Transformada de Fourier em Reconhecimento de Fala

Sinais e Sistemas (ES413) – Cin/UFPE

2024.1

Alunos(as): Aline Fortaleza Ferreira da Silva (affs2)
Beatriz Galhardo Carneiro Leao (bgcl)
Danilo Lima de Carvalho (dlc3)
Luisa Fonseca Leiria de Andrade (lfla)
Rafael Alves de Azevedo Silva (raas)

1- Dataset

É notório que a escolha do dataset é fundamental para o desenvolvimento do projeto. A precisão, generalização e desempenho do modelo dependem diretamente da qualidade dos dados fornecidos.

O dataset escolhido foi: DataSet, nele contém várias amostras que estão em uma estrutura de dados em que usamos: o vetor (áudio), a taxa de amostragem (intervalo no qual os componentes do vetor foram capturados) e o rótulo (qual palavra foi dita no aúdio). O conjunto de dados está dividido em três partes: teste, treinamento e validação. Para o projeto, foi necessário o uso das duas primeiras partes, apenas.

Inicialmente, foi utilizado o dataset completo, porém, para facilitar a execução do código, foram selecionadas as 10 primeiras classes e 2900 amostras para cada rótulo selecionado, isso foi necessário para não acontecer um desbalanceamento entre as classes no processo de treinamento.

2 - Pré-processamento

Em projetos de processamento de sinais de áudio, o pré-processamento é uma etapa crucial. Ele garante que o sinal de áudio esteja em um formato consistente e adequado para processamento, preparando os dados brutos para as etapas seguintes de análise e extração de características. As etapas principais do pré-processamento do projeto atual incluem normalização, pré-ênfase e trim (recorte do silêncio).

Figura 1 - Código do pré-processamento

```python
def pre_process_audio(audio, sr, top_db=20):
    """Aplica normalização, pré-ênfase e trim ao áudio."""
    audio = librosa.util.normalize(audio)
    audio = librosa.effects.preemphasis(audio, coef=0.97)
    audio, _ = librosa.effects.trim(audio, top_db=top_db)
    return audio
```

2.1 - Normalização:

A normalização ajusta a amplitude dos sinais de áudio para que todas as amostras estejam dentro de um intervalo específico. Isso é importante para evitar que variações na intensidade do áudio afetem o desempenho dos algoritmos usados em seguida.

No código, a normalização define que a amplitude do sinal de áudio nunca passe do valor máximo de 1. Isso garante que todos os sons estejam na mesma escala de amplitude, o que facilita as comparações e análises subsequentes.

Essa etapa evita que os sons com volumes diferentes influenciem os resultados do modelo proposto, promovendo o mesmo tratamento a todas as amostras.

2.2 - Pré-Ênfase:

A fase de pré-ênfase usa um filtro no sinal de áudio para aumentar as frequências mais altas. Isso é feito para compensar a natural atenuação dessas frequências na maioria dos sinais de áudio, particularmente nos sinais de fala.

No código, o filtro de pré-processamento foi aplicado por meio da fórmula $y[t] = x[t] - \alpha x[t-1]$, com α sendo um coeficiente de pré-ênfase com valor usual de 0.95. Como esse filtro aumenta as frequências altas, ele ajuda a destacar as características que podem ser mais relevantes nas análises.

A aplicação da pré-ênfase melhora a nitidez do sinal, destacando os componentes de alta frequência, que são essenciais para tarefas como reconhecimento de fala.

2.3 - Trim

O Trim ou recorte de silêncio remove áreas do sinal que não contêm som, como silêncios antes ou depois do som relevante. Isso é útil para remover as partes inúteis do sinal e limitar a análise aos bits relevantes.

Na implementação, o trim foi realizado identificando e removendo as seções de baixa energia (e consequentemente, de silêncio) do início e do final do sinal de áudio. Isso é feito comparando a energia de cada fração com um valor de referência predeterminado.

Esse recorte reduz a quantidade de dados a serem processados, diminuindo o tempo da análise e garantindo que as características extraídas são de conteúdo relevante do áudio. Além disso, melhora a precisão do modelo de aprendizado de máquina usado em seguida, já que evita que partes irrelevantes do áudio influenciem os resultados.

3- Extração das características

A etapa crucial do processamento de sinais de áudio é a extração de características. O objetivo dessa etapa é converter o sinal bruto em uma representação pequena e informativa que possa ser usada em tarefas de análise e aprendizado de máquina. No projeto atual, várias características do sinal de áudio foram extraídas e capturadas, cada uma capturando diferentes características do som. Zero Crossing Rate (ZCR), Short-Time Fourier Transform (STFT), Mel-Frequency Cepstral Coefficients (MFCC), Chroma Feature, e Spectral Contrast foram as principais técnicas utilizadas para extrair características.

3.1 - Zero Crossing Rate (ZCR)

Zero Crossing Rate (ZCR) é uma medida que quantifica a taxa na qual o sinal de áudio cruza o eixo zero (passa de positivo para negativo ou vice-versa) durante certo período de tempo.

No código, a função librosa.feature.zero_crossing_rate() é aplicada para calcular o ZCR. Isso cria uma série temporal que tem a taxa de cruzamentos por zero ao longo do sinal.

O ZCR é útil para distinguir sons percussivos e harmônicos. A taxa de cruzamento por zero é mais baixa para sons como vozes, enquanto é mais alta para ruídos ou sons percussivos.

Figura 2 - Código do ZCR

```
# Zero Crossing Rate
zero_crossings = librosa.feature.zero_crossing_rate(y=audio, hop_length=hop_length)
zcr_mean = np.mean(zero_crossings)
zcr_std = np.std(zero_crossings)
```

3.2 - Short-Time Fourier Transform (STFT)

O método STFT divide o sinal de áudio em segmentos menores e usa a Transformada de Fourier para dar uma representação tempo-frequência.

Na implementação, a STFT foi calculada usando a função librosa.stft(), e a magnitude do espectro foi obtida aplicando np.abs() ao resultado da STFT. Logo depois, foram extraídas várias estatísticas, como a média, variância, mediana, desvio padrão, mínimo e máximo das magnitudes.

A STFT é fundamental para analisar como as frequências de um sinal de áudio mudam ao longo do tempo, sendo particularmente útil para caracterizar sons que possuem variações temporais consideráveis.

Figura 3 - Código do STFT

```
# Short-Time Fourier Transform (STFT)
stft = np.abs(librosa.stft(audio, n_fft=n_fft, hop_length=hop_length))

# STFT statistics
fft_mean = np.mean(stft, axis=1)
fft_var = np.var(stft, axis=1)
fft_median = np.median(stft, axis=1)
fft_std = np.std(stft, axis=1)
fft_min = np.min(stft, axis=1)
fft_max = np.max(stft, axis=1)
```

3.3 - Mel-Frequency Cepstral Coefficients (MFCC)

MFCCs são coeficientes que representam a forma espectral de um sinal de áudio em uma escala de frequência Mel, que é baseada na percepção auditiva humana.

No código, foi usado a função librosa.feature.mfcc() para extrair os MFCCs, ela calcula os coeficientes a partir do espectro de potência do sinal após mapeá-lo para a escala Mel. Foram calculadas várias estatísticas desses coeficientes, como a média, desvio padrão, mínimo e máximo.

Os MFCCs são extremamente utilizados em reconhecimento de fala e outras aplicações de processamento de áudio, porque capturam as características relevantes do som.

Figura 4 - Código do MFCC

```
# Mel-Frequency Cepstral Coefficients (MFCC)
mfccs = librosa.feature.mfcc(y=audio, sr=sr, n_mfcc=n_mfcc, n_fft=n_fft, hop_length=hop_length)
mfcc_mean = np.mean(mfccs, axis=1)
mfcc_std = np.std(mfccs, axis=1)
mfcc_min = np.min(mfccs, axis=1)
mfcc_max = np.max(mfccs, axis=1)
```

3.4 - Chroma Feature

As *Chroma Features* agrupam a energia do espectro de áudio em 12 bins correspondentes às 12 notas da escala musical ocidental.

No código, as *Chroma Features* foram calculadas usando a função librosa.feature.chroma_stft(), que mapeia a energia espectral do sinal para as classes de pitch correspondentes. Estatísticas como média, desvio padrão, mínimo e máximo dessas energias foram então calculadas.

Essas *Features*, são especialmente úteis para análise de música, porque permite a identificação de padrões harmônicos - como os acordes e progressões de notas.

Figura 5 - Código de Chroma Feature

```
# Chroma Feature
chroma = librosa.feature.chroma_stft(y=audio, sr=sr, n_fft=n_fft, hop_length=hop_length)
chroma_mean = np.mean(chroma, axis=1)
chroma_std = np.std(chroma, axis=1)
chroma_min = np.min(chroma, axis=1)
chroma_max = np.max(chroma, axis=1)
```

3.5 - Spectral Contrast

O Spectral Contrast mede a diferença entre os picos e vales do espectro em diferentes bandas de frequência, refletindo a distribuição da energia espectral ao longo dessas bandas.

Na implementação, a função librosa.feature.spectral_contrast() foi utilizada para calcular o Spectral Contrast. Foram extraídas várias estatísticas dessas bandas, incluindo a média, o desvio padrão, os valores mínimos e máximos.

O contraste de espectro é especialmente útil na distinção de sons com diferentes características de timbre e textura. Ele é essencial para análises de áudio, pois tem poder de identificar gêneros musicais ou classificar diferentes tipos de instrumentos.

Figura 6 - Código do Spectral Contrast

```
# Spectral Contrast
spectral_contrast = librosa.feature.spectral_contrast(y=audio, sr=sr, n_fft=n_fft, hop_length=hop_length)
contrast_mean = np.mean(spectral_contrast, axis=1)
contrast_std = np.std(spectral_contrast, axis=1)
contrast_min = np.min(spectral_contrast, axis=1)
contrast_max = np.max(spectral_contrast, axis=1)
```

3.6 - Concatenação

Realiza a combinação de várias características extraídas anteriormente do sinal de áudio em um único vetor de características. Isso produz uma representação compacta e detalhada do áudio, que pode ser utilizada para análise ou treinamento de modelos de aprendizado de máquina(no nosso caso, o modelo de regressão logística).

Figura 7 - Código da concatenação

```
# Combine all features into a single feature vector
combined_features = np.concatenate((
    [zcr_mean, zcr_std],
    fft_mean, fft_var, fft_median, fft_std, fft_min, fft_max,
    mfcc_mean, mfcc_std, mfcc_min, mfcc_max,
    chroma_mean, chroma_std, chroma_min, chroma_max,
    contrast_mean, contrast_std, contrast_min, contrast_max
))
```

4- Treinamento

A fase de treinamento é essencial no processo de aprendizado de máquina. Ela determina a capacidade do modelo de aprender a partir dos dados e fazer previsões precisas em novos conjuntos de dados. Um treinamento bem conduzido resulta em modelos robustos e eficazes, prontos para serem implementados em aplicações práticas. Antes da realização do treinamento foi feita uma normalização das características adquiridas de cada áudio:

Figura 8 - Código da Normalização

```
# Normalizar os dados, o vetor inteiro dos dados das características
scaler = StandardScaler()
X_train_scaled = scaler.fit_transform(X_train)
X_test_scaled = scaler.transform(X_test)
```

As características extraídas (como ZCR, STFT, MFCC, Chroma, e Spectral Contrast) são usadas como entrada para o modelo de aprendizado. Durante o treinamento, o modelo ajusta seus parâmetros internos para maximizar a precisão das previsões com base nos dados conhecidos. O algoritmo reconhecedor de padrões escolhido foi a Regressão Logística, que é um modelo estatístico usado para prever a probabilidade de um determinado evento ocorrer.

No contexto deste projeto, o algoritmo foi utilizado para classificar sinais de áudio em diferentes categorias, com base nas características extraídas. O modelo foi treinado utilizando as características combinadas dos sinais de áudio como variáveis preditoras e as respectivas classes de áudio como a variável alvo. O processo de treinamento envolve ajustar os coeficientes de cada característica para que o modelo possa prever com precisão a classe a que cada exemplo de áudio pertence.

Figura 9 - Código da Regressão Logística

```python
# Criar e treinar o modelo de Regressão Logística
modelo_regressao_logistica = LogisticRegression(max_iter=1000)
print("\nTreinando e avaliando o modelo Regressão Logística com dados não processados")
modelo_regressao_logistica.fit(X_train_scaled, y_train)
y_pred = modelo_regressao_logistica.predict(X_test_scaled)
accuracy = accuracy_score(y_test, y_pred)
print(f"Acurácia do modelo Regressão Logística: {accuracy * 100:.2f}%")
print(classification_report(y_test, y_pred))
```

O treinamento foi realizado utilizando a função LogisticRegression da biblioteca scikit-learn. O parâmetro max_iter=1000 foi ajustado para garantir que o algoritmo tivesse iterações suficientes para convergir para uma solução estável, especialmente em casos onde os dados podem ser complexos.

Os coeficientes resultantes do modelo indicam a força e a direção do impacto de cada característica sobre a probabilidade de pertencimento a uma classe específica. Um coeficiente positivo indica que um aumento na característica aumenta a probabilidade da classe alvo, enquanto um coeficiente negativo indica o oposto. Compreender os coeficientes ajuda a interpretar quais características são mais influentes na decisão do modelo, fornecendo insights valiosos sobre os padrões presentes nos dados de áudio.

5- Resultados

Após o treinamento do modelo de regressão logística, um conjunto de dados de teste foi usado para avaliar a eficácia do modelo. A avaliação foi feita com base em várias métricas de desempenho, incluindo precisão, recall, f1-score e acurácia, para cada uma das classes do problema.

Figura 10 - Resultados Gerados:

```
Treinando e avaliando o modelo Regressão Logística com dados não processados
Acurácia do modelo Regressão Logística: 76.22%
              precision    recall  f1-score   support

           0       0.87      0.90      0.89       419
           1       0.66      0.73      0.69       405
           2       0.77      0.78      0.78       425
           3       0.73      0.69      0.71       406
           4       0.72      0.72      0.72       412
           5       0.83      0.75      0.79       396
           6       0.76      0.78      0.77       396
           7       0.76      0.77      0.77       402
           8       0.83      0.82      0.82       411
           9       0.71      0.68      0.69       402

    accuracy                           0.76      4074
   macro avg       0.76      0.76      0.76      4074
weighted avg       0.76      0.76      0.76      4074
```

O modelo alcançou uma acurácia geral de 76,22%. Ou seja, em média o modelo classificou 76,22% dos exemplos de maneira correta.

As principais métricas de avaliação usadas para cada classe foram:

- Precisão (Precision): Mede a proporção de exemplos classificados corretamente como uma determinada classe em relação a todos os exemplos classificados como essa classe.
- Recall: Mede a proporção de exemplos de uma classe específica que foram corretamente identificados pelo modelo.
- F1-Score: A média harmônica entre precisão e recall, proporcionando um equilíbrio entre as duas métricas.
- Support: O número de ocorrências reais de cada classe no conjunto de teste.

As classes são respectivamente:

- Classe 0: Yes
- Classe 1: No
- Classe 2: UP
- Classe 3: Down
- Classe 4: Left
- Classe 5: Right
- Classe 6: On
- Classe 7: Off
- Classe 8: Stop
- Classe 9: Go

Interpretando os resultados, observa-se que as classes com menor precisão são a 1 e a 9. Podemos notar, que a fonética das duas palavras são semelhantes, o que gera uma troca nos resultados (onde áudios com a label “go” foram classificados como “no” por exemplo), diminuindo assim a acurácia do modelo nessas duas classes e consequentemente da média como um todo. Isso mostra que o modelo não é sensível a amostras semelhantes e que precisa ser otimizado para essas situações.