio para estas análises. A *Mel Frequency Cepstral (MFC)*, a qual representa um espectro de frequência dentro a *escala de Mel*. A *escala de Mel* realiza transformações no sinal de áudio para representar as frequências mais próxima das quais o ouvido humano consegue identificar. Abaixo, temos o MFC das letras X (Figura 4) e B (Figura 5) representado graficamente. As cores representam a intensidade da frequência em dB.

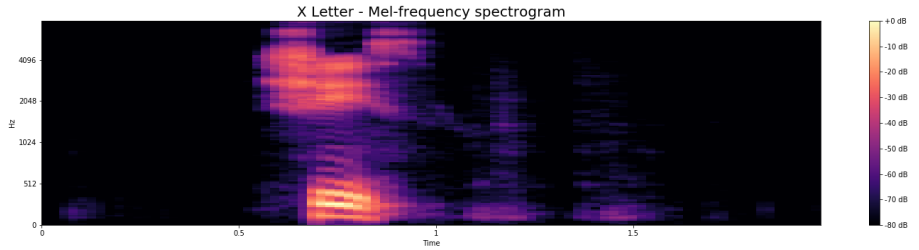

Figura 4. MFC completo da letra X.

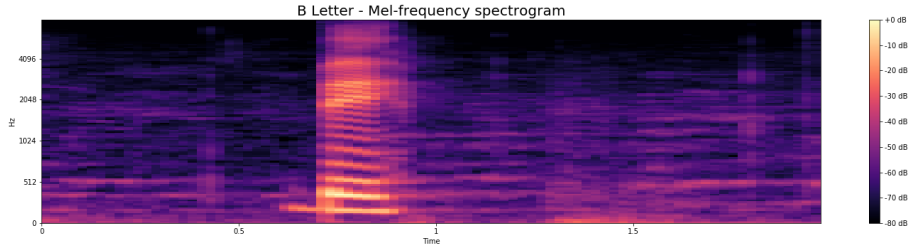

Figura 5. MFC completo da letra B.

Através dos gráficos acima, é possível verificar que esta transformação pode ser promissora para diferenciar os caracteres contidos no sinais de áudio.

A segunda transformação escolhida foi a *Mel Frequency Cepstral Coefficients* (MFCC), que é o cálculo de coeficientes que constroem o MFC. Aparentemente, MFC

e MFCC representam a mesma grandeza, no entanto, o MFCC desmembra o MFC em uma quantidade de coeficientes parametrizada, o que trás uma representação diferente do MFC do mesmo sinal. Abaixo temos a representação ao longo do tempo do espectro total para as letras X (Figura 6) e B (Figura 7), retiradas do *dataset* de estudo. As cores representam a intensidade da frequência.

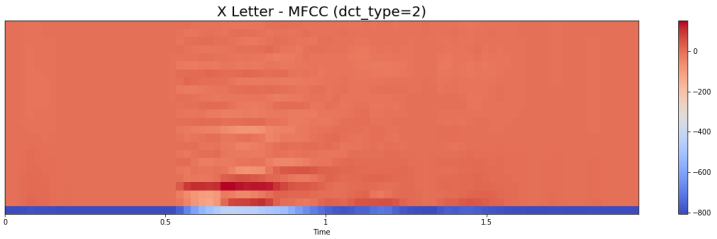

Figura 6. MFCC completo da letra X.

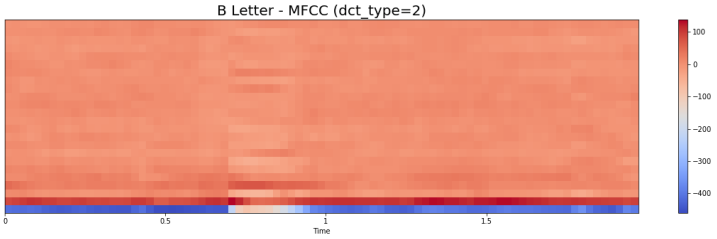

Figura 7. MFCC completo da letra B.

No Python, a implementação do MFCC na biblioteca librosa resulta em um *array* de coeficientes das frequências para cada janela gerada pelo algoritmo. Por isso, normalmente a média entre as janelas é calculada para cada coeficiente. Na Figura 8 podemos observar um mesmo coeficiente para diferentes letras. A linha azul representa a curva para a letra 'x', e as demais para diferentes áudios de letras 'b'.

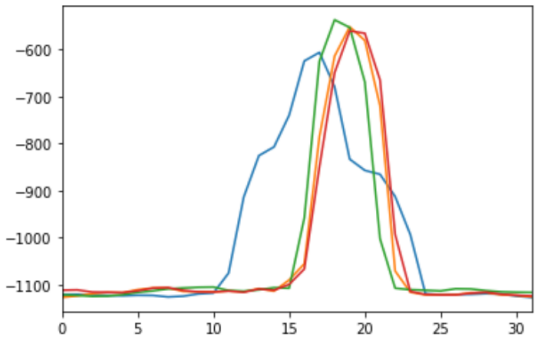

Figura 8. Coeficiente MFCC para as letras X e B.

Outra característica utilizada foi a *Chroma Energy Normalized*, a qual baseia-se nos tons que os humanos percebem e na energia do espectro da frequência, i. e., demonstra a quantidade de energia em cada tom (A, B, C, D, E, F, G), o que facilita a diferenciação de timbres e articulações de fala. Abaixo temos a representação do cromagrama para amostras X (Figura 9) e B (Figura 10) do *dataset*.

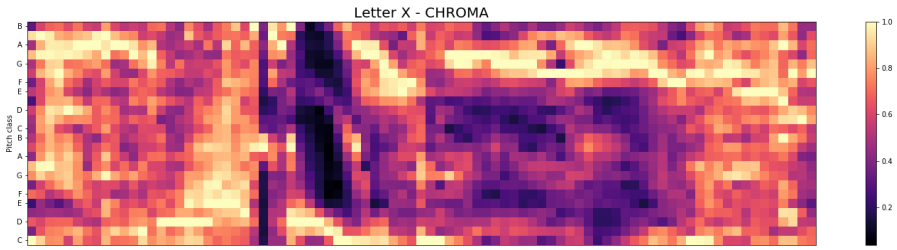

Figura 9. Croma completo da letra X.

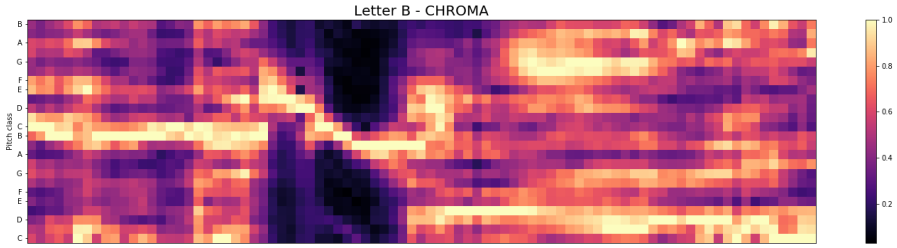

Figura 10. Croma completo da letra B.

Para maiores detalhes dos resultados das análises, existe uma pasta com o nome *"analysis"* no pacote do projeto que contém os *Jupyter notebooks* utilizados para gerar as análises e os resultados que serviram de insumos para esta sessão.

3. Metodologia

Foram considerados principalmente dois algoritmos de classificação, a fim de obtermos dados para comparação e definição do melhor algoritmo para o problema proposto. Abaixo encontram-se breve explicações sobre cada um deles, além de detalhes sobre as escolhas de implementação feitas pelo grupo:

- ***Support Vector Machine* (SVM), ou *Support Vector Classification* (SVC) -** Algoritmo originário da teoria do aprendizado estatístico, muito utilizado para problemas de classificação. Realiza a separação dos dados com base em hiperplanos de margem máxima, i.e., hiperplanos com a maior distância entre elementos de diferentes classes. Essa abordagem busca manter a qualidade da predição para exemplos não vistos anteriormente. Para essa análise, instanciamos um modelo de SVC com *kernel* linear.
- ***Random Forest* –** Algoritmo pertencente a uma técnica de classificação conhecida como *ensemble* (em português, "comitê"), a qual agrega as predições de múltiplos classificadores para chegar em um "consenso" e melhorar a acurácia da classificação final. Métodos do tipo *ensemble* tendem a ter um desempenho melhor do que qualquer classificador singular.
  Duas importantes decisões a serem tomadas ao trabalhar com *Random Forest* estão relacionadas ao número de classificadores/árvores e à profundidade máxima dos mesmos. Para a primeira parte do projeto, conforme recomendado em aula, utilizamos 500 classificadores (parâmetro *n_estimators*) como padrão de instanciação. Já a profundidade máxima (parâmetro *max_depth*) foi definida como 50.
  Para a segunda entrega, foi o utilizado o *Grid Search CV* (GSCV) para buscar os melhores hiper-parâmetros para a *Random Forest*. O GSCV realiza uma busca

exaustiva pela combinação dos melhores parâmetros dados pelo usuário. Nos testes realizados, os melhores parâmetros encontrados para a *Random Forest* variaram dependendo das *seeds* geradas para fazer o embaralhamento (*shuffle*) dos arquivos de áudio. Um exemplo de conjunto de parâmetros obtidos pelo GSCV foi: *'bootstrap'* = True, *'max_depth'* = 40, *'n_estimators'* = 1833.

Quanto à metodologia de avaliação, treinamos os modelos e consideramos a acurácia (*score*) das predições feitas para a base de validação, a nível de caracteres, como métrica para avaliar seus respectivos desempenhos. A acurácia, também conhecida como taxa de acerto, calcula a porcentagem de amostras que foram classificadas corretamente.

Além disso, geramos as matrizes de confusão dos modelos de duas formas. Ao executar o programa ("*project.py*"), as matrizes de confusão (em representação numérica) de cada modelo são impressas na tela e, logo em seguida, são salvas, no mesmo diretório, imagens de seus respectivos gráficos de cores.

Por último, é importante ressaltar que os testes foram feitos utilizando o Sistema Operacional Ubuntu (Linux) e a versão 3.6 da linguagem de programação Python. Para executar o programa, é necessário passar um argumento indicando se o modo de execução será ou não de *teste* (1 ou 0, respectivamente).

4. Resultados

Dentre os algoritmos de classificação mencionados, e considerando a metodologia de avaliação adotada, o modelo que obteve melhores resultados foi o *Random Forest*.

Em relação aos valores de acurácia obtidos, nos testes realizados a acurácia do modelo SVC variou de aproximadamente 58% a 61%. Já o modelo *Random Forest* apresentou acurácia de aproximadamente 73% a 74%. A Tabela 1 representa o relatório de saída do programa, para o modelo *Random Forest* com a máxima acurácia obtida nos testes realizados pelo grupo, de 74,34%.

Caractere	Precision	Recall	F1-Score	Support
6	0.73	0.79	0.76	100
7	0.95	0.95	0.95	114
a	0.97	0.94	0.95	97
b	0.53	0.62	0.57	106
c	0.79	0.69	0.74	107
d	0.56	0.53	0.54	110
h	0.94	0.96	0.95	122
m	0.50	0.54	0.52	104
n	0.60	0.50	0.54	113
x	0.88	0.94	0.91	95
accuracy	0.74			1068
macro avg	0.74	0.74	0.74	1068
weighted avg	0.74	0.74	0.74	1068

Tabela 1. Relatório de execução *Random Forest*.

Além disso, observando a Tabela 1, pode-se avaliar o balanço de outras métricas do modelo *Random Forest* para cada caractere. Por exemplo, para o caractere ‘x’, notamos que há uma tendência maior de classificar outras letras como ‘x’, pois sua precisão está menor do que sua revocação, o que resulta na diminuição do f1-score. O mesmo ocorre para o caractere ‘6’. Outra observação interessante está relacionada aos caracteres que tiveram pior resultado, como os pares (‘m’, ‘n’) e (‘b’, ‘d’), dado que na língua portuguesa os elementos de cada um desses pares soam semelhantes entre si.

Comparando o gráfico das matrizes de confusão dos modelos SVC e *Random Forest* (Figura 11 e Figura 12, respectivamente) conseguimos ver facilmente que o *Random Forest* foi o que teve melhor desempenho, visto que a concentração de predições na diagonal principal da matriz é maior para ele do que para o SVC. Nesta representação, a diagonal principal da matriz diz respeito aos elementos classificados corretamente.

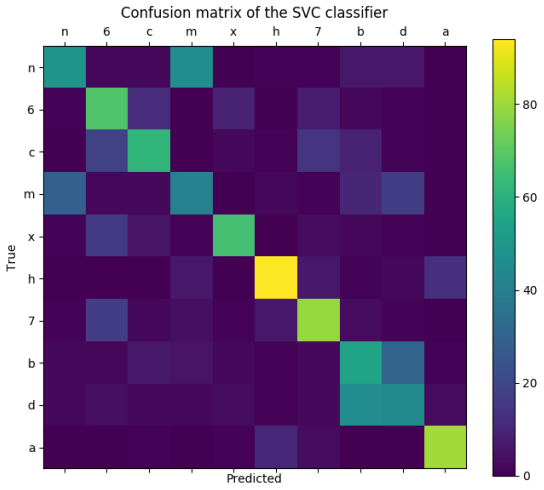

Figura 11. Exemplo de Matriz de Confusão do modelo SVC.

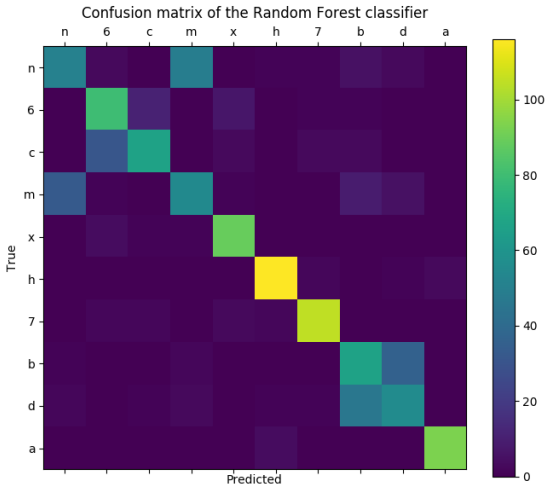

Figura 12. Exemplo de Matriz de Confusão do modelo *Random Forest*.

5. Comentários Finais

Com relação a primeira entrega, houveram evoluções no projeto em relação ao tratamento de dados, extração de *features*, as quais, nesta entrega, estão mais consistentes. Isso é de grande importância em projetos de Mineração de Dados, já que, se o entendimento dos dados não é correto, o problema, seja de classificação, regressão ou otimização, não terá uma solução satisfatória. Além disso, foram aplicados conceitos de validação cruzada, os quais auxiliaram na otimização dos modelos de aprendizado de máquina testados no projeto. A adoção desta técnica resultou em um melhor desempenho médio do modelo de *Random Forest*, em termos de acurácia.

Ainda assim, destaca-se a dificuldade de processamento de validações cruzadas do tipo de busca por exaustão, pois buscas desse tipo consomem alto processamento. Também houve tentativa do uso de seleção de características utilizando o *Grid Seach CV*, porém não obteve-se sucesso pela demora de processamento e, muita das vezes, pela falha dos computadores ao tentar processar por longos períodos.

Por fim, o projeto ainda possui melhorias a serem realizadas, pois nem todas as técnicas de mineração de dados para evolução dos modelos e melhor seleção de *features* foram utilizadas.