دانشکدگان فنی دانشگاه تهران

یادگیری ماشین

گزارش پروژه پایانی

(پیش‌پردازش، استخراج ویژگی صوتی، طبقه‌بندی و خوشه‌بندی)

نام افراد گروه:

امید ملایی ۸۱۰۱۰۳۲۴۱

نادیه محمدی ۸۱۰۱۰۳۳۳۸

فاطمه صدری ۸۱۰۱۰۲۰۲۷

دی ۱۴۰۳

فهرست مطالب

۱ مقدمه‌ای بر روش انجام کار

۱_۱ مقدمه

در این پروژه، با دقت فراوان تلاش کردیم تا در هر مرحله، ابتدا تحلیل و ارزیابی‌های دقیق را با استفاده از نمودارهای توزیع در فایل‌هایNotebook (پوشه appendix) انجام دهیم. سپس با توجه به نتایج حاصل شده و اهمیت شفافیت کار، سورس کد پروژه را به طور کامل و دقیق تدوین کردیم.

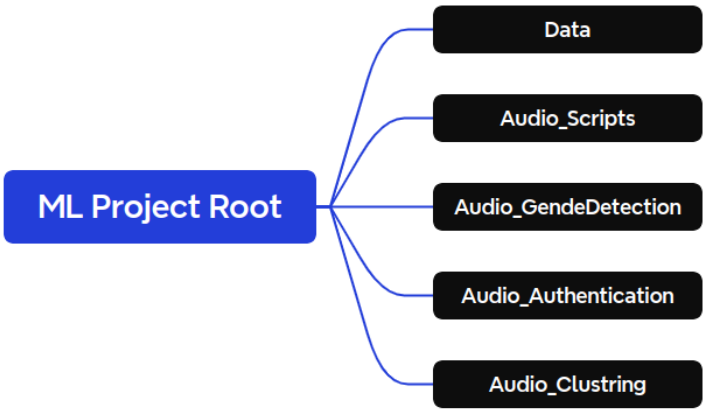
آنچه در تکمیل این گزارش بسیار مهم و تعیین‌کننده است، نتایج، توابع و نمودارهای موجود در پوشه appendix می‌باشد که به عنوان پایه و اساس این گزارش و سورس کد عمل می‌کنند. این عناصر کلیدی، نقشی اساسی در صحت و دقت نتایج نهایی ایفا می‌نمایند و اهمیت ویژه‌ای در روند اجرای پروژه دارند.

تاکید ما بر فایل‌های Notebook بوده است تا تمامی جزئیات و مراحل کار به طور کامل و با دقت مستند شوند، به گونه‌ای که هیچ نکته‌ای از قلم نیفتد و تمامی جوانب پروژه با وضوح و شفافیت بیشتری منعکس شوند.

۱_۱_۱ ساختار پروژه و سورس‌کد

برای سورس کد پروژه با تکیه بر نتایج حاصل از گزارشات Notebook و ارزیابی‌های پایه، ما با استفاده از الگوریتم و روش‌های مختلف، بطور کلی سه هدف را دنبال کرده‌ایم: تشخیص جنسیت از طریق صوت، احراز هویت فرد گوینده و خوشه‌بندی داده‌های صوتی. در هر یک از این مسائل، روش‌ها و الگوریتم‌های مختلفی که در ادامه به توضیح آن‌ها می‌پردازیم، استفاده شده است. داده‌های خام اولیه استفاده شده برای هر یک از این سه مسئله در سورس‌کد یکسان بوده و تنها از داده‌های در اختیار داده شده استفاده شده است. در این

سورس‌کد، هر تسک خواسته شده، شامل تشخیص جنسیت، احراز هویت و خوشه‌بندی، در فولدر مخصوص خود قرار گرفته است.



ساختار کلی سورس‌کد:

Data

- در این پوشه، داده‌های خام اولیه مورد نیاز برای پروژه قرار گرفته است. تعداد فایل‌های صوتی در اختیار داده شده ۴۷۳ فایل صوتی و مجموعا حدود ۳۵.۵ ساعت صوت می‌باشد. بطور کلی مشخصات داده‌های خام اولیه به این صورت می‌باشد:

- داده‌های خام اولیه پروژه می‌بایستی در مسیر Data/raw قرار گیرند.

- تعداد فایل‌های صوتی: ۴۷۳

- تعداد فایل‌های قابل استفاده: ۴۴۹ (شماره دانشجویی و یا جنسیت برخی صداها قابل شناسایی از روی نام فایل نمی‌باشد)

- تعداد فایل‌های صوتی مربوط به آقایان: ۳۲۷

- تعداد فایل‌های صوتی مربوط به خانم‌ها: ۱۱۲

- مجموع زمان همه‌ی صوت‌های قابل استفاده: ۳۵.۴ ساعت

- مجموع زمان همه‌ی صوت‌های آقایان: ۲۵.۶۵ ساعت

- مجموع زمان همه‌ی صوت‌های خانم‌ها: ۹.۶۸ ساعت

- تعداد گویندگان متمایز: ۱۱۴ نفر

- تعداد گویندگان آقا: ۸۶ نفر

- تعداد گویندگان خانم: ۲۸ نفر

Audio_Scripts

کدهای خاص هر مسئله در پوشه مربوط به خود آن مسئله قرار گرفته‌اند، اما برخی از کدها که در ادامه توضیح خواهیم داد در تمامی مسائل به نحوی مورد استفاده قرار گرفته‌اند. این کدها در پوشه ml_project/Audio_Scripts قرار داده شده‌اند.

Audio_GenderDetection

کدهای مربوط به مسئله تشخیص جنسیت در این مسیر قرار گرفته‌اند.

Audio_Authentication

کدهای مسئله احراز هویت در این مسیر قرار گرفته‌اند.

Audio_Clustering

کدهای خوشه‌بندی صداها در این مسیر قرار گرفته‌اند.

کتابخانه‌های مورد استفاده:

برای اجرای این پروژه از کتابخانه‌های مختلفی (فایل requirement.txt) استفاده شده است که برخی از آن‌ها به این صورت می‌باشند:

- scikit-learn: برای پیاده‌سازی الگوریتم‌های مختلف در این پروژه و همچنین ارزیابی‌های مختلف از عملکرد مدل‌ها.

- pandas: برای ذخیره‌سازی ویژگی‌های استخراج شده از فایل‌های صوتی و اعمال تغییرات بر روی آن‌ها.

- Torch: برای پیاده‌سازی شبکه‌های عصبی.

- matplotlib: برای رسم نمودارهای مختلف در پروژه.

- librosa: برای خواندن فایل‌های صوتی از روی دیسک و همچنین استخراج مشخصات فایل‌های صوتی شامل مدت زمان، تعداد کانال‌های صوتی، نرخ نمونه‌برداری و ...

- soundfile: برای ذخیره صوت بر روی دیسک.

۱_۱_۲ چالش‌های دیتاست

در این پروژه سعی بر این بوده است که از روش‌های مختلف تدریس شده استفاده شود و تا جای ممکن عملکرد و دقت مدل‌ها مناسب باشد. یکی از موارد چالش‌ساز در تمامی بخش‌های پروژه، متعادل نبودن داده‌ها است. همانطور که قبل‌تر اشاره شد، تعداد صداهای آقایان تقریباً ۳.۵ برابر خانم‌ها بوده است(برای متوازن کردن داده آنچه برای تحلیل اولیه ما انجام دادیم برای یکی شدن تعداد نمونه‌ها این است که، از صدای هر خانم ۳ نمونه ۳ ثانیه‌ای جدا کردیم و ویس هر آقا یک نمونه ۳ ثانیه‌ای). همچنین نام برخی از فایل‌ها به دلیل قابل استخراج نبودن شماره دانشجویی و یا جنسیت، برای برخی مسائل مناسب نبوده و ناچاراً برخی از فایل‌ها برای برخی از مسائل در نظر گرفته نشده‌اند.

در درون پروژه گاهی بارها نیاز به اطلاعات جزئی و کوچکی بوده است که استخراج این اطلاعات در قالب پیاده‌سازی یکسری توابع در مسیر و فایل ml_final_project/Audio_Scripts/audio_utils.py انجام شده است. در ادامه به بررسی توابع نوشته شده در این فایل می‌پردازیم.

توابع:

- convert_filenames_to_lowercase: این تابع به عنوان ورودی یک مسیر فولدر را گرفته و نام تمامی فایل‌های درون آن فولدر را به همان نام مشابه ولی با حروف کوچک تبدیل می‌کند. این بدین دلیل است که مشاهده شده است برخی از دانشجویان از کلماتی مثل Male به جای male و یا Female به جای female

استفاده کرده‌اند. ما این تابع را بر روی پوشه داده‌های خام اعمال کرده‌ایم تا اسامی تمامی فایل‌ها از یک قالب پیروی کند.

- raw_audio_files: این تابع نیز مسیر داده‌های خام را دریافت نموده و اسامی همه‌ی فایل‌های درون این فولدر را بر می‌گرداند. خروجی این تابع دو لیست می‌باشد: لیست اول اسامی فایل‌هایی می‌باشد که یا شماره دانشجویی آن‌ها و یا جنسیت آن‌ها قابل تشخیص نبوده است و لیست دوم، لیست تمامی فایل‌هایی می‌باشد که نامگذاری آن‌ها قالب استاندارد را دارد.

- extract_speaker_id: این تابع فایل‌های صوتی مربوط به هر دانشجویی را برمی‌گرداند. خروجی این تابع بدین صورت می‌باشد:

{ شماره دانشجویی ۱: [صوت‌های مربوط به این شماره دانشجویی],

شماره دانشجویی ۲: [صوت‌های مربوط به این شماره دانشجویی],

... }

- extract_gender: این تابع جنسیت گوینده صوت را از نام فایل استخراج می‌کند.

- extract_speaker_idx: استخراج شماره دانشجویی.

- get_audio_info: این تابع نام یک فایل صوتی را گرفته و مشخصات آن را شامل فرکانس، زمان و تعداد کانال‌ها را بر می‌گرداند.

۲ مرحله پیش‌پردازش / Preprocessing

با توجه به اینکه در تمامی مسائل نیاز به پیش پردازش داده‌ها وجود دارد ، کد مربوط به بخش پیش پردازش داده‌های در فایل و مسیر ml_project/Audio_Scripts/preprocessing.py نوشته شده است.

پیش‌پردازش داده‌ها در پروژه‌های احراز هویت و تشخیص جنسیت از اهمیت بالایی برخوردار است، زیرا کیفیت داده‌ها مستقیماً بر دقت و عملکرد مدل‌های تشخیصی تأثیر می‌گذارد. با انجام مراحل پیش‌پردازش مانند حذف نویز، نرمال‌سازی بلندی صدا و حذف سکوت، داده‌های صوتی بهینه‌سازی می‌شوند و اطلاعات

زائد و نامرتبط از بین می‌رود. این مراحل به کاهش خطاها و افزایش دقت مدل‌ها کمک می‌کند، به‌ویژه در پروژه‌های حساس مانند احراز هویت و تشخیص جنسیت. علاوه بر این، پیش‌پردازش مناسب باعث می‌شود تا مدل‌ها بتوانند الگوها و ویژگی‌های مهم را بهتر شناسایی و استخراج کنند، که نهایتاً به بهبود کارایی و اثربخشی سیستم‌های تشخیص منجر می‌شود. در ادامه به بررسی مراحل مهم و اصلی در پیش‌پردازش می‌پردازیم.

۲_۱ طراحی فیلتر باندپاس

استفاده از فیلتر باندپاس، نویز صدا و فرکانس‌های خارج از محدوده مشخص شده را حذف می‌کند تا تنها بخش‌های مورد نظر صدا عبور کنند. که در نتیجه باعث بهبود کیفیت صدا میشود. اطلاعات صوتی تمیز و بدون نویز باعث می‌شود تا مدل‌های تشخیص هویت و جنسیت دقیق‌تر عمل کنند.

در ادامه به بررسی کدی که با استفاده از فیلتر باندپاس عمل حذف نویز را روی داده انجام می‌دهد را بررسی میکنیم:

```
from scipy.signal import butter, filtfilt

def denoise_speech_bandpass(audio_data, sr, lowcut, highcut, order=5):
  # Calculate Nyquist frequency
  nyq = 0.5 * sr

  # Normalize cutoff frequencies
  low = lowcut / nyq
  high = highcut / nyq

  # Design bandpass filter
  b, a = butter(order, [low, high], btype='band')

  # Apply filter
  denoised_audio = filtfilt(b, a, audio_data)

  return denoised_audio
```

وارد کردن کتابخانه‌های مورد نیاز:

```
from scipy.signal import butter, filtfilt
```

این خط کتابخانه‌های butterو filtfiltرا از scipy.signal وارد می‌کند. این توابع برای طراحی و اعمال فیلترهای دیجیتال استفاده می‌شوند.

تعریف تابع :denoise_speech_bandpass

```
def denoise_speech_bandpass(audio_data, sr, lowcut, highcut, order=5):
```

این تابع برای حذف نویز از داده‌های صوتی استفاده می‌شود. آرگومان‌های آن شامل داده‌های صوتی (audio_data)، نرخ نمونه‌گیری(sr) ، فرکانس قطع پایین(lowcut) ، فرکانس قطع بالا(highcut) ، و مرتبه فیلتر (order) می‌باشد.

محاسبه فرکانس:Nyquist

```
nyq = 0.5 * sr
```

این خط فرکانس Nyquist را محاسبه می‌کند که نصف نرخ نمونه‌گیری است. این فرکانس برای نرمال‌سازی فرکانس‌های قطع استفاده می‌شود.

نرمال‌سازی فرکانس‌های قطع:

```
low = lowcut / nyq
high = highcut / nyq
```

این خطوط فرکانس‌های قطع پایین و بالا را نسبت به فرکانس Nyquist نرمال‌سازی می‌کنند تا در طراحی فیلتر به درستی عمل کنند.

طراحی فیلتر باندپاس:

```
b, a = butter(order, [low, high], btype='band')
```

این خط یک فیلتر باندپاس با استفاده از تابع butterطراحی می‌کند که دارای مرتبه مشخص شده و محدوده فرکانسی نرمال‌سازی شده است

اعمال فیلتر:

```
denoised_audio = filtfilt(b, a, audio_data)
```

این خط فیلتر باندپاس را بر روی داده‌های صوتی اعمال می‌کند و نویزها را حذف می‌کند. تابع filtfilt فیلتر را به صورت مستقیم و معکوس اعمال می‌کند تا فاز داده‌ها به هم نریزد.

۲_۲ نرمال‌سازی صوت

این قسمت برای نرمال‌سازی بلندی صدای داده‌های صوتی به سطح هدف LUFS (Loudness Units Full Scale)استفاده می‌شود. کدی که در ادامه از این بخش مشاهده میکنید بلندی صدای داده‌های صوتی را اندازه‌گیری می‌کند و بلندی صدا را به سطح هدف LUFS (معمولاً -۱۴) تنظیم می‌کند. نرمال‌سازی بلندی صدا به بهبود یکنواختی و کیفیت تجربه شنیداری کمک می‌کند و اطمینان می‌هد از اینکه تمام فایل‌های صوتی برای پخش پایدار و مطمئن در یک سطح بلندی یکسان هستند.

در پروژه‌های احراز هویت و جنسیت، نرمال‌سازی بلندی صدا به دقت و کارایی تشخیص کمک می‌کند.

```
from pyloudnorm import Meter, normalize

def normalize_audio(audio_data, sr,  target_lufs=-14):
  """Normalizes all audio files in a directory to a target LUFS level."""
  # measure the loudness first
  meter = Meter(sr) # create BS.1770 meter
  loudness = meter.integrated_loudness(audio_data)

  # loudness normalize audio to target_lufs dB
  loudness_normalized_audio = normalize.loudness(audio_data, loudness,
target_lufs)
  return loudness_normalized_audio
```

ورود کتابخانه‌های مورد نیاز:

```
from pyloudnorm import Meter, normalize
```

تعریف تابع :normalize_audio

```
def normalize_audio(audio_data, sr, target_lufs=-14):
```

این تابع برای نرمال‌سازی بلندی صدا به سطح هدف LUFS (Loudness Units Full Scale) در داده‌های صوتی استفاده می‌شود. آرگومان‌های آن شامل داده‌های صوتی(audio_data) ، نرخ نمونه‌گیری(sr) ، و سطح هدف LUFS (target_lufs) هستند.

ایجاد متغیر:Meter

```
meter = Meter(sr)
```

این خط یک متر BS.1770 ایجاد می‌کند که برای اندازه‌گیری بلندی صدا استفاده می‌شود.

اندازه‌گیری بلندی صدا:

```
loudness = meter.integrated_loudness(audio_data)
```
این خط بلندی صدای داده‌های صوتی را اندازه‌گیری می‌کند و نتیجه را در متغیر loudnessذخیره می‌کند.

نرمال‌سازی بلندی صدا به سطح هدف:LUFS

```
loudness_normalized_audio = normalize.loudness(audio_data, loudness,
target_lufs)
```

این خط بلندی صدای داده‌های صوتی را به سطح هدف LUFS تنظیم می‌کند. تابع normalize.loudness این کار را انجام می‌دهد و نتیجه را در متغیر loudness_normalized_audioذخیره می‌کند.

و در نهایت داده نرمال شده return می‌شود تا بتوان از آن استفاده کرد.

نکته: هنگام نرمال‌سازی بلندی صدا(Loudness Normalization) ، سطح هدف LUFS (Loudness Units Full Scale)معمولاً کمتر از -۱۰ دسی‌بل (dB) است. سطح پیشنهادی برای LUFS معمولاً -۱۴ دسی‌بل است.

LUFS یک واحد اندازه‌گیری بلندی صدا است که میزان انرژی صوتی را با توجه به چگونگی دریافت توسط گوش انسان اندازه‌گیری می‌کند.

سطح هدف LUFS سطح بلندی صدای مطلوب است که به آن نرمال‌سازی می‌شود. در این پروژه، سطح پیشنهادی -۱۴ دسی‌بل است.

کاربرد:

کنترل کیفیت :نرمال‌سازی به سطح پیشنهادی LUFS به یکنواختی و کیفیت صدا کمک می‌کند.

استانداردسازی :مطمئن شدن از اینکه تمام فایل‌های صوتی در یک سطح بلندی یکسان هستند.

۲_۳ حذف سکوت

در این مرحله سکوت را از داده‌های صوتی حذف می‌کنیم. ابتدا انرژی ریشه‌میانگین‌مربع (RMS) برای هر فریم از داده‌های صوتی محاسبه می‌شود. سپس فریم‌هایی که انرژی RMS آن‌ها بالاتر از مقدار آستانه‌ای مشخص شده است، شناسایی می‌شوند. فریم‌های غیرساکت به عنوان قسمت‌های صوتی انتخاب و به هم

پیوسته می‌شوند تا سکوت از داده‌ها حذف شود. این فرآیند باعث می‌شود داده‌های صوتی موثرتر و جمع و جورتر باشند و دقت تحلیل و پردازش صوتی افزایش یابد، که در پروژه‌های تشخیص هویت و جنسیت بسیار مفید است. در ادامه به بررسی کد می‌پردازیم:

```
def remove_silence(audio_data, sr, threshold=0.05, frame_length=4096,
hop_length=512):
    # Calculate RMS energy for each frame
    rms = librosa.feature.rms(y=audio_data, frame_length=frame_length,
hop_length=hop_length)[0]

    # Find frames above the threshold
    frames_above_threshold = np.where(rms > threshold)[0]

    # Initialize an empty list to store non-silent segments
    non_silent_segments = []

    # Iterate over frames above the threshold
    for i in range(len(frames_above_threshold)):
        # Get start and end samples for the current segment
        start_sample = frames_above_threshold[i] * hop_length
        end_sample = min((frames_above_threshold[i] + 1) * hop_length,
len(audio_data))  # Prevent exceeding audio length

        # Append the current non-silent segment to the list
        non_silent_segments.append(audio_data[start_sample:end_sample])

    # Concatenate all non-silent segments
    if len(non_silent_segments) > 0:
      non_silent_audio = np.concatenate(non_silent_segments)
    else:
      non_silent_audio = audio_data

    return non_silent_audio
```

محاسبه انرژی RMS:

```
rms = librosa.feature.rms(y=audio_data, frame_length=frame_length,
hop_length=hop_length)[0]
```

این خط انرژی ریشه‌میانگین‌مربع (RMS) را برای هر فریم از داده‌های صوتی محاسبه می‌کند.

یافتن فریم‌های بالای آستانه:

```
frames_above_threshold = np.where(rms > threshold)[0]
```

این خط فریم‌هایی را که انرژی RMS آن‌ها بالاتر از مقدار آستانه‌ای است، شناسایی می‌کند.

پیوستن به قسمت‌های غیرساکت:

```
non_silent_segments.append(audio_data[start_sample:end_sample])
```

این خط، فریم‌های غیرساکت را به لیست اضافه می‌کند تا سکوت از داده‌ها حذف شود.

ترکیب قسمت‌های غیرساکت:

```
if len(non_silent_segments) > 0:
  non_silent_audio = np.concatenate(non_silent_segments)
else:
  non_silent_audio = audio_data
```

این خطوط، قسمت‌های غیرساکت را به هم پیوند می‌دهند تا داده‌های صوتی نهایی بدون سکوت ایجاد شود

۲-۴نحوه و ترتیب اعمال توابع روی داده

```
def preprocess_audio(audio_data, sr, duration = 5):
  y_denoised = denoise_speech_bandpass(audio_data, sr, lowcut=100,
highcut=8000, order=6)
  y_normalized = normalize_audio(y_denoised, sr, -14)
  non_silent_audio = remove_silence(y_normalized, sr, threshold=0.05,
frame_length=4096, hop_length=512)
  duration = sr*duration
  if len(non_silent_audio) > duration:
    non_silent_audio = non_silent_audio[0:duration]
  return non_silent_audio
```

حذف نویز :داده‌های صوتی با استفاده از تابع denoise_speech_bandpass، با یک فیلتر باندپاس که محدوده فرکانسی بین ۱۰۰ و ۸۰۰۰ هرتز را عبور می‌دهد، نویز زدایی می‌شوند.

نرمال‌سازی بلندی صدا :داده‌های نویز زدایی شده با استفاده از تابع normalize_audio به سطح بلندی صدای -۱۴ LUFS نرمال‌سازی می‌شوند.

حذف سکوت :سکوت‌ها از داده‌های نرمال‌سازی شده با استفاده از تابع remove_silence حذف می‌شوند.

تنظیم طول داده‌های صوتی :داده‌های صوتی به مدت زمان مشخصی (بر حسب نمونه‌ها) تنظیم می‌شوند. اگر طول داده‌ها بیشتر از مدت زمان مشخص شده باشد، بخشی از ابتدای داده‌ها برش داده می‌شود.

```
duration = sr*duration
if len(non_silent_audio) > duration:
  non_silent_audio = non_silent_audio[0:duration]
```

و در نهایت non_silent_audio خروجی بازگشتی از این تابع خواهد بود.

این مراحل به بهبود کیفیت و کارایی داده‌های صوتی کمک می‌کنند و آنها را برای تحلیل‌های بعدی مانند احراز هویت و تشخیص جنسیت آماده می‌سازند.

۲-۵ ارزیابی

در طول کار با داده‌های صوتی، یک نکته مهم که باید حتماً مد نظر قرار گیرد، ترتیب صحیح اعمال مراحل پیش‌پردازش است. ابتدا باید عمل نرمال‌سازی (normalize) بر روی داده‌های صوتی انجام شود و سپس حذف نویز (denoising) صورت گیرد. این ترتیب به دلایل زیر اهمیت دارد:

- جلوگیری از حذف نمونه‌هایی با ولوم صدای پایین: اگر حذف نویز قبل از نرمال‌سازی انجام شود، نمونه‌هایی که ولوم صدای پایینی دارند، ممکن است در مرحله حذف سکوت (silence removal) حذف شوند.
- حفظ داده‌ها: با نرمال‌سازی اولیه، اطمینان حاصل می‌شود که تمامی نمونه‌ها با سطح ولوم یکسان پردازش می‌شوند، بنابراین حتی صداهای با ولوم پایین نیز در مراحل بعدی حفظ شده و داده‌های مهم از دست نمی‌روند.

این ترتیب مراحل پیش‌پردازش کمک می‌کند تا دقت و کارایی مدل‌های تشخیص صوتی بهبود یابد و از از دست رفتن داده‌های مفید جلوگیری شود.

۳ مرحله استخراج ویژگی

استخراج ویژگی نیز به دلیل استفاده مکرر در تمامی بخش های پروژه در فایلAudio_Scripts/feature_extraction.py نوشته شده است . از طریق متد های نوشته شده در این بخش ، ما حدود ۶۷ ویژگی برای هر صوت می توانیم استخراج کنیم . اما از تمام این ویژگی‌ها به تبع استفاده نخواهیم کرد، بلکه باید با آزمون و خطا ویژگی یا مجموعه ویژگی‌های مناسب را بدست آورد.

مرحله استخراج ویژگی یکی از حیاتی‌ترین بخش‌ها در پروژه‌های احراز هویت و تشخیص جنسیت است. این مرحله شامل شناسایی و استخراج ویژگی‌های مهم از داده‌های خام است که مدل‌های یادگیری ماشین برای آموزش و پیش‌بینی به آنها نیاز دارند. ویژگی‌ها می‌توانند شامل جنبه‌های مختلفی از داده‌ها باشند، مانند الگوهای صوتی، ویژگی‌های فرکانسی، آماری و زمانی. هدف اصلی استخراج ویژگی، کاهش ابعاد داده‌ها و برجسته کردن اطلاعات مهم و مفید است که می‌تواند عملکرد مدل‌ها را بهبود بخشد. با تمرکز بر ویژگی‌های بارز و حذف نویز و داده‌های غیرمفید، مدل‌ها قادر خواهند بود به صورت کارآمدتر و دقیق‌تری به تحلیل و تشخیص بپردازند. استخراج ویژگی همچنین به افزایش قابلیت تفسیر مدل‌ها کمک می‌کند و باعث می‌شود تا الگوریتم‌های تشخیصی بتوانند الگوهای خاص را بهتر شناسایی و پیش‌بینی کنند. در ادامه به بررسی دو دسته مهم الگوریتم های استخراج ویژگی در یادگیری ماشین و این پروژه می‌پردازیم.

۳_۱ الگوریتم‌های فرکانسی

الگوریتم‌های فرکانسی در تحلیل سیگنال‌های صوتی نقش مهمی ایفا می‌کنند. این الگوریتم‌ها به استخراج ویژگی‌های فرکانسی از داده‌های صوتی می‌پردازند و اطلاعات کلیدی را از سیگنال‌های خام استخراج می‌کنند. تکنیک‌هایی مانند تبدیل فوریه (FFT)، تبدیل موجک و تحلیل طیفی از جمله روش‌های رایج در این زمینه هستند. این الگوریتم‌ها با تجزیه سیگنال به اجزای فرکانسی مختلف، به ما کمک می‌کنند تا الگوهای مهم و ویژگی‌های پنهان در صدا را شناسایی کنیم. استفاده از الگوریتم‌های فرکانسی در پروژه‌های احراز هویت و

تشخیص جنسیت باعث افزایش دقت و کارایی مدل‌های یادگیری ماشین می‌شود، چرا که این ویژگی‌ها اطلاعات مهمی درباره ساختار و خصوصیات سیگنال صوتی ارائه می‌دهند.

۳_۱_۱ نمایش سیگنال در دامنه فرکانس با استفاده از تبدیل لاگاریتمی مِل Log-Mel Spectrogram

در کدی که در ادامه مشاهده می‌کنید به استخراج ویژگی‌های اسپکترگرام مل لاگریتمی (log-mel spectrogram) از فایل‌های صوتی پرداخته‌ایم و آنها را در یک فایل CSV ذخیره می‌کنیم. ابتدا، داده‌های صوتی از فایل‌ها بارگذاری و نرمال‌سازی می‌شوند. سپس اسپکترگرام مل محاسبه شده و به دسی‌بل تبدیل می‌شود. ویژگی‌های اسپکترگرام مل لاگریتمی میانگین‌گیری می‌شوند و اطلاعات آنها همراه با شناسه دانش‌آموز و جنسیت در قالب دیتافریم پانداس ذخیره می‌گردند. این فرآیند به تحلیل و تشخیص بهتر داده‌های صوتی کمک کرده و برای پروژه‌های احراز هویت و تشخیص جنسیت بسیار مفید است.

```
import librosa
import numpy as np
import pandas as pd
import os
from tqdm import tqdm
def extract_log_mel_spectrogram(file_path, sr=22050, n_mels=40, n_fft=1024,
hop_length=512):
    try:
        y, sr = librosa.load(file_path, sr=sr)
        y = librosa.util.normalize(y)
        mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr,
n_mels=n_mels, n_fft=n_fft, hop_length=hop_length)
        log_mel_spectrogram = librosa.power_to_db(mel_spectrogram, ref=np.max)
        return np.mean(log_mel_spectrogram, axis=1)
    except Exception as e:
        print(f"Error at processing: {file_path}: {e}")
        return None
def extract_log_mel_spectrogram_features(audio_folder, output_csv):
    data = []
    audio_files = [f for f in os.listdir(audio_folder) if f.endswith(".wav")]
    for file in tqdm(audio_files):
        file_path = os.path.join(audio_folder, file)
        log_mel_features = extract_log_mel_spectrogram(file_path)
        if log_mel_features is not None:
            parts = file.split('_')
            if len(parts) >= 4:
                student_id = parts[2]  # Student id
```

```
                gender = parts[3].split('.')[0]  # Gender (male/female)
                feature_dict = {"filename": file, "student_id": student_id,
"gender": gender}
                for i in range(len(log_mel_features)):
                    feature_dict[f'log_mel_{i+1}'] = log_mel_features[i]
                data.append(feature_dict)
    df = pd.DataFrame(data)
    df.to_csv(output_csv, index=False)
    print(f"Log Mel Spectrogram features saved at {output_csv}")
```

بارگذاری و نرمال‌سازی داده‌های صوتی:

```
y, sr = librosa.load(file_path, sr=sr)
y = librosa.util.normalize(y)
```

محاسبه اسپکترگرام مل:

```
mel_spectrogram = librosa.feature.melspectrogram(y=y, sr=sr, n_mels=n_mels,
n_fft=n_fft, hop_length=hop_length)
```

تبدیل به دسی‌بل:

```
log_mel_spectrogram = librosa.power_to_db(mel_spectrogram, ref=np.max)
```

میانگین‌گیری از ویژگی‌های اسپکترگرام مل:

```
return np.mean(log_mel_spectrogram, axis=1)
```

این خط میانگین‌گیری از ویژگی‌های اسپکترگرام مل لاگریتمی را انجام می‌دهد.

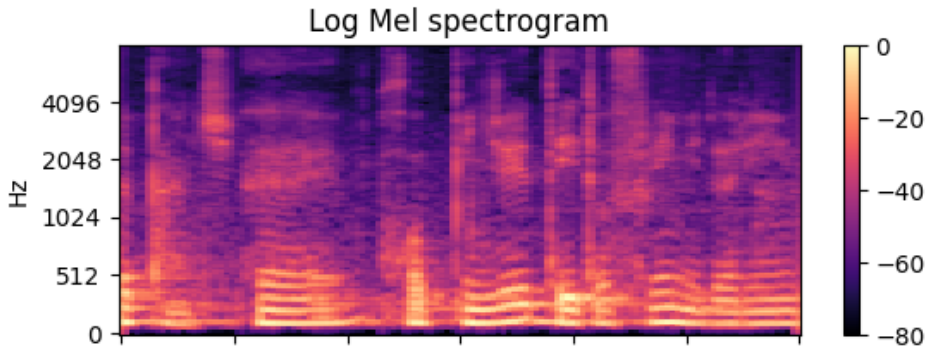
بارگذاری فایل‌های صوتی و استخراج ویژگی‌ها:

```
audio_files = [f for f in os.listdir(audio_folder) if f.endswith(".wav")]
for file in tqdm(audio_files):
    file_path = os.path.join(audio_folder, file)
    log_mel_features = extract_log_mel_spectrogram(file_path)
```

این خطوط فایل‌های صوتی را بارگذاری و ویژگی‌های اسپکترگرام مل لاگریتمی را استخراج می‌کنند.

و در نهایت داده‌های استخراج شده در یک دیتافریم پانداس ذخیره و در فایل CSV خروجی ثبت می‌کنند.

در شکل زیر سیگنال صوتی Log Mel Spectrogram را روی نرخ نمونه برداری ۲۲۰۵۰، فرکانس ۸۰۰۰ هرتز و تعداد باند ۱۲۸ مشاهده میکنیم و همچنین فرکانس :



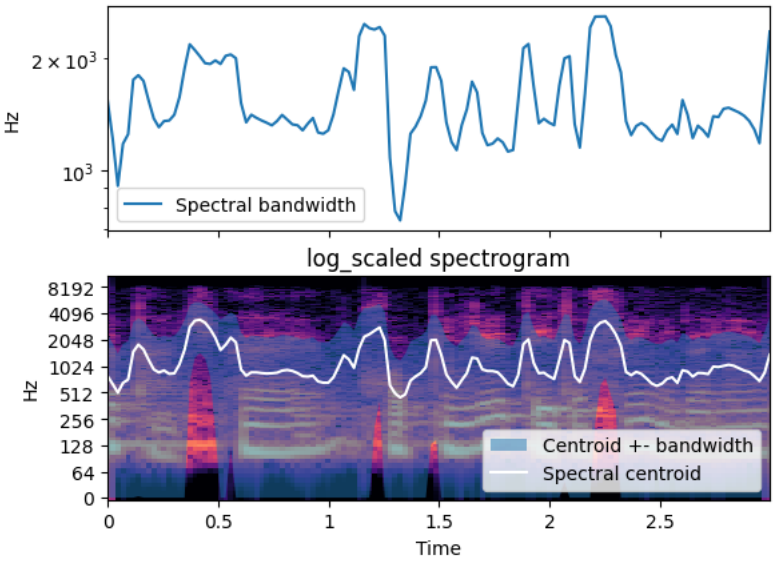
۳_۱_۲ Spectral Bandwidth پهنای باند طیفی سیگنال

تابع ویژگی‌های پهنای باند طیفی را از فایل‌های صوتی استخراج می‌کند و آنها را در یک فایل CSV ذخیره می‌کند. ابتدا داده‌های صوتی بارگذاری و نرمال‌سازی می‌شوند، سپس ویژگی‌های پهنای باند طیفی محاسبه شده و به همراه اطلاعاتی مانند شناسه دانشجو و جنسیت در یک دیتافریم پانداس ذخیره می‌شوند. این فرآیند به تحلیل بهتر داده‌های صوتی و استخراج ویژگی‌های مهم برای پروژه‌های احراز هویت و تشخیص جنسیت کمک می‌کند.

```
import librosa
import numpy as np
import pandas as pd
import os
from tqdm import tqdm

def extract_spectral_bandwidth(file_path, sr=22050):
    try:
        # **بارگذاری و نرمال‌سازی داده‌های صوتی**
        y, sr = librosa.load(file_path, sr=sr)
        y = librosa.util.normalize(y)

        # **محاسبه پهنای باند طیفی**
        spectral_bandwidth = librosa.feature.spectral_bandwidth(y=y, sr=sr)

        # **بازگشت میانگین پهنای باند طیفی**
        return np.mean(spectral_bandwidth, axis=1)
    except Exception as e:
        print(f"Error at processing: {file_path}: {e}")
        return None

def extract_spectral_bandwidth_features(audio_folder, output_csv):
    data = []
    # **بارگذاری فایل‌های صوتی و استخراج ویژگی‌ها**
```

```
    audio_files = [f for f in os.listdir(audio_folder) if f.endswith(".wav")]
    for file in tqdm(audio_files):
        file_path = os.path.join(audio_folder, file)
        spectral_bandwidth_features = extract_spectral_bandwidth(file_path)
        if spectral_bandwidth_features is not None:
            parts = file.split('_')
            if len(parts) >= 4:
                student_id = parts[2]  # شناسه دانشجو
                gender = parts[3].split('.')[0]  # جنسیت (مرد/زن)
                feature_dict = {"filename": file, "student_id": student_id,
"gender": gender}
                for i in range(len(spectral_bandwidth_features)):
                    feature_dict[f'bandwidth_{i+1}'] =
spectral_bandwidth_features[i]
                data.append(feature_dict)

    # **ایجاد دیتافریم و ذخیره در فایل CSV**
    df = pd.DataFrame(data)
    df.to_csv(output_csv, index=False)
    print(f"Spectral bandwidth features saved at {output_csv}")
```

در شکل زیر سیگنال حاصل از الگوریتم `Spectral Bandwidth` را مشاهده میکنیم:



۳_۱_۳ Spectral Centroid مرکز ثقل طیف فرکانسی سیگنال

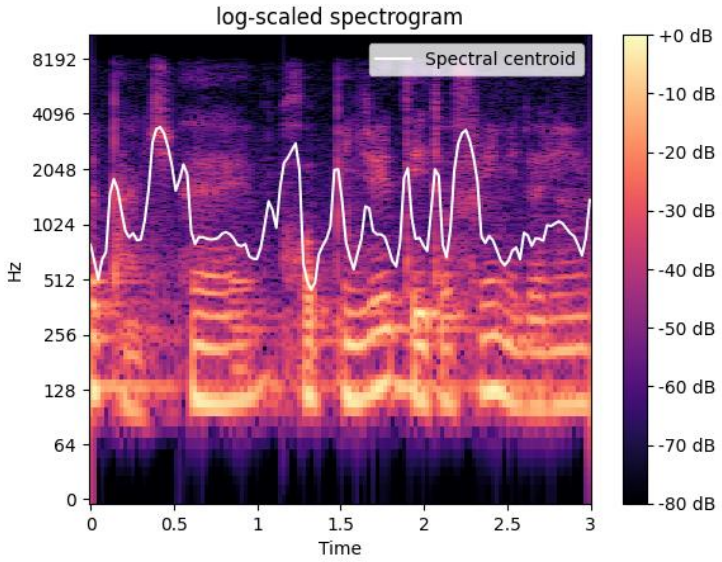
کدی که در ادامه مشاهده می‌فرمایین، ویژگی‌های مرکز ثقل طیفی را از فایل‌های صوتی استخراج کرده و آنها را در یک فایل CSV ذخیره می‌کند. ابتدا داده‌های صوتی بارگذاری و نرمال‌سازی می‌شوند، سپس ویژگی‌های مرکز ثقل طیفی محاسبه و به همراه اطلاعاتی مانند شناسه دانش‌آموز و جنسیت در یک

دیتافریم پانداس ذخیره می‌شوند. این فرآیند به تحلیل بهتر داده‌های صوتی و استخراج ویژگی‌های مهم برای پروژه‌های احراز هویت و تشخیص جنسیت کمک می‌کند.

```
import librosa
import numpy as np
import pandas as pd
import os
from tqdm import tqdm

def extract_spectral_centroid(file_path, sr=22050):
    try:
        # **بارگذاری و نرمال‌سازی داده‌های صوتی**
        y, sr = librosa.load(file_path, sr=sr)
        y = librosa.util.normalize(y)

        # **محاسبه مرکز ثقل طیفی**
        spectral_centroid = librosa.feature.spectral_centroid(y=y, sr=sr)

        # **بازگشت میانگین مرکز ثقل طیفی**
        return np.mean(spectral_centroid, axis=1)
    except Exception as e:
        print(f"Error at processing: {file_path}: {e}")
        return None


def extract_spectral_centroids_features(audio_folder, output_csv):
    data = []
    # **بارگذاری فایل‌های صوتی و استخراج ویژگی‌ها**
    audio_files = [f for f in os.listdir(audio_folder) if f.endswith(".wav")]
    for file in tqdm(audio_files):
        file_path = os.path.join(audio_folder, file)
        spectral_centroid_features = extract_spectral_centroid(file_path)
        if spectral_centroid_features is not None:
            parts = file.split('_')
            if len(parts) >= 4:
                student_id = parts[2]  # شناسه دانشجو
                gender = parts[3].split('.')[0]  # جنسیت (مرد/زن)
                feature_dict = {"filename": file, "student_id": student_id,
"gender": gender}
                for i in range(len(spectral_centroid_features)):
                    feature_dict[f'spectral_centroid_{i+1}'] =
spectral_centroid_features[i]
                data.append(feature_dict)

    # **ایجاد دیتافریم و ذخیره در فایل CSV**
    df = pd.DataFrame(data)
    df.to_csv(output_csv, index=False)
```

```
print(f"Spectral Centroids features saved at {output_csv}")
```

در شکل زیر سیگنال حاصل از اعمال الگوریتم Spectral Centroid را روی ویس مشاهده می‌کنیم:



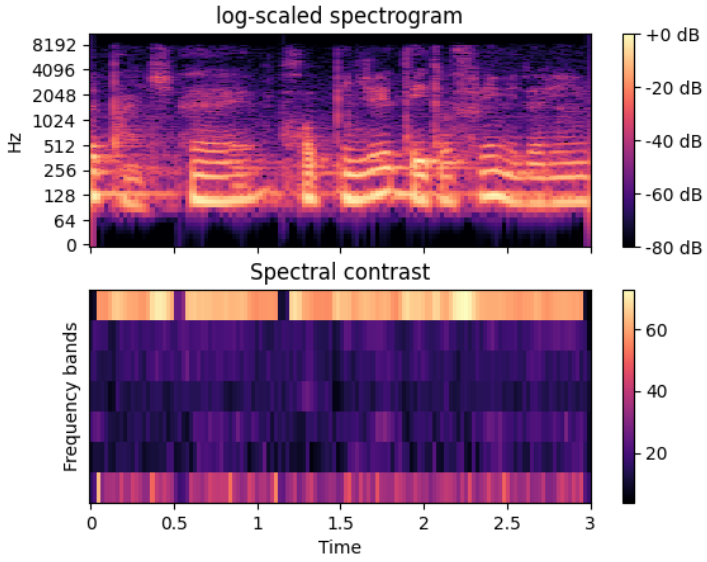
۳_۱_۴ Spectral Contrast کنتراست یا اختلاف طیفی بین بخش‌های مختلف سیگنال

این الگوریتم ویژگی‌های کنتراست طیفی را از فایل‌های صوتی استخراج می‌کند و آنها را در این پروژه در یک فایل CSV ذخیره می‌کند. ابتدا داده‌های صوتی از فایل بارگذاری و نرمال‌سازی می‌شوند. سپس تبدیل فوریه کوتاه‌مدت (STFT) بر روی سیگنال اعمال شده و کنتراست طیفی با استفاده از نوارهای فرکانسی مختلف محاسبه می‌شود. میانگین کنتراست طیفی به عنوان ویژگی استخراج شده و به همراه اطلاعاتی مانند شناسه دانش‌آموز و جنسیت در یک دیتافریم پانداس ذخیره می‌گردد. این فرآیند به تحلیل دقیق‌تر داده‌های صوتی و استخراج ویژگی‌های مهم برای پروژه‌های احراز هویت و تشخیص جنسیت کمک می‌کند.

```
import librosa
import numpy as np
import pandas as pd
import os
from tqdm import tqdm
```

```
def extract_spectral_contrast(file_path, sr=22050, n_bands=6):
    try:
        # **بارگذاری و نرمال‌سازی داده‌های صوتی**
        y, sr = librosa.load(file_path, sr=sr)
        y = librosa.util.normalize(y)

        # **تبدیل فوریه کوتاه‌مدت (STFT)**
        stft = np.abs(librosa.stft(y, n_fft=1024, hop_length=512))

        # **محاسبه کنتراست طیفی**
        spectral_contrast = librosa.feature.spectral_contrast(S=stft, sr=sr,
n_bands=n_bands)

        # **بازگشت میانگین کنتراست طیفی**
        return np.mean(spectral_contrast, axis=1)
    except Exception as e:
        print(f"Error at processing: {file_path}: {e}")
        return None


def extract_spectral_contrast_features(audio_folder, output_csv):
    data = []
    # **بارگذاری فایل‌های صوتی و استخراج ویژگی‌ها**
    audio_files = [f for f in os.listdir(audio_folder) if f.endswith(".wav")]
    for file in tqdm(audio_files):
        file_path = os.path.join(audio_folder, file)
        spectral_contrast_features = extract_spectral_contrast(file_path)
        if spectral_contrast_features is not None:
            parts = file.split('_')
            if len(parts) >= 4:
                student_id = parts[2]  # شناسه دانشجو
                gender = parts[3].split('.')[0]  # جنسیت (مرد/زن)
                feature_dict = {"filename": file, "student_id": student_id,
"gender": gender}
                for i in range(len(spectral_contrast_features)):
                    feature_dict[f'spectral_contrast_{i+1}'] =
spectral_contrast_features[i]
                data.append(feature_dict)

    # **ایجاد دیتافریم و ذخیره در فایل CSV**
    df = pd.DataFrame(data)
    df.to_csv(output_csv, index=False)
    print(f"Spectral contrast features saved at {output_csv}")
```

در شکل زیر نحوه نمایش سیگناهای صوتی حاصل از Spectral Contrast را مشاهده می‌کنیم:



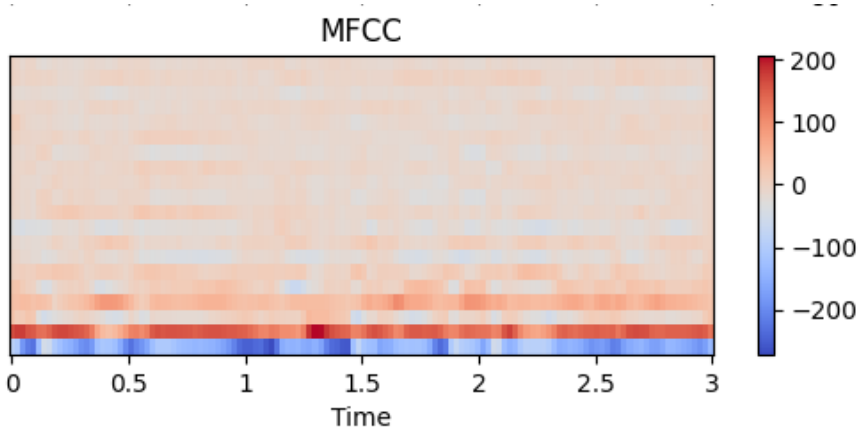
۳_۱_۵ Spectral MFCC (Mel-Frequency Cepstral Coefficients) ضرایب کپسترال فرکانس مِل، که ویژگی‌های فرکانسی سیگنال را استخراج می‌کنند

در این مرحله ویژگی‌های MFCC (ضریب‌های کپسترال فرکانس مل) را از فایل‌های صوتی استخراج و در یک فایل CSV ذخیره می‌کنیم. ابتدا داده‌های صوتی بارگذاری و نرمال‌سازی می‌شوند. سپس با استفاده از تبدیل فوریه کوتاه‌مدت (STFT)، اسپکترگرام توان محاسبه شده و از طریق فیلتر مل به اسپکترگرام مل تبدیل می‌شود. سپس اسپکترگرام مل به دسی‌بل تبدیل شده و ویژگی‌های MFCC استخراج می‌شوند. این ویژگی‌ها همراه با اطلاعاتی مانند شناسه دانش‌آموز و جنسیت در یک دیتافریم پانداس ذخیره می‌شوند. این فرآیند به تحلیل دقیق‌تر داده‌های صوتی و استخراج ویژگی‌های مهم برای پروژه‌های احراز هویت و تشخیص جنسیت کمک می‌کند.

```
import librosa
import numpy as np
import pandas as pd
import os
from tqdm import tqdm

def extract_spectral_contrast(file_path, sr=22050, n_bands=6):
    try:
        # **بارگذاری و نرمال‌سازی داده‌های صوتی**
        y, sr = librosa.load(file_path, sr=sr)
        y = librosa.util.normalize(y)
```

```
        # **تبدیل فوریه کوتاه‌مدت (STFT)**
        stft = np.abs(librosa.stft(y, n_fft=1024, hop_length=512))

        # **محاسبه کنتراست طیفی**
        spectral_contrast = librosa.feature.spectral_contrast(S=stft, sr=sr,
n_bands=n_bands)

        # **بازگشت میانگین کنتراست طیفی**
        return np.mean(spectral_contrast, axis=1)
    except Exception as e:
        print(f"Error at processing: {file_path}: {e}")
        return None


def extract_spectral_contrast_features(audio_folder, output_csv):
    data = []
    # **بارگذاری فایل‌های صوتی و استخراج ویژگی‌ها**
    audio_files = [f for f in os.listdir(audio_folder) if f.endswith(".wav")]
    for file in tqdm(audio_files):
        file_path = os.path.join(audio_folder, file)
        spectral_contrast_features = extract_spectral_contrast(file_path)
        if spectral_contrast_features is not None:
            parts = file.split('_')
            if len(parts) >= 4:
                student_id = parts[2]  # شناسه دانشجو
                gender = parts[3].split('.')[0]  # جنسیت (مرد/زن)
                feature_dict = {"filename": file, "student_id": student_id,
"gender": gender}
                for i in range(len(spectral_contrast_features)):
                    feature_dict[f'spectral_contrast_{i+1}'] =
spectral_contrast_features[i]
                data.append(feature_dict)

    # **ایجاد دیتافریم و ذخیره در فایل CSV**
    df = pd.DataFrame(data)
    df.to_csv(output_csv, index=False)
    print(f"Spectral contrast features saved at {output_csv}")
```

آنچه مشاهده می‌کنیم در تصویر زیر سیگنال‌های حاصل از MFCC است:



۳_۲ الگوریتم‌های زمانی

در این مرحله از دو الگوریتم زمانی برای استخراج ویژگی در پروژه استفاده شده است. الگوریتم‌های زمانی در تحلیل سیگنال‌های صوتی نقش بسیار مهمی ایفا می‌کنند. این الگوریتم‌ها به تحلیل و استخراج ویژگی‌هایی از سیگنال‌های صوتی در حوزه زمان می‌پردازند که می‌تواند اطلاعات مهمی درباره الگوهای زمانی و تغییرات سیگنال ارائه دهد. استفاده از ویژگی‌های زمانی مانند زهره انرژی، انرژی میانگین، و خودهمبستگی در پروژه‌های احراز هویت و تشخیص جنسیت می‌تواند دقت مدل‌ها را بهبود بخشد، زیرا این ویژگی‌ها می‌توانند الگوهای خاصی از صدا را که ممکن است در حوزه فرکانسی نادیده گرفته شوند، برجسته کنند و باعث می‌شود مدل‌های یادگیری ماشین با دقت بیشتری به تحلیل و تشخیص بپردازند.

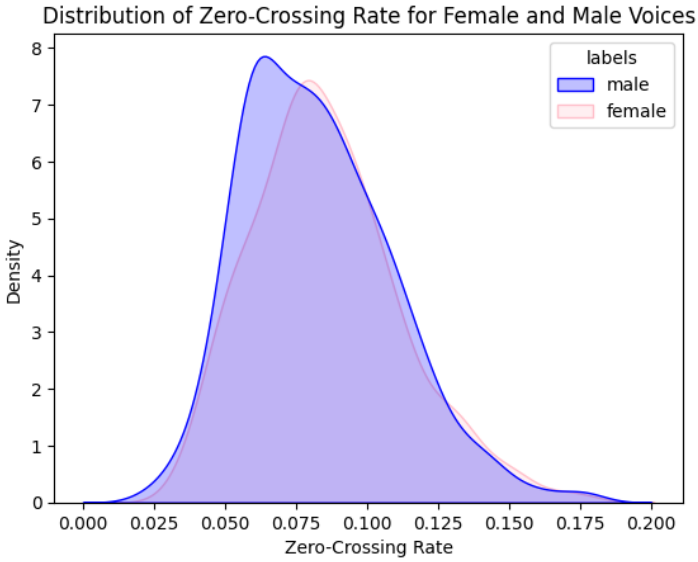
۳_۲_۱ Zero Crossing تعداد دفعاتی که سیگنال از خط صفر عبور می‌کند

در این الگوریتم و این پروژه ویژگی نرخ عبور از صفر (Zero Crossing Rate) را از فایل‌های صوتی استخراج و در یک فایل CSV ذخیره می‌کنیم. ابتدا داده‌های صوتی بارگذاری و نرمال‌سازی می‌شوند. سپس نرخ عبور از صفر محاسبه و میانگین‌گیری می‌شود. ویژگی‌های استخراج شده همراه با اطلاعاتی مانند شناسه دانشجو و جنسیت در یک دیتافریم پانداس ذخیره می‌شوند. این فرآیند به تحلیل دقیق‌تر داده‌های صوتی و استخراج ویژگی‌های مهم برای پروژه‌های احراز هویت و تشخیص جنسیت کمک می‌کند.

```
import librosa
import numpy as np
```

```
import pandas as pd
import os
from tqdm import tqdm

def extract_zero_crossing_rate(file_path, sr=22050):
    try:
        # **بارگذاری و نرمال‌سازی داده‌های صوتی**
        y, sr = librosa.load(file_path, sr=sr)
        y = librosa.util.normalize(y)

        # **محاسبه نرخ عبور از صفر**
        zcr = librosa.feature.zero_crossing_rate(y=y)

        # **بازگشت میانگین نرخ عبور از صفر**
        return np.mean(zcr)
    except Exception as e:
        print(f"Error at processing: {file_path}: {e}")
        return None


def extract_zero_crossing_rate_features(audio_folder, output_csv):
    data = []
    # **بارگذاری فایل‌های صوتی و استخراج ویژگی‌ها**
    audio_files = [f for f in os.listdir(audio_folder) if f.endswith(".wav")]
    for file in tqdm(audio_files):
        file_path = os.path.join(audio_folder, file)
        zcr_feature = extract_zero_crossing_rate(file_path)
        if zcr_feature is not None:
            parts = file.split('_')
            if len(parts) >= 4:
                student_id = parts[2]  # شناسه دانش‌آموز
                gender = parts[3].split('.')[0]  # جنسیت (مرد/زن)
                data.append({"filename": file, "student_id": student_id,
"gender": gender, "zcr": zcr_feature})

    # **ایجاد دیتافریم و ذخیره در فایل CSV**
    df = pd.DataFrame(data)
    df.to_csv(output_csv, index=False)
    print(f"Zero crossing rate features saved at {output_csv}")
```

آنچه در شکل زیر مشاهده می‌کنیم نمایش توزیع دوکلاس زن و مرد حاصل از اعمال ویژگی زمانی Zero Crossing است. همانطور که مشاهده می‌شود هم‌پوشانی زیادی ایجاد شده است که نشان از عملکرد ضعیف این الگوریتم است:



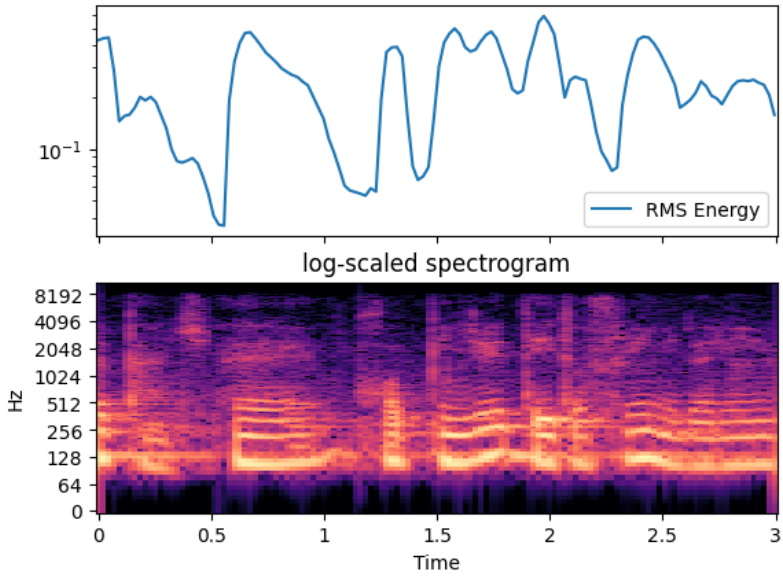
۳_۲_۲ Energy مقدار انرژی موجود در سیگنال در یک بازه زمانی خاص

در اینجا ویژگی‌های انرژی ریشه‌میانگین‌مربع (RMS) را از فایل‌های صوتی استخراج و در یک فایل CSV ذخیره می‌کنیم. ابتدا داده‌های صوتی بارگذاری و نرمال‌سازی می‌شوند. سپس انرژی RMS محاسبه و میانگین‌گیری می‌شود. ویژگی‌های استخراج شده به همراه اطلاعاتی مانند شناسه دانش‌آموز و جنسیت در یک دیتافریم پانداس ذخیره می‌شوند. این فرآیند به تحلیل دقیق‌تر داده‌های صوتی و استخراج ویژگی‌های مهم برای پروژه‌های احراز هویت و تشخیص جنسیت کمک می‌کند. در ادامه کد را می‌بینیم:

```
import librosa
import numpy as np
import pandas as pd
import os
from tqdm import tqdm

def extract_energy(file_path, sr=22050, hop_length=512):
    try:
        # **بارگذاری و نرمال‌سازی داده‌های صوتی**
        y, sr = librosa.load(file_path, sr=sr)  # بارگذاری فایل صوتی
        y = librosa.util.normalize(y)  # نرمال‌سازی

        # **محاسبه انرژی ریشه‌میانگین‌مربع (RMS)**
        rms_energy = librosa.feature.rms(y=y, hop_length=hop_length)  # انرژی ریشه‌میانگین‌مربع

        # **تولید بردار زمانی بر اساس تعداد فریم‌ها**
```

```
        times = librosa.frames_to_time(np.arange(rms_energy.shape[1]), sr=sr,
hop_length=hop_length)

        return rms_energy, times
    except Exception as e:
        print(f"Error While processing {file_path}: {e}")
        return None, None

def extract_energy_features(audio_folder, output_csv):
    data = []
    # **بارگذاری فایل‌های صوتی و استخراج ویژگی‌ها**
    audio_files = [f for f in os.listdir(audio_folder) if f.endswith(".wav")]
    for file in tqdm(audio_files):
        file_path = os.path.join(audio_folder, file)
        rms_energy, _ = extract_energy(file_path)
        if rms_energy is not None:
            parts = file.split('_')
            if len(parts) >= 4:
                student_id = parts[2]  # شماره دانشجو
                gender = parts[3].split('.')[0]  # جنسیت (مرد/زن)

                # **محاسبه و ذخیره ویژگی‌های انرژی**
                energy_features = np.mean(rms_energy, axis=1)
                feature_dict = {"filename": file, "student_id": student_id,
"gender": gender}
                for i in range(len(energy_features)):
                    feature_dict[f'energy_{i + 1}'] = energy_features[i]

                data.append(feature_dict)

    # **ایجاد دیتافریم و ذخیره در فایل CSV**
    df = pd.DataFrame(data)
    df.to_csv(output_csv, index=False)
    print(f"Energy features saved at {output_csv}")
```

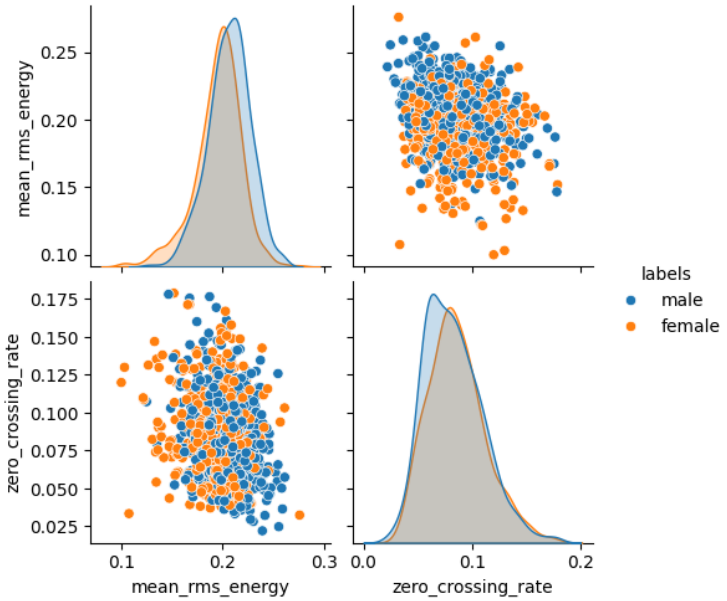
در شکل زیر سیگنال‌های استخراج ویژگی Energy را مشاهده می‌کنیم:



۳-۴ ارزیابی

در این مرحله به بررسی و نتیجه‌گیری از الگوریتم‌های استخراج ویژگی زمانی و فرکانسی پرداخته‌ایم که گزارش کاملی همراه با توزیع الگوریتم‌ها در فایل نوت‌بوک(پوشه appendix) مربوطه موجود است. هم‌چنین در اثر این مشاهدات متوجه شدیم که الگوریتم‌های استخراج ویژگی زمانی به تنهایی حتی روی دادگان ساده دو کلاسه نیز عملکرد مثبت و قابل توجهی ندارند و هم‌پوشانی زیادی ایجاد می‌کنند.

شکل زیر دو نوع توزیع روی سیگنال‌های صوتی زمانی را نشان می‌دهد که گویای نتیجه‌گیری بالا می‌تواند باشد:



٤ طبقه‌بندی

۴_۱ تشخیص جنسیت

در این بخش از مستندات پروژه ،به بررسی کد های نوشته شده و نتایج بدست آماده برای تشخیص جنسیت افراد از روی صوت آن‌ها می‌پردازیم . کد های مربوط به این بخش در مسیر ml_project/Audio_GenderDetection/ قرار گرفته‌اند.

۴_۱_۱ جداسازی داده test و train

با تشخیص جنسیت صوتی یکی از حوزه‌های مهم در پردازش سیگنال‌های صوتی و هوش مصنوعی است که کاربردهای گسترده‌ای در زمینه‌های مختلف دارد. این فناوری به طور عمده در سیستم‌های شناسایی گفتار، دستیارهای صوتی، و خدمات مشتریان استفاده می‌شود تا بتوانند به طور بهینه با کاربران ارتباط برقرار کنند. همچنین در زمینه‌های پزشکی، به ویژه در تشخیص اختلالات گفتاری و صوتی، تشخیص جنسیت می‌تواند به تحلیل و درمان بهتر بیماران کمک کند. در حوزه‌های امنیتی نیز، این تکنولوژی می‌تواند در شناسایی و تأیید هویت افراد بر اساس ویژگی‌های صوتی آن‌ها مورد استفاده قرار گیرد. به طور کلی، تشخیص جنسیت

صوتی نه تنها به بهبود تعاملات انسانی-کامپیوتری کمک می‌کند، بلکه در تحلیل داده‌های اجتماعی و فرهنگی نیز می‌تواند مفید باشد. مسئله تشخیص جنسیت را می‌توان به صورت یک مسئله دو طبقه بندی دو کلاسه مدلسازی کرد . در این مسئله ما صوت های مربوط به کلاس خانم‌ها را ۰ و کلاس مربوط به داده‌های آقایان را کلاس ۱ در نظر گرفته‌ایم . برای جداسازی داده‌های آموزشی از داده‌های تست روش‌های مختلفی وجود دارد . روش اول بدین صورت است که می‌توانیم همه های داده‌ها را ابتدا پیش پردازش کرده ، سپس ویژگی‌های آن‌ها را استخراج کرده و در نهایت ۲۵ درصد نمونه‌های تولید شده را به عنوان داده تست در نظر گرفت و جدا کرد . نکته مهمی که باید به آن توجه داشت این است . در مرحله پیش پردازش ، ما صوت های اولیه را به سگمنت هایی به طول ۳ ثانیه می شکنیم . تمام داده‌های ما جمعا ۳۵ ساعت می‌باشد و این عمل منجر به تولید ۴۲۰۰۰ سگمنت ۳ ثانیه ای می‌شود . می‌توانیم تمامی این ۴۲۰۰۰ فایل صوتی تولید شده را استخراج ویژگی کرده و سپس بطور تصادفی ۲۵ درصد این فایل‌های سه ثانیه ای را برای تست جدا کرد . در این پروژه ، برای جداسازی داده آموزش و داده تست ، ابتدا این روش پیاده‌سازی شد . اما پس از مدلسازی ، دقت همه ی مدلهای پیاده‌سازی شده به بالای ۹۹ درصد رسید . پس از بررسی های بیشتر متوجه موضع مهمی شدیم . در داده‌های آموزش و داده‌های تست ، سگمنت هایی وجود دارد که صاحب آن صوت ، بخشی از صدایش در داده تست و بخشی در داده آموزش می‌باشد . در مرحله آموزش مدل بر روی صدای این فرد آموزش دیده است . بنابراین می‌تواند صدای وی و جنسیت آن را به خوبی تشخیص دهد . تقریباً تمامی سگمنت هایی که برای تست مدل استفاده شدند ،صدای افرادی بودند که این افراد بخشی از صدایشان نیز در داده آموزشی وجود دارد . بنابراین مدل براحتی صدای آن‌ها را شناخته و جنسیت آن‌ها را تشخیص می‌دهد . این شیوه جداسازی داده ترین و تست در مسئله تشخیص جنیست ، به نوعی قرار دادن لیبل داده‌های تست در داد های آموزشی است . پس بنابراین چنین روشی برای جداسازی داده‌های ترین از داده‌های تست مناسب نیست . نکته مهم دیگری که باید به آن توجه داشت این است . همانطور که قبلتر اشاره کردیم ، حجم داده‌های صوتی مربوط به آقایان تقریباً ۳ برابر داده‌های خانم‌ها می‌باشد . ما ابتدا بدون در نظر گرفتن این نکته مدلسازی را انجام دادیم که این کار منجر به مدلی می‌شود که همواره خروجی ۱ را تولید میکرد . چراکه داده آموزش آن

به سمت یکی از کلاس‌های دچار bias بود . برای انجام یک ارزیابی قابل اطمینان و مناسب از روشی که در ادامه آن را توضیح می‌دهیم ، استفاده کرده‌ایم . کد مربوط به ساخت و جداسازی داده‌های تست و ترین برای مسئله تشخیص جنسیت در Audoi_GenderDetection/prepare_train_test.py پیاده‌سازی شده است .

```
# --------------- Read Raw Data File Names ------------------------- #
raw_data_path = os.path.join("..", "Data", "raw")
preprocessing.convert_filenames_to_lowercase(raw_data_path)
bad_files, filenames = au.raw_audio_files(raw_data_path)
```

در ابتدا ما مسیر داده‌های خام را مشخص کرده و اسامی تمامی فایلها را به حروف کوچک تبدیل میکنیم و سپس لیستی از اسامی تمامی فایل‌های صوتی را در متغیر filenames ذخیره می‌کنیم .

```
# ----------------- Extract general information from good data ----------- #
raw_data_info = au.get_audio_info_from_files(filenames)
n_distinct_males = len(raw_data_info["n_speakers"]["males"])
n_distinct_females = len(raw_data_info["n_speakers"]["females"])
n_distinct_unknowns = len(raw_data_info["n_speakers"]["unknown"])
```

در ادامه اطلاعات مربوط به فایلهای صوتی ، شامل همان داده‌هایی که در ابتدای گزارش به آن‌ها اشاره کرده‌ایم را نمایش می‌دهیم . خروجی اطلاعات زیر است :

- بطور کلی مشخصات داده‌های خام اولیه در اختیار داده شده بدین صورت می‌باشد :
  - داده‌های خام اولیه پروژه می بایستی در مسیر Data/raw قرار گیرد .
  - تعداد فایل‌های صوتی : ۴۷۳
  - تعداد فایلهای قابل استفاده : ۴۹۹ (شماره دانشجویی و یا جنسیت برخی صوتها قابل شناسایی از روی نام فایل نمی باشد)
  - تعداد فایل‌های صوتی مربوط به آقایان : ۳۲۷
  - تعداد فایل‌های صوتی مربوط به خانم‌ها : ۱۱۲
  - مجموع زمان همه ی صوت های قابل استفاده : ۳۵.۴ ساعت

- مجموع زمان همه ی صوت های آقایان : ۲۵.۶۵ ساعت
- مجموع زمان همه ی صوت های خانم‌ها : ۹.۶۸ ساعت
- تعداد گویندگان متمایز : ۱۱۴ نفر
- تعداد گویندگان آقا : ۸۶ نفر
- تعداد گویندگان خانم : ۲۸ نفر

منطق جداسازی داده آموزش از داده ترین در این مسئله بدین صورت می‌باشد

۱. تعداد گویندگان متمایز ۱۱۴ نفر است که ۲۵ درصد این تعداد برابر است با ۲۸ نفر .
۲. قبل از انجام هر پیش پردازشی و هر استخراج ویژگی ، داده‌های مربوط به ۲۸ نفر از افراد کلاس که شامل ۱۴ خانم و ۱۴ آقا می‌باشد را برای تست جدا میکنیم .
۳. سپس به پیش پردازش و استخراج ویژگی می‌پردازیم .

این منطق در برنامه به وسیله خطوط زیر پیاده‌سازی شده است :

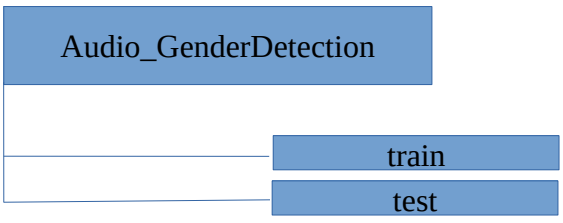
```
# ------------ Calculate required number of files for train and test set -----
------ #
num_test_speakers = int((n_distinct_males + n_distinct_females) * test_ratio)
n_females_speakers_test = int(num_test_speakers / (m2f_ratio_test + 1))
n_males_speaker_test = num_test_speakers - n_females_speakers_test
num_train_speakers = (n_distinct_males + n_distinct_females) -
num_test_speakers
n_females_speakers_train = n_distinct_females - n_females_speakers_test
n_males_speaker_train = n_distinct_males - n_males_speaker_test

# ------------------- Split dataset into train and test ---------------- #
males, females, unknowns, bad = au.extract_speakers_id(filenames)
train_males_speaker, train_females_speakers = set(males.keys()),
set(females.keys())

test_males_speaker = set(random.choices(list(train_males_speaker),
k=n_males_speaker_test))
test_females_speaker = set(random.choices(list(train_females_speakers),
k=n_females_speakers_test))
train_males_speaker.difference_update(test_males_speaker)
train_females_speakers.difference_update(test_females_speaker)

train_males_speaker = set(random.choices(list(train_males_speaker),
k=n_males_speaker_train))
train_females_speakers = set(random.choices(list(train_females_speakers),
k=n_females_speakers_train))

train_speakers = train_males_speaker.union(train_females_speakers)
test_speakers = test_males_speaker.union(test_females_speaker)
```

```
train_path_raw = os.path.join('.', 'train', 'raw')
test_path_raw = os.path.join('.', 'test', 'raw')
os.makedirs(train_path_raw)
os.makedirs(test_path_raw)
```

در نهایت ۱۴ آقا و ۱۴ خانم انتخاب شده برای تست را در پوشه ای متفاوت ذخیره می‌کنیم:

```
for m_speaker in train_males_speaker:
    for file_ in males[m_speaker]:
 shutil.copyfile(file_, os.path.join(train_path_raw, os.path.basename(file_)))
for f_speaker in train_females_speakers:
    for file_ in females[f_speaker]:
        shutil.copyfile(file_, os.path.join(train_path_raw,
os.path.basename(file_)))

n_test_per_person = 2
for m_speaker in test_males_speaker:
    for i, file_ in enumerate(males[m_speaker]):
        shutil.copyfile(file_, os.path.join(test_path_raw,
os.path.basename(file_)))
        if i >= n_test_per_person -1:
            break
for f_speaker in test_females_speaker:
    for i, file_ in enumerate(females[f_speaker]):
        shutil.copyfile(file_, os.path.join(test_path_raw,
os.path.basename(file_)))
        if i >= n_test_per_person -1:
            break
```

تا کنون تنها صوت افرادی که برای تست انتخاب شده‌اند را از صوت افرادی که برای آموزش مانده اند را جدا کردیم . ساختار داده‌های مسئله تشخیص جنسیت ، بدین صورت می‌باشد



حال می بایستی که صوت های انتخاب شده را پیش پردازش کنیم .

```
train_path_processed = os.path.join('.', 'train', 'processed')
train_path_raw = os.path.join('.', 'train', 'raw')
output_csv = os.path.join(train_path_processed, 'train.csv')
preprocessing.process_directory(train_path_raw, train_path_processed,
output_csv)
```

اینکار باعث می‌شود که تنها صوت های مربوط به داده ترین انتخاب شده ، سگمنت شوند . در نهایت برای آماده شدن دیتاست آموزشی ، ویژگی‌های هر سگمنت را با استفاده از توابعی که قبلتر توضیح دادیم (در بخش مقدمه) ، استخراج ویژگی را انجام می‌دهیم .

```
# --------------------- Feature extraction --------------- #
train_path_processed = os.path.join('.', 'train', 'processed')
features_files =
featrue_extraction.extract_features_from_audios(train_path_processed)
features = pd.read_csv(features_files)
features['label'] = features['gender'].apply(lambda x: 1 if x == 'male' else
0)
features = features.drop(columns=['gender', 'filename', 'student_id'])
features.to_csv(os.path.join(".", "train", "train_features.csv"), index=False)
```

۴_۱_۲ مدل‌سازی و ارزیابی اولیه

در ابتدای مدلسازی اولیه ، سعی شده است که بدون هیچ انتخاب ویژگی و کاهش ویژگی ، مدلسازی بر روی دیتاست های آماده شده انجام شود. این مدلسازی اولیه در فایل Audio_GenderDetection/train_model_base.ipynb/ انجام شده است . در ادامه به بررسی کد های اولیه برای آموزش می‌پردازیم .

برای مدلسازی از چندین مدل استفاده شده است . برای سادگی کار و مقایسه بهتر تمامی مدل های موردنظر و متد های مورد نیاز در قالب یک Abastraction Class در برنامه پیاده‌سازی شده‌اند . این مدل

ها در فایل Audio_GenderDetection/models.py پیاده‌سازی شده‌اند . ابتدا به بررسی کد های درون این فایل می‌پردازیم :

```
class GenderDetectionModel(ABC):
    @abstractmethod
    def __init__(self):
        self.model = None
        self.name = None

    def train(self, x, y):
        self.model.fit(x, y)

    def predict(self, x):
        return self.model.predict(x)

    def evaluate(self, x, y, metric: str = "accuracy"):
        y_pred = self.predict(x)
        if metric == "accuracy":
            return accuracy_score(y, y_pred)
        elif metric == "f1":
            return f1_score(y, y_pred)
        elif metric == "precision":
            return precision_score(y, y_pred)
        elif metric == "recall":
            return recall_score(y, y_pred)
        else:
raise ValueError("Metric must be either 'accuracy' or 'f1'
or               'precision' or 'recall'")

    def cross_validate(self, x, y, folds=5):
        scores = cross_val_score(self.model, x, y, cv=folds)
        return scores

    def save_model(self, directory: str = "models"):
joblib.dump(self.model, os.path.join(directory,
f"{self.name}               _model.joblib"))
        print(f"Model saved as {os.path.join(directory,
f'{self.name}               _model.joblib')}")
```

در این کد ، یک کلاس انتزاعی به نام GenderDetectionModel را تعریف می‌شود که به عنوان پایه‌ای برای مدل‌های تشخیص جنسیت صوتی عمل می‌کند. این کلاس شامل چندین متد است که اجازه می‌دهد تا مدل‌های خاص خود را با ویژگی‌های مختلف پیاده‌سازی کنیم. در این کلاس، متد __init__ برای تعریف ویژگی‌های اولیه مانند مدل و نام آن استفاده می‌شود، در حالی که متد train برای آموزش مدل با استفاده از داده‌های ورودی x و برچسب‌های y طراحی شده است. متد predict برای پیش‌بینی جنسیت بر اساس داده‌های ورودی استفاده می‌شود و متد evaluate به کاربر این امکان را می‌دهد که عملکرد مدل را با استفاده از معیارهای مختلف مانند دقت، F1، دقت و یادآوری ارزیابی کند. همچنین، متد cross_validate به انجام اعتبارسنجی متقابل مدل کمک می‌کند و در نهایت، متد save_model برای ذخیره‌سازی مدل آموزش‌دیده در یک دایرکتوری مشخص به کار می‌رود. این ساختار این امکان را می‌دهد که به راحتی مدل‌های مختلف را پیاده‌سازی و ارزیابی کنند.

```
class SupportVectorMachine(GenderDetectionModel):
    def __init__(self, probability: bool = True):
        self.probability = probability
        self.model = SVC(probability=probability)
        self.name = "support_vector_machine"


class KNearestNeighbors(GenderDetectionModel):
    def __init__(self):
        self.model = KNeighborsClassifier()
        self.name = "knn"


class AdaBoost(GenderDetectionModel):
    def __init__(self):
        self.model = AdaBoostClassifier()
        self.name = "adaboost"


class MultiLayerPerceptron(GenderDetectionModel):
    def __init__(self, max_iter: int = 1000):
        self.model = MLPClassifier(max_iter=max_iter)
        self.name = "multilayer_perceptron"
```

```
class LogisticRegression(GenderDetectionModel):
    def __init__(self, max_iter: int = 1000):
        self.model = LogReg(max_iter=max_iter)
        self.name = "logistic_regression"
```

این کد شامل چندین کلاس است که هر یک از آن‌ها به پیاده‌سازی مدل‌های مختلف یادگیری ماشین برای تشخیص جنسیت صوتی می‌پردازند و همگی از کلاس انتزاعی GenderDetectionModel ارث‌بری می‌کنند.

- SupportVectorMachine: این کلاس یک مدل ماشین بردار پشتیبان (SVM) را پیاده‌سازی می‌کند و قابلیت محاسبه احتمال پیش‌بینی‌ها را نیز دارد. با استفاده از پارامتر probability می‌توان تعیین کرد که آیا مدل باید احتمال پیش‌بینی‌ها را محاسبه کند یا خیر.
- KNearestNeighbors: این کلاس به پیاده‌سازی الگوریتم نزدیک‌ترین همسایگی (KNN) می‌پردازد و به سادگی با استفاده از KNeighborsClassifier مدل را ایجاد می‌کند.
- AdaBoost: این کلاس مدل آدا بوست را پیاده‌سازی می‌کند و از AdaBoostClassifier برای ایجاد یک مدل قوی بر پایه ترکیب چندین مدل ضعیف استفاده می‌کند.
- MultiLayerPerceptron: این کلاس یک شبکه عصبی چندلایه (MLP) را پیاده‌سازی می‌کند که با استفاده از MLPClassifier و پارامتر max_iter برای تعیین حداکثر تعداد تکرارهای آموزش، به کار می‌رود.
- LogisticRegression: این کلاس مدل رگرسیون لجستیک را پیاده‌سازی می‌کند و از LogReg (که باید به LogisticRegression اشاره داشته باشد) برای انجام پیش‌بینی‌ها استفاده می‌کند و همچنین پارامتر max_iter را برای کنترل تعداد تکرارهای آموزش در نظر می‌گیرد.

پس از تعریف این کلاسها ،از آن‌ها استفاده میکنیم . برای مدلسازی اولیه این مراحل را پیش می‌رویم

۱. فراخوانی کلاسهای های تعریف شده : در بخش ۱ از فایل نوتبوک (train_base_models.ipynb) تمامی کلاس‌های تعریف شده که در بخش قبلی به توضیح آن‌ها پرداختیم را وارد برنامه میکنیم .

```
import os
import joblib
import shutil
import numpy as np
import pandas as pd

from sklearn.preprocessing import StandardScaler

from models import AdaBoost
from models import KNearestNeighbors
from models import LogisticRegression
from models import MultiLayerPerceptron
from models import SupportVectorMachine

import sys
sys.path.insert(1, os.path.join("..", "Audio_Scripts"))
sys.path.insert(1, os.path.join(".."))
from Audio_Scripts import audio_utils as au
from Audio_Scripts import preprocessing
from Audio_Scripts import featrue_extraction
```

۲. خواندن داده‌های آماده شده :داده هایی که در بخش ۴.۱ پروژه روند آماده سازی آن‌ها را توضیح دادیم را از روی دیسک لود می‌شوند

```
3.
4.   train_data = os.path.join("train", "train_features.csv")
5.
6.   train_df = pd.read_csv(train_data)
7.
8.   print("Num train samples:", len(train_df))
9.
```

۱۰. نرمالسازی داده‌ها : پس از لود کردن داده‌ها ، آن‌ها را با استفاده از کلاس StandardScaler به مقیاس یکسان تبدیل می‌کنیم .همچنین ما می بایستی که شی scaler ای که میسازیم را ذخیره کنیم تا بتوانیم در زمان تست از آن استفاده کنیم

```
11.    x_train, y_train = np.array(train_df.drop("label", axis=1)),
       np.array(train_df['label'])
12.
13.    col_mean = np.nanmean(x_train, axis=0)
14.    inds = np.where(np.isnan(x_train))
15.    x_train[inds] = np.take(col_mean, inds[1])
16.
17.    # Scale features
18.    std_scaler = StandardScaler()
19.    x_train_scaled = std_scaler.fit_transform(x_train)
20.
```

۲۱. آموزش ۵ مدل و ارزیابی cross validation

```
models = {
"svm": SupportVectorMachine(),
"mlp": MultiLayerPerceptron(),
"log": LogisticRegression(),
"knn": KNearestNeighbors(),
"ada": AdaBoost(),
}

for model_name, model in models.items():
print(f"Train and evaluate {model_name}.")
scores = model.cross_validate(x_train_scaled, y_train)
print(f"\tCross val score: {scores}")
print("-"*20)
```

همانطور که قبلتر گفته شد ، ارزیابی بدین شکل به نحوی می‌باشد که مدل ما صدای افراد در داده تست را در زمان آموزش یادگرفته است . بنابراین دقت همه ی مدلها در ارزیابی بدین صورت همواره بالای 98 درصد بوده است .

۲۲. ارزیابی مدل روی داده‌های تست :

حالا برخی از این مدلها را بر روی داده‌های تست ارزیابی می‌کنیم . داده‌های تست ، داده‌هایی هستند که صدای آن‌ها برای افرادی است که هیچ بخشی از صدایشام در زمان آموزش جز داده‌ها نبوده است . بنابراین ارزیابی مدل بر روی صدای این افراد ، قابل اطمینان است . با استفاده از کد زیر و کد مشابه با آن ، ما ابتدا یک svm و سپس یک رگرسیون لاجستیک را مجدداً آموزش داده‌ایم و بوسیله داده‌های تست آن‌ها را ارزیابی کرده‌ایم .

```
 from inference import detect_gender
test_data_path = os.path.join(".", "test", "raw")

test_data = [os.path.join(test_data_path, t) for t in
os.listdir(test_data_path)]
test_data = [(t, 0 if "female" in t else 1) for t in test_data]
test_data


svm_clf = SupportVectorMachine()
svm_clf.train(x_train_scaled, y_train)

joblib.dump(svm_clf, os.path.join("saved_models", "svm.joblib"))
```

برای اجرای مدل بر روی یک صدا با هدف تشخیص جنیست منطق بدین صورت می‌باشد : ۱۵ ثانیه از صوت ورودی را استخراج کرده (۵ قسمت ۳ ثانیه ای) و مدل را بر روی این پنج سگمنت اجرا می‌کنیم . برای هر سگمنت مدل یک خروجی یک یا صفر تولید می‌کند . کلاسی که بیشترین رأی را بیاورد ، به عنوان خروجی تصمیم نهایی انتخاب می‌شود . طبق این منطق ما مدل را بروی ۲۸ نمونه تستی اجرا کردیم که شاخص دقت به اندازه ۵۹ درصد بود . به توجه به دقت و در عین حال سرعت پایین مدل، در مرحله بعدی فرآیند انتخاب ویژگی و کاهش ویژگی پیاده‌سازی شده است .

۴_۱_۳ انتخاب ویژگی و کاهش ابعاد(Feature Selection & Feature Reduction)

در مرحله قبلی سعی بر این بود که تعدادی مدل اولیه را بر روی تمامی ویژگی‌های استخراج شده (۶۷ ویژگی) آموزش دهیم و عمل‌کرد مدلها را بسنجیم . اما علاوه بر دقت پایین مدل ها ، سرعت آموزش و سرعت inference بسیار پایین بود . به همین دلیل در این مرحله به سراغ انتخاب زیر مجموعه‌ای sub-optimal از تمام ویژگی‌ها و سپس ترکیب آن‌ها با هم می‌رویم . مرحله feature selection و feature reduction در فایل نوتبوک select_features.ipynb پیاده‌سازی شده است که در ادامه به بررسی بخش‌های مهم آن و تحلیل‌های لازم آن می‌پردازیم .

```
# IMPORT REQUIRED LIBRARIES

import os
import librosa
import librosa.display
import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from IPython.display import Audio
from scipy.signal import butter, filtfilt
from pyloudnorm import Meter, normalize
```

در ابتدا کتابخانه‌های مورد نیاز را وارد برنامه می‌کنیم . برای درک بهتر از ماهیت صوتی داده ، ابتدا کار را با یک صوت شروع می‌کنیم و صحت توابع نوشته شده را با آن بررسی می‌کنیم .

```
data_path = "../Data/raw/"
all_audio = [os.path.join(os.path.abspath(data_path), d) for d in
os.listdir(data_path)]

y, sr = librosa.load(all_audio[0])
display(Audio(y, rate=sr))
```

در این کد اولین فایل صوتی خوانده شده از دیسک را انتخاب میکنیم .

```
def denoise_speech_bandpass(audio_data, sr, lowcut, highcut, order=5):
  nyq = 0.5 * sr
  low = lowcut / nyq
  high = highcut / nyq
  b, a = butter(order, [low, high], btype='band')
```

```
  denoised_audio = filtfilt(b, a, audio_data)

  return denoised_audio

y_denoised = denoise_speech_bandpass(y, sr, lowcut=100, highcut=8000, order=6)
display(Audio(y_denoised, rate=sr))
```

تابع denoise_speech_bandpass به منظور کاهش نویز در سیگنال‌های صوتی طراحی شده است. این تابع با دریافت داده‌های صوتی (audio_data)، نرخ نمونه‌برداری (sr)، و محدوده‌های فرکانسی پایین (lowcut) و بالا (highcut) به عنوان ورودی، یک فیلتر باندپاس طراحی می‌کند. ابتدا، فرکانس نیکو (Nyquist frequency) محاسبه می‌شود و سپس مقادیر فرکانس‌های پایین و بالا به نسبت نیکو نرمال می‌شوند. با استفاده از تابع butter، ضرایب فیلتر باندپاس با مرتبه مشخص (order) تولید می‌شود. سپس، تابع filtfilt برای اعمال فیلتر به داده‌های صوتی استفاده می‌شود که باعث کاهش اثرات فاز و حفظ شکل سیگنال می‌گردد. در نهایت، سیگنال صوتی بدون نویز (denoised_audio) برگردانده می‌شود. در مثال ارائه شده، این تابع برای داده‌های صوتی y با نرخ نمونه‌برداری sr و تنظیمات فرکانسی مشخص فراخوانی می‌شود و نتیجه آن برای پخش به کاربر نمایش داده می‌شود.

```
def remove_silence(audio_data, sr, threshold=0.05, frame_length=4096,
hop_length=512):
    rms = librosa.feature.rms(y=audio_data, frame_length=frame_length,
hop_length=hop_length)[0]
    frames_above_threshold = np.where(rms > threshold)[0]
    non_silent_segments = []
    for i in range(len(frames_above_threshold)):
        start_sample = frames_above_threshold[i] * hop_length
        end_sample = min((frames_above_threshold[i] + 1) * hop_length,
len(audio_data))  # Prevent          exceeding audio length
        non_silent_segments.append(audio_data[start_sample:end_sample])
    if len(non_silent_segments) > 0:
      non_silent_audio = np.concatenate(non_silent_segments)
    else:
      non_silent_audio = audio_data
    return non_silent_audio

non_silent_audio = remove_silence(y_denoised, sr, threshold=0.05,
frame_length=4096, hop_length=512)
display(Audio(non_silent_audio, rate=sr))
```

تابع remove_silence به منظور حذف بخش‌های ساکت از یک سیگنال صوتی طراحی شده است. این تابع با دریافت داده‌های صوتی (audio_data)، نرخ نمونه‌برداری (sr)، آستانه ساکت بودن (threshold)، طول فریم (frame_length)، و طول پرش (hop_length) به عنوان ورودی، ابتدا مقدار ریشه میانگین مربعات (RMS) سیگنال را با استفاده از تابع librosa.feature.rms محاسبه می‌کند. سپس، فریم‌هایی که مقدار RMS آن‌ها بالاتر از آستانه مشخص شده است شناسایی می‌شوند. برای هر یک از این فریم‌ها، نقاط شروع و پایان نمونه‌ها محاسبه می‌شود و بخش‌های غیر ساکت به لیستی اضافه می‌شوند. در نهایت، اگر بخش‌های غیر ساکت وجود داشته باشد، این بخش‌ها با استفاده از np.concatenate به یک سیگنال صوتی یکپارچه تبدیل می‌شوند؛ در غیر این صورت، سیگنال اصلی برگردانده می‌شود. این تابع به کاربر این امکان را می‌دهد که تنها بخش‌های صوتی فعال را از سیگنال حذف کند و نتیجه آن برای پخش به کاربر نمایش داده می‌شود.

```
def normalize_audio(audio_data, sr,  target_lufs=-14):
  meter = Meter(sr)
  loudness = meter.integrated_loudness(audio_data)
  loudness_normalized_audio = normalize.loudness(audio_data, loudness,
target_lufs)
  return loudness_normalized_audio

y_normalized = normalize_audio(non_silent_audio, sr, -14)
display(Audio(y_normalized, rate=sr))
```

تابع normalize_audio به منظور نرمال‌سازی بلندی صوتی یک سیگنال طراحی شده است. این تابع با دریافت داده‌های صوتی (audio_data)، نرخ نمونه‌برداری (sr)، و هدف بلندی صوتی (LUFS) (target_lufs) به عنوان ورودی، ابتدا از کلاس Meter برای محاسبه بلندی صوتی کلی سیگنال استفاده می‌کند. این بلندی به صورت یک مقدار عددی (integrated loudness) محاسبه می‌شود. سپس، با استفاده از تابع normalize.loudness، سیگنال صوتی به گونه‌ای نرمال‌سازی می‌شود که بلندی آن به مقدار هدف مشخص شده (در اینجا ۱۴- LUFS) برسد. در نهایت، سیگنال صوتی نرمال‌شده

(loudness_normalized_audio) برگردانده می‌شود. این تابع به کاربر این امکان را می‌دهد که بلندی صوتی یک سیگنال را به سطح مطلوبی برساند و نتیجه آن برای پخش به کاربر نمایش داده می‌شود.

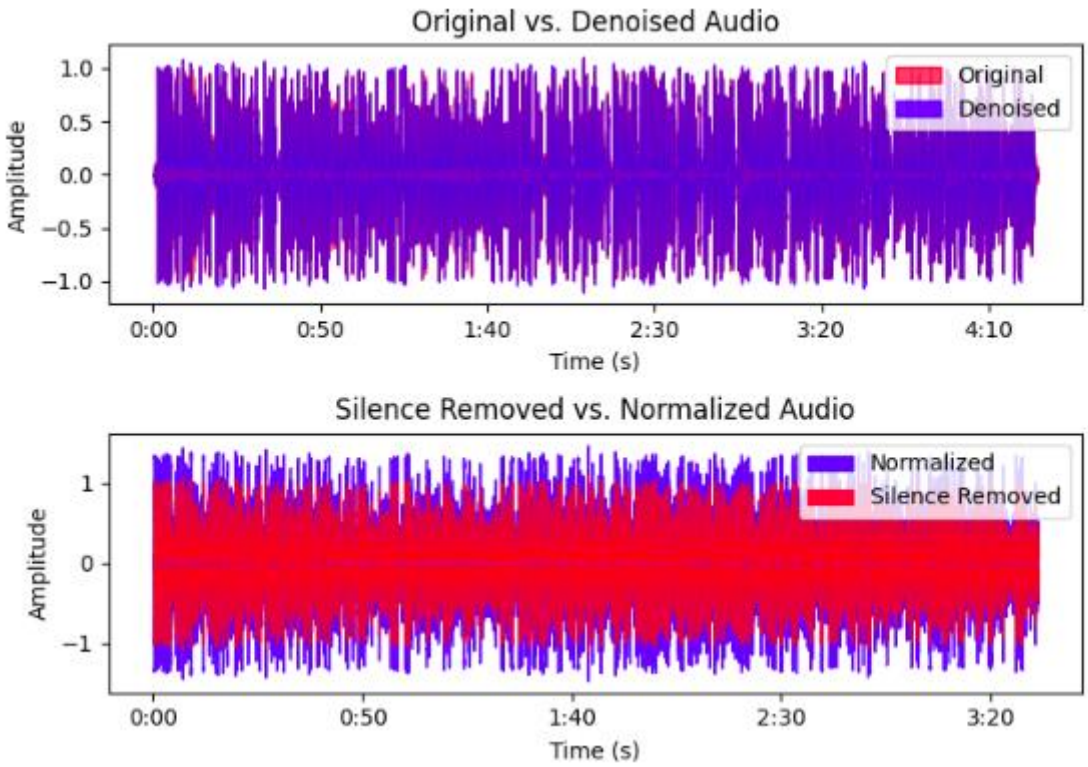
```
def preprocess_audio(audio_data, sr, duration = 5):
  y_denoised = denoise_speech_bandpass(audio_data, sr, lowcut=100,
highcut=8000, order=6)
  y_normalized = normalize_audio(y_denoised, sr, -14)
  non_silent_audio = remove_silence(y_normalized, sr, threshold=0.05,
frame_length=4096, hop_length=512)
  duration = sr*duration
  if len(non_silent_audio) > duration:
    non_silent_audio = non_silent_audio[0:duration]
  return non_silent_audio
```

تابع preprocess_audio به منظور پیش‌پردازش یک سیگنال صوتی طراحی شده است. این تابع با دریافت داده‌های صوتی (audio_data)، نرخ نمونه‌برداری (sr)، و مدت زمان مورد نظر (duration) به عنوان ورودی، مراحل مختلفی را برای بهبود کیفیت سیگنال انجام می‌دهد.

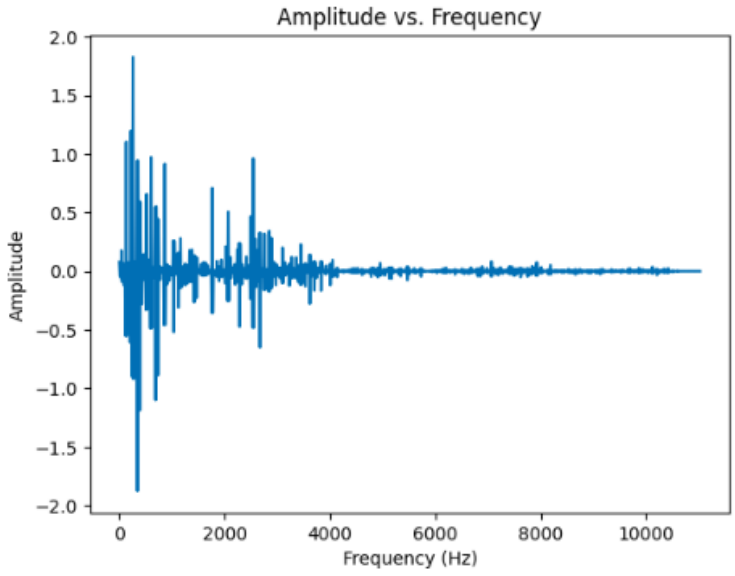
ابتدا، سیگنال صوتی با استفاده از تابع denoise_speech_bandpass از نویزهای غیرمطلوب حذف می‌شود، که در این مرحله تنها فرکانس‌های بین ۱۰۰ تا ۸۰۰۰ هرتز حفظ می‌شوند. سپس، سیگنال حاصل نرمال‌سازی می‌شود تا بلندی آن به سطح هدف -۱۴ LUFS برسد، با استفاده از تابع normalize_audio. در مرحله بعد، بخش‌های ساکت سیگنال با استفاده از تابع remove_silence حذف می‌شوند تا تنها بخش‌های فعال باقی بمانند.

در نهایت، اگر طول سیگنال غیر ساکت بیشتر از مدت زمان مشخص شده باشد، تنها بخش اول سیگنال به طول مشخص شده برگردانده می‌شود. در غیر این صورت، سیگنال بدون تغییر باقی می‌ماند. این تابع به کاربر این امکان را می‌دهد که یک سیگنال صوتی را به صورت مؤثر پیش‌پردازش کند و نتیجه آن به عنوان خروجی برگردانده می‌شود.

پس از تعریف این توابع و اطمینان از عمل‌کرد آن‌ها ، این توابع را روی یک صوت اجرا کرده و سیگنال نهایی را با سیگنال ابتدایی مقایسه میکنیم .



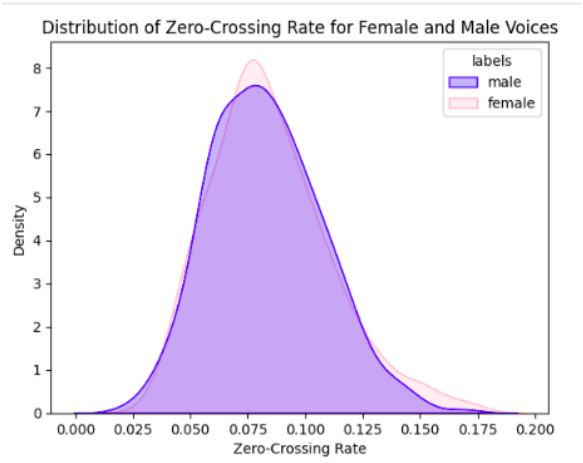
همچنین تبدیل SFTF آن بدین صوت می‌باشد :



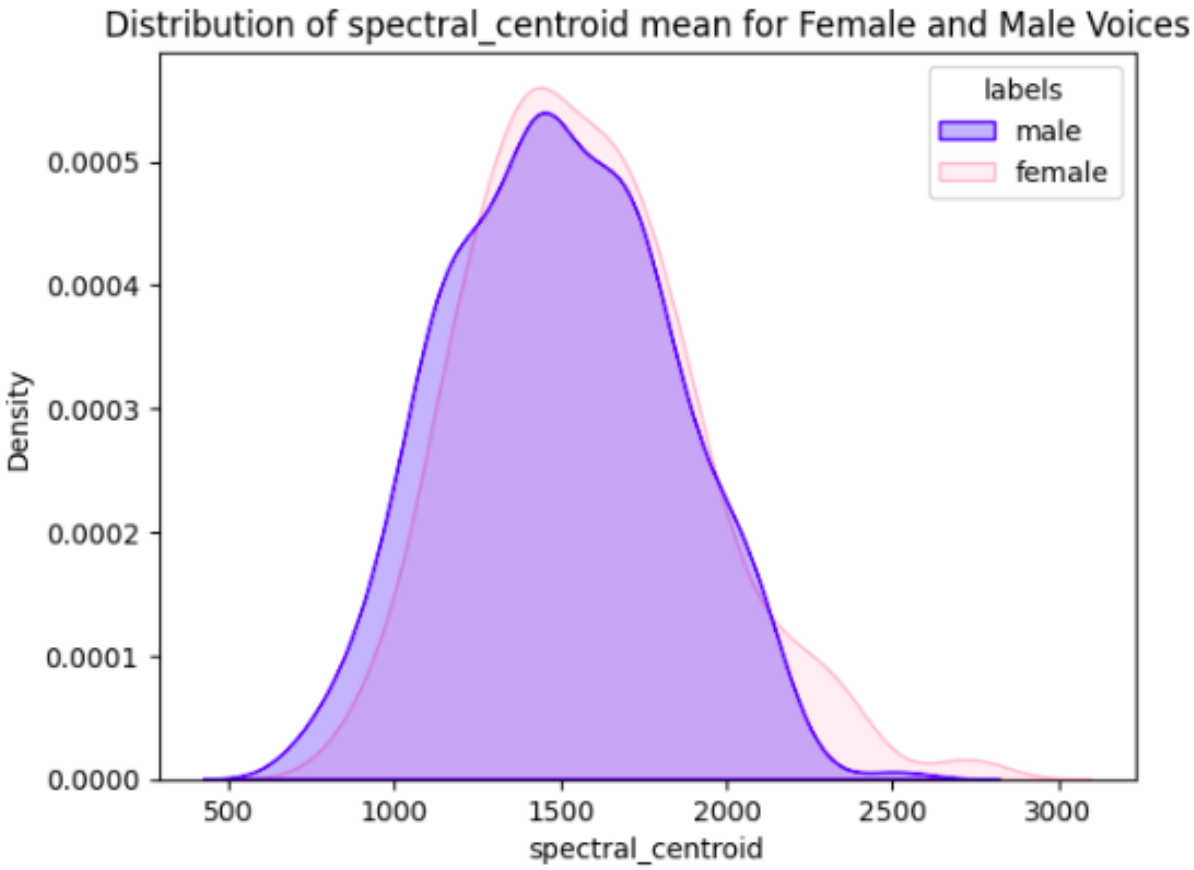
در ادامه برنامه صوت ها را به سگمنت های ۳ ثانیه ای شکسته و برای برچسب مرد یک و زن ۰ را در نظر میگیریم . توابعی که تعریف کردیم را بر روی تمامی داده‌های صوتی اعمال میکنیم . حال می‌خواهیم بررسی کنیم که کدام یک از ویژگی‌های قبلتر استخراج کرده‌ایم ، تفکیک بهتری بین نمونه‌های دو کلاس ایجاد میکند .

```
def zero_crossing_rate(y):
    zero_crossing_rate = np.mean(librosa.feature.zero_crossing_rate(y=y).T,
axis=0)
    return zero_crossing_rate[0]

feature_df['zero_crossing_rate'] =
feature_df['preprocessed_segments'].apply(zero_crossing_rate)

import seaborn as sns

sns.kdeplot(data=feature_df, x='zero_crossing_rate', hue='labels', fill=True,
palette={'female': 'pink', 'male': 'blue'})
plt.title('Distribution of Zero-Crossing Rate for Female and Male Voices')
plt.xlabel('Zero-Crossing Rate')
plt.ylabel('Density')
plt.show()
```

این قطعه کد نموداری رسم میکند که توزیع داده‌های کلاس مرد و زن را در یک نمودار بر اساس ویژگی ZCR نمایش می‌دهد . نمودار حاصل بدین صورت می‌باشد



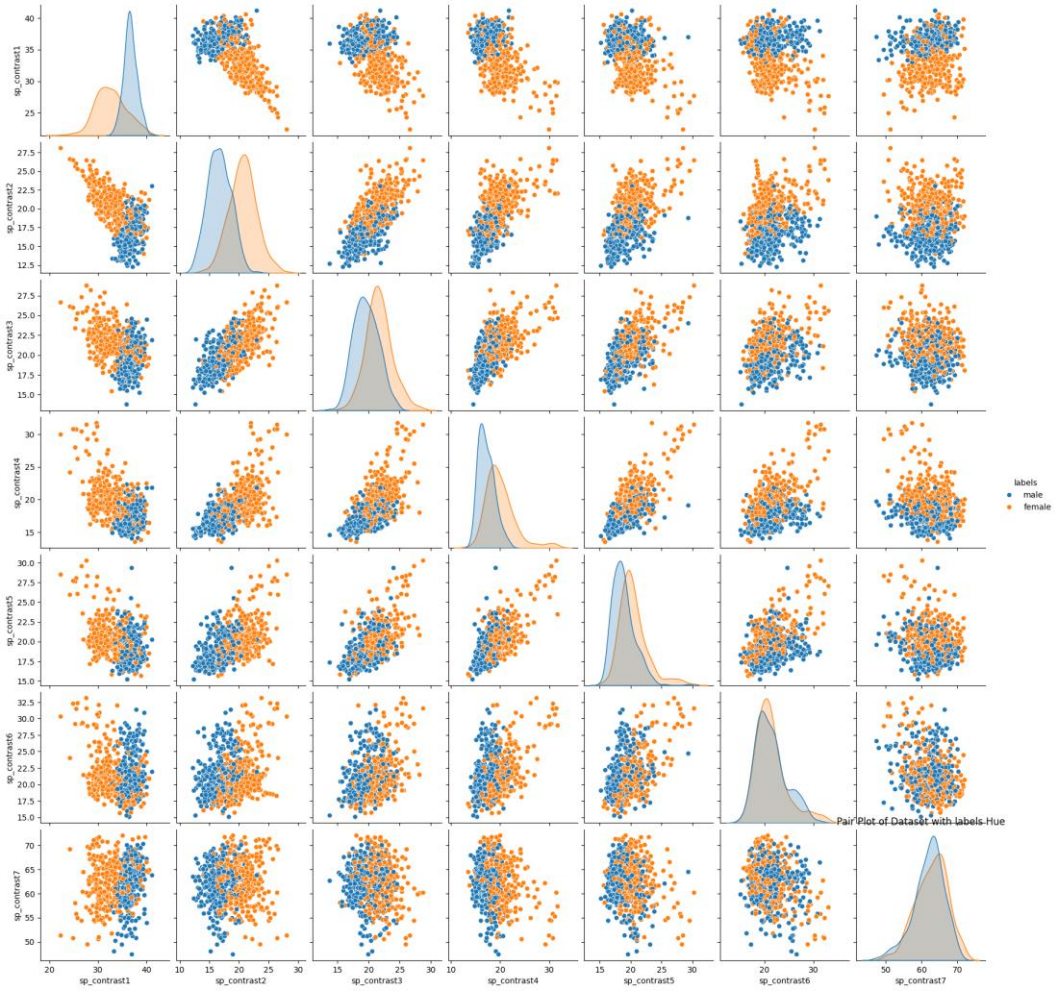
همانطور که مشخص است ، این ویژگی نمی‌تواند به تنهایی تفکیکی بین دو کلاس مختلف ایجاد کند و حال همین نمودار ها را برای مابقی ویژگی‌ها نیز رسم میکنیم .



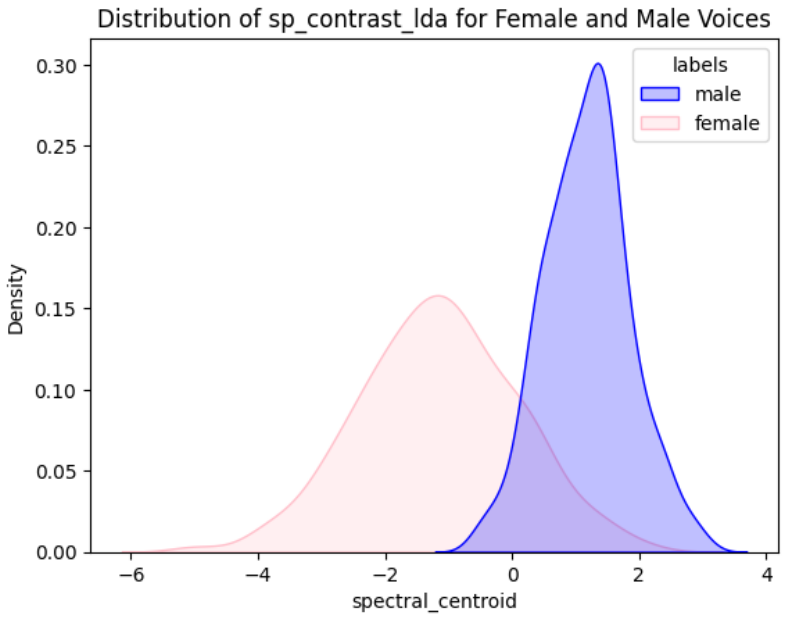
ویژگی Spectral centroids

برای ویژگی scpetral centroid ما مقادیر مختلفی داریم که نمودار فوق میانگین همه ی آن‌ها برای تفکیک دو کلاس را نمایش می‌دهد . اما خوب است که تک تک آن‌ها را با یکدیگر نیز مقایسه کنیم .

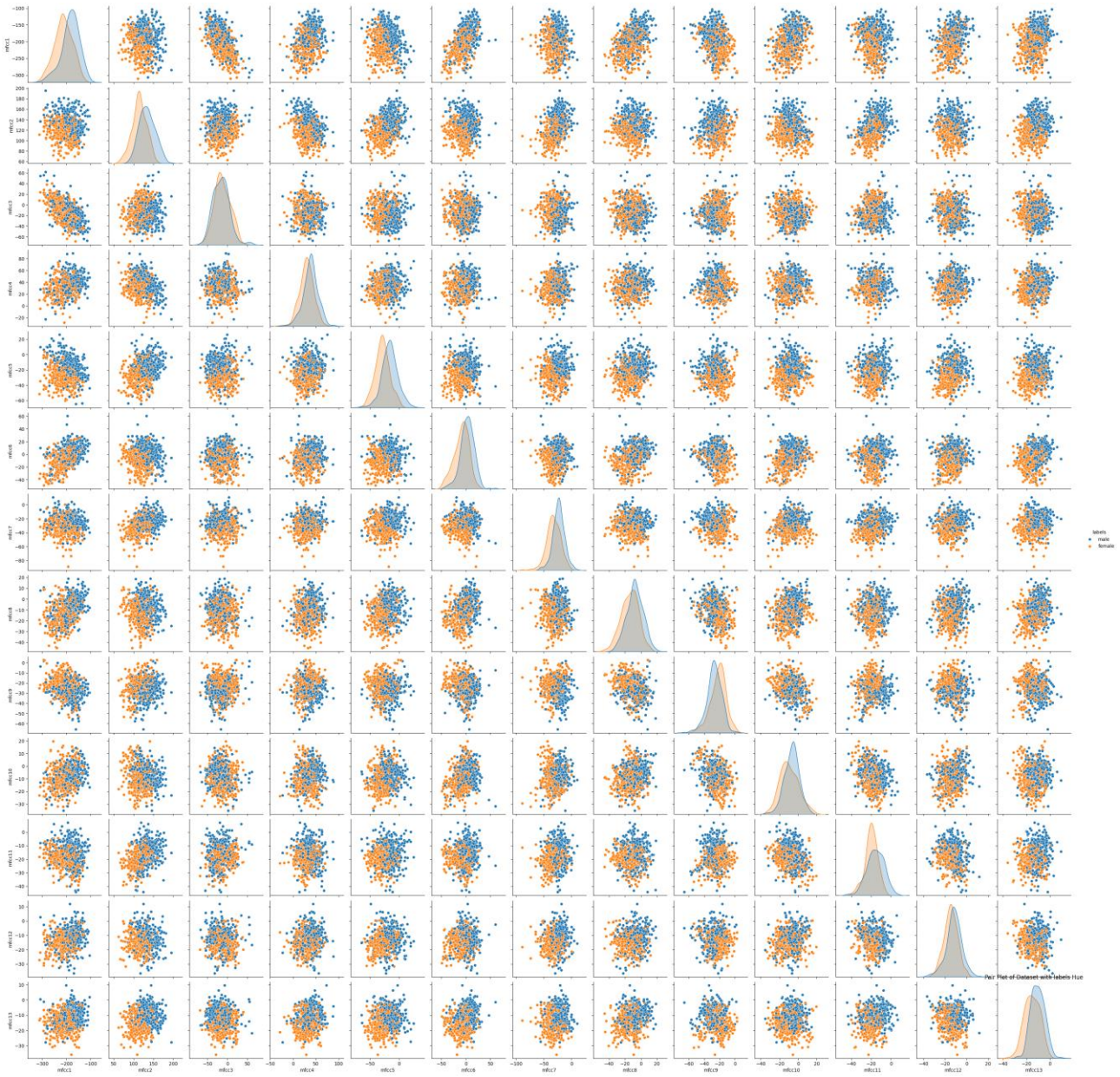
در زیر pairplot بین برخی از ویژگی ها برای تفکیک دو کلاس نمایش داده شده است. Pairplot زیر ترکیب دو ویژگی برای تفکیک دو کلاس مرد و زن را نمایش می‌دهد .



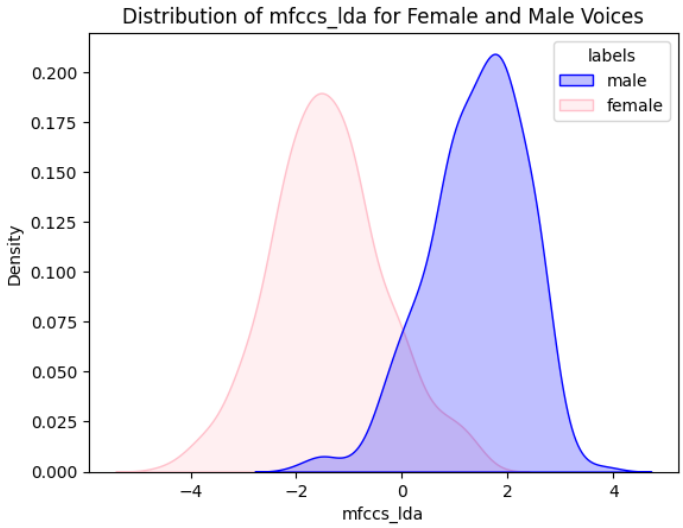
می‌توانیم که تمامی ویژگی‌های مربوط به Spectral Contrast را با استفاده از LDA به ویژگی‌های کمتری تبدیل کنیم . نتیجه این به صورت زیر می‌باشد :



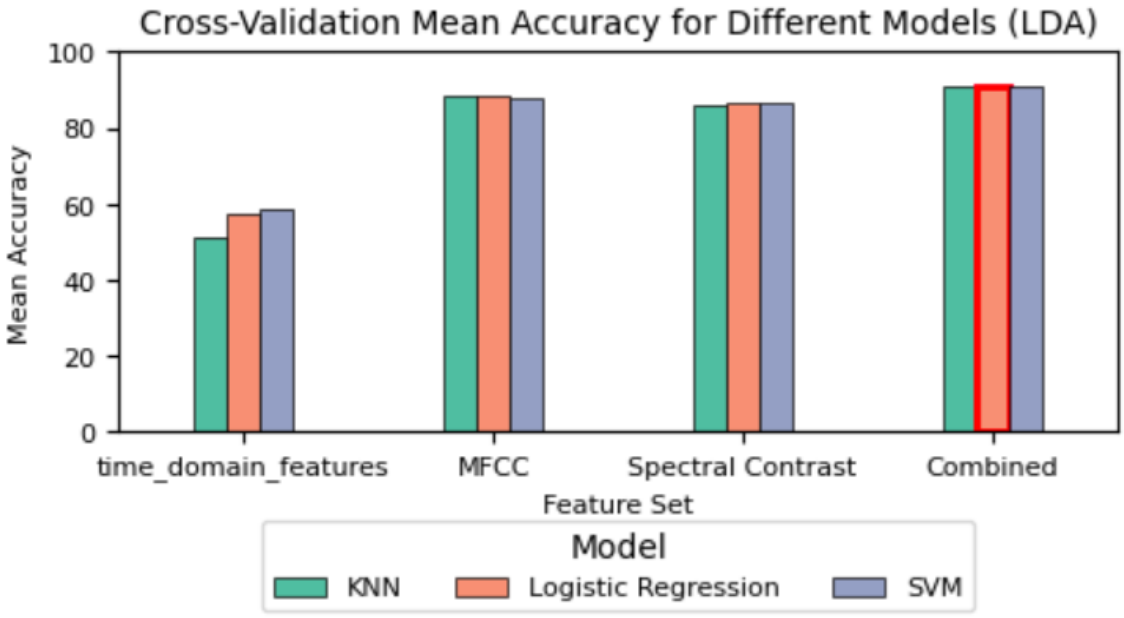
مشاهده می‌شود که یک ترکیب خطی از ویژگی‌های مربوط به Spectral contrast تا چه حد می‌تواند بین نمونه‌های دو کلاس تفکیک ایجاد کند . حال به همین صورت برای مابقی ویژگی‌ها تحلیل را انجام داده‌ایم (درفایل نوتبوک مروبطه تمامی ویژگی‌های تحلیل شده‌اند). یکی دیگر از ویژگی‌های خوب برای تفکیک جنسیت کلاس‌های ، MFCC می‌باشد . این روش نیز چندین ویژگی تولید می‌کند که در تصاویر زیر ابتدا pairplot این ویژگی‌ها و سپس نتیجه LDA بر روی آن‌ها نمایش داده شده است .



در تصویر زیر نتیجه حاصل از LDA روی ویژگی‌های MFCC نمایش داده شده است

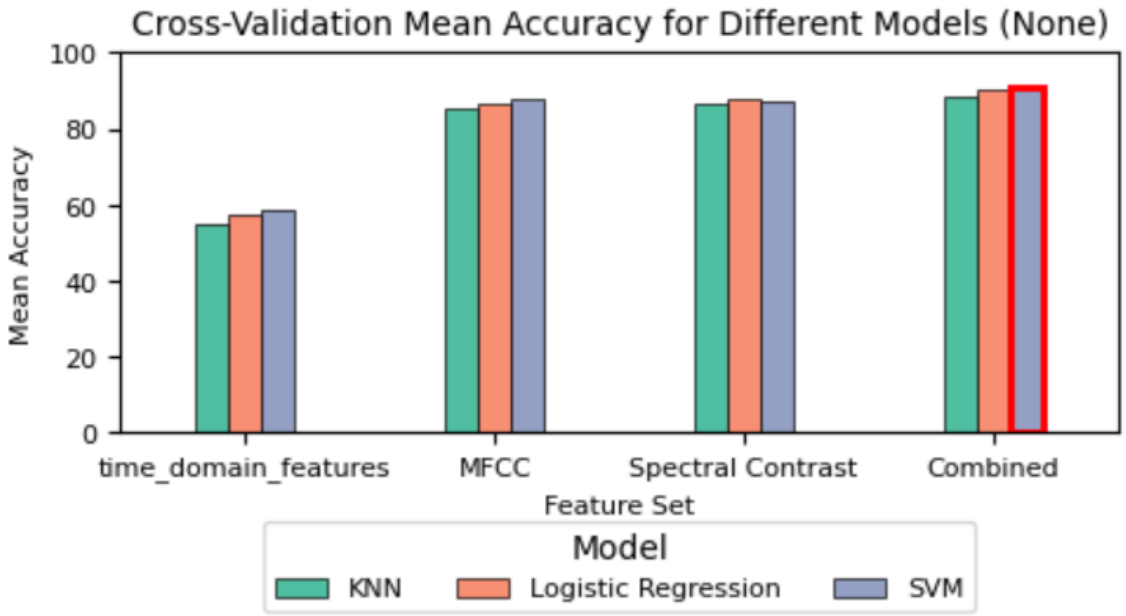


به همین صورت ما تمامی ۶۷ ویژگی را بررسی کرده‌ایم و بهترین آن‌ها را برای کار خود انتخاب کرده‌ایم . همچنین با استفاده از یکسیری مدل classifier ساده سعی کرده‌ایم که کار انتخاب ویژگی را بهتری انجام دهیم . تأثیر و نتایج هر یک از این ویژگی‌ها در عمل‌کرد یک مدل classifier در نمودار های زیر قابل مشاهده است :



در این نمودار مشخیص است که با کاهش ویژگی‌ها با استفاده از روش LDA ما بهترین عمل‌کرد زمانی داریم که هم از ویژگی های MFCC و هم از ویژگی‌های Spectral Contrast استفاده کنیم . اگر از هیچ

روش کاهش ابعادی استفاده نکنیم و تنها بر اساس ویژگی‌های انتخاب شده یک classifier آموزش دهیم ، نتیجه آن بدین صورت است :



کاهش ویژگی می‌تواند منجر به افزایش زمان آموزش و ارزیابی شود .

۴_۱_۴ آموزش مدل بر روی ویژگی‌های انتخاب شده و ارزیابی آن

ویژگی‌هایی که برای آموزش مجدد انتخاب کرده‌ایم بدین صورت می‌باشد :

mfcc_1 , mfcc_2, mfcc_3, mfcc_4 , mfcc_5, mfcc_6 , mfcc_7,
spectral_contrast_1, spectral_contrast_2, spectral_contrast_3,
spectral_contrast_4, spectral_contrast_5, spectral_contrast_6,
spectral_contrast_7

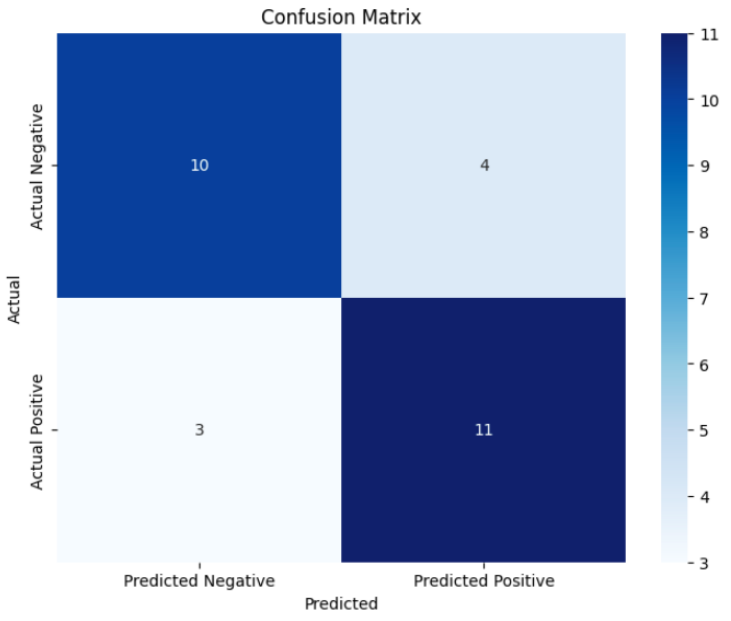
از بین ویژگی‌هایی که برای داده‌ها آموزشی خود انتخاب کرده‌ایم ، ابتدا تمام آن‌ها را لود کرده و سپس ویژگی‌های فوق را از بین آن‌ها جدا می‌کنیم .

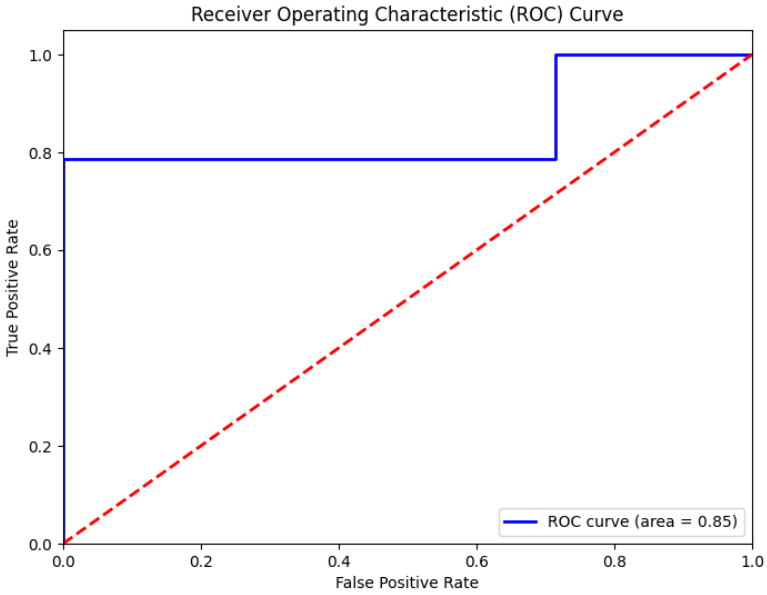
```
MFCC_columns = ['mfcc_1', 'mfcc_2', 'mfcc_3', 'mfcc_4', 'mfcc_5', 'mfcc_6', 'mfcc_7',
                'mfcc_8', 'mfcc_9', 'mfcc_10', 'mfcc_11', 'mfcc_12', 'mfcc_13']

Sp_contrast_columns = ['spectral_contrast_1', 'spectral_contrast_2', 'spectral_contrast_3',
                       'spectral_contrast_4', 'spectral_contrast_5', 'spectral_contrast_6', 'spectral_contrast_7']

selected_features = MFCC_columns + Sp_contrast_columns + ['label']

x_train_selected = train_df[selected_features]
```

در این کد ، ما ویژگی‌هایی که انتخاب کرده‌ایم را انتخاب می‌کنیم . سپس دیتاست جدید را آماده می‌کنیم

```
x_train, y_train = np.array(x_train_selected.drop("label", axis=1)), np.array(train_df['label'])

col_mean = np.nanmean(x_train, axis=0)
inds = np.where(np.isnan(x_train))
x_train[inds] = np.take(col_mean, inds[1])

# Scale features
std_scaler = StandardScaler()
x_train_scaled = std_scaler.fit_transform(x_train)
```

پس از آموزش مدل بر روی داده‌های جدید ، آن را ارزیابی می‌کنیم . روش ارزیابی مشابه روشی است که در قسمت قبلی استفاده کرده‌ایم .نتایج ارزیابی بدین صورت می‌باشد .





همانطور که مشاهده می‌شود ، پس از انتخاب زیر مجموعه‌ای از ویژگی‌ها ، عمل‌کرد مدل بسیار بهتر می‌شود و تا حد خوبی می‌تواند حدود ۸۵ درصد افراد هر جنسیت را تشخیص دهد . نکته مهمی که باید یادآوری کرد این است که داده‌های تستی صدای افرادی بودند که در زمان آموزش هیچ بخشی از صدایشان جز داده آموزشی نبوده است . بنابراین می‌توان گفت که ارزیابی انجام شده ، قابل اعتماد می‌باشد .

۴_۱_۵ نتیجه‌گیری

در این بخش پروژه سعی شد که با استفاده از الگوریتم های کلاسیک، ویژگی‌های صدای آقایان و خانم‌ها آموزش داده شوند تا مدل بتواند تفکیک خوبی بین صدای افراد بر اساس جنسیت انجام دهد . در ابتدا که مدل را بر روی ۶۷ ویژگی (لزوماً نه مستقل) آموزش داده‌ایم ، سبب ایجاد بایاس مدل به سمت کلاس آقایان شد و در نتیجه عمل‌کرد مدل بسیار ضعیف بود . در عین حال کار کردن با ۶۷ ویژگی نیازمند صرف زمان

زیادی بود . به همین دلیل پس از مشاهده چنین وضعییتی ، انتخاب ویژگی و کاهش ابعاد را پیاده‌سازی کردیم . همانطور که مشاهده می‌شود ، پس از آن ، به دقتی حدود ۸۵ درصد رسیده‌ایم و مدل به سمت کلاس خاصی بایاس پیدا نکرده است .

۴_۲ احراز هویت(طبقه‌بندی ۶ کلاسه)

هدف این بخش، پیاده‌سازی تشخیص هویت به روش close-set است. در این روش، مدل طبقه‌بند با استفاده از نمونه‌های ۶ نفر آموزش داده می‌شود و هدف آن پیش‌بینی کلاس نمونه ورودی بر اساس این ۶ کلاس است.

۴_۲_۱ آماده‌سازی داده

نتایج این بخش نیز بر اساس دیتاست آماده شده از صدای دانشجویان است که در آن نسبت دانشجویان دختر و پسر برابر است. به همین دلیل تعداد کل افراد موجود در این دیتاست ۵۶ نفر است که ۲۸ نفر از آنان دانشجویان پسر و ۲۸ نفر دیگر دانشجویان دختر هستند. اما از آنجایی که فایل‌های صوتی مختلفی از یک دانشجو وجود دارد؛ تعداد نمونه‌های صوتی متعلق به هر فرد با هم متفاوت است. به همین دلیل لازم است؛ پس از انتخاب این ۶ نفر به صورت رندم، دیتاست مناسبی که در آن تعداد نمونه‌های همه‌ی افراد با هم برابر هستند ایجاد کنیم. تکه کد زیر، متعادل کردن تعداد نمونه‌های مربوط به هر دانشجوی انتخاب‌شده را نشان می‌دهد. در واقع از بین دانشجویان انتخاب شده، به تعداد حداقل نمونه‌های موجود برای هر فرد، برای بقیه افراد انتخاب می‌کنیم.

```
from sklearn.utils import shuffle
filtered_data = data[data['student_id'].isin(selected_students)]
# Ensure each student has an equal number of samples
```

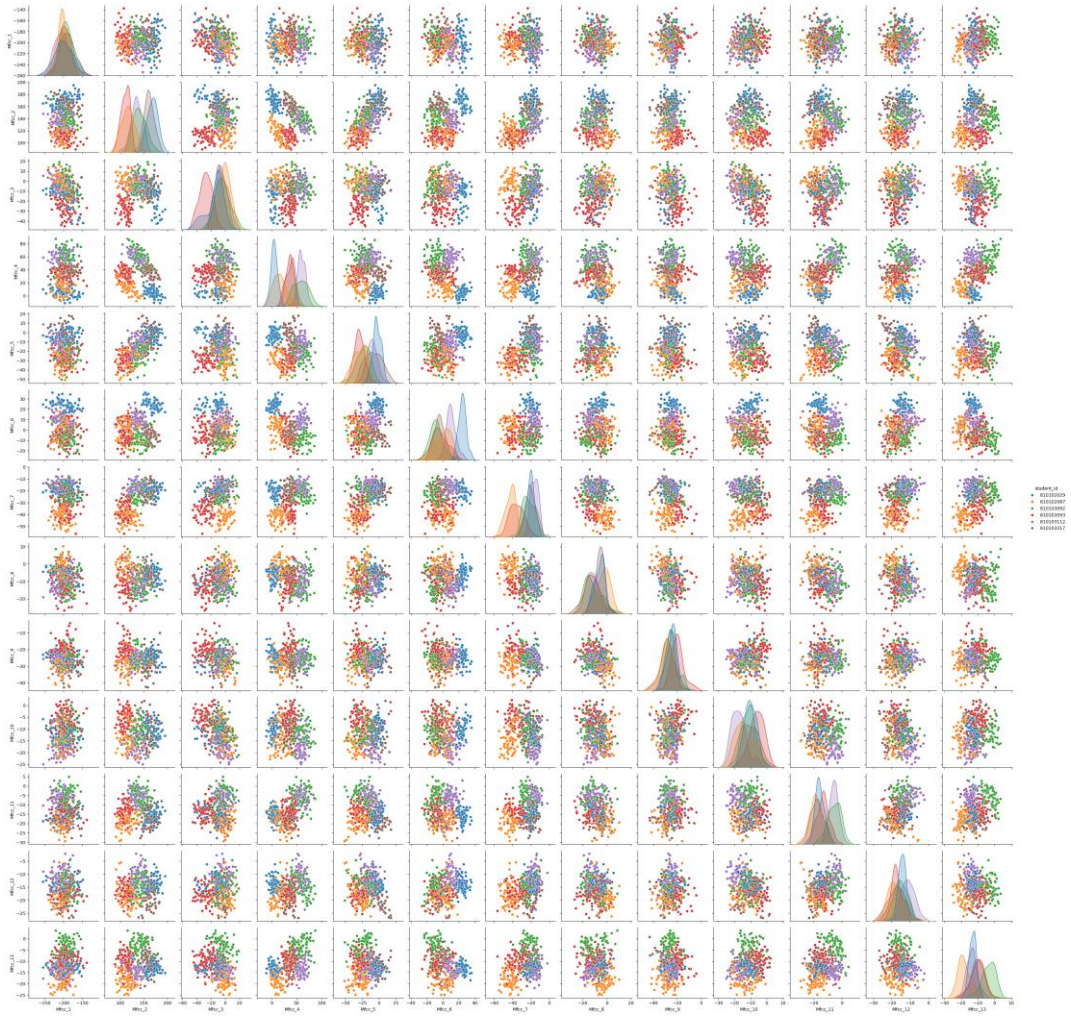
```
balanced_data = filtered_data.groupby('student_id').apply(lambda x:
x.sample(n=filtered_data['student_id'].value_counts().min(),
random_state=42)).reset_index(drop=True)
balanced_data = shuffle(balanced_data, random_state=42)
balanced_data.reset_index(drop=True, inplace=True)

print(balanced_data['student_id'].value_counts())
```

۴_۳ بررسی ویژگی‌ها و Visualization آن‌ها

برای بررسی توزیع نمونه‌های صوتی دانشجویان انتخاب‌شده در فضای ویژگی‌ها، از pair plot استفاده شده است. در برخی از موارد نیز از TSNE برای کاهش ابعاد بردارهای ویژگی برای نمایش آن‌ها در فضای دو بعدی بررسی شده است.

۴_۳_۱ بصری‌سازی ویژگی‌های MFCC و کاربرد در تشخیص هویت

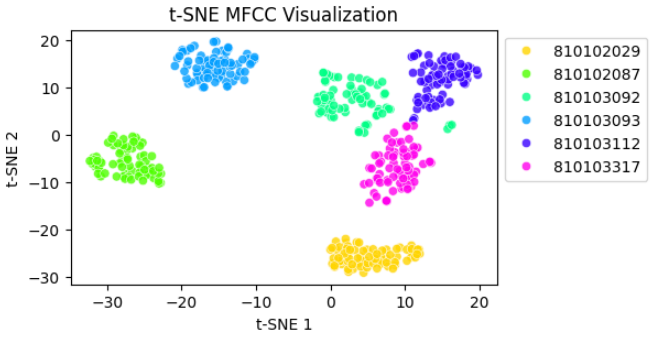
شکل زیر، نتایج فیلترهای مختلف در ویژگی MFCC را نشان می‌دهد. همان‌طور که در این شکل مشخص شده است، میتوان از ویژگی MFCC برای تشخیص هویت استفاده کرد؛ زیرا مولفه‌های مختلف آن، جداسازی‌های خوبی بین افراد مختلف ایجاد می‌کند.



MFCC1 (یا همان ضریب اول) در واقع با انرژی سیگنال مرتبط است. این ضریب، مقدار کلی طیف یا به‌عبارتی انرژی کلی سیگنال صوتی را نشان می‌دهد و گاهی به عنوان مولفه‌ی DC شناخته می‌شود. ضرایب بعدی (MFCC2، MFCC3، و غیره) جزئیات بیشتری از شکل طیفی را نشان می‌دهد. اما ضرایب خیلی بالا (مانند MFCC13) نیز حساسیت بیشتری به نویز موجود در سیگنال صوتی دارند و به همین دلیل، اطلاعاتشان برای تفکیک کلاس‌های مختلف مفید نیستند.

برای نمایش داده‌های با ابعاد بالا می‌توان از تکنیک کاهش ابعاد TSNE برای کاهش ابعاد داده به منظور نمایش آن‌ها در دو بعد استفاده کرد. داده‌های MFCC مربوط به نمونه‌های دانشجویان مختلف توسط

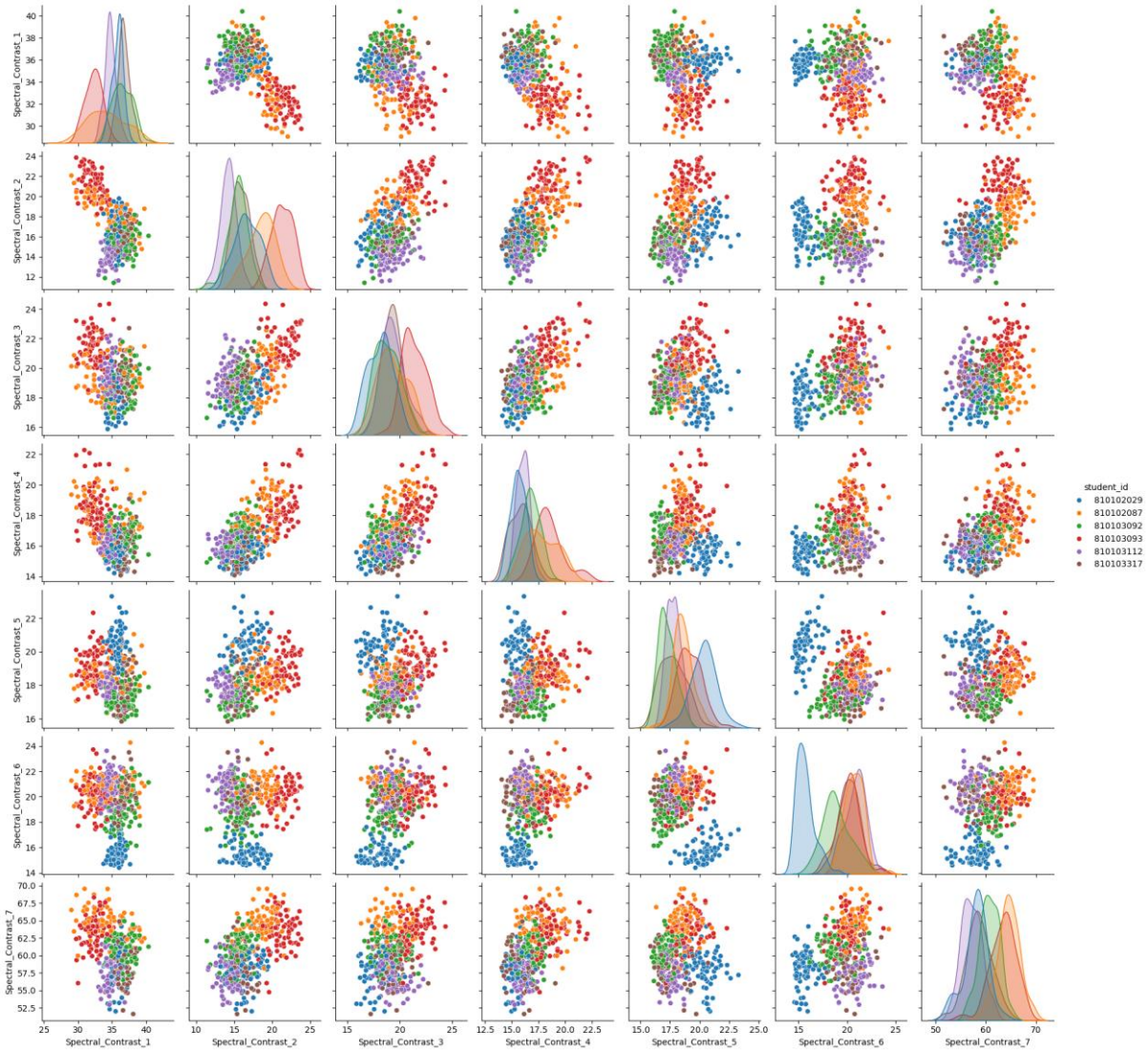
تکنیک کاهش بعد TSNE، در دو بعد نمایش داده شده است. همانطور که این شکل هم نشان میدهد؛ ویژگی‌های MFCC جداپذیری خوبی بین نمونه‌های دانشجویان مختلف، ایجاد می‌کند.



در ادامه از تمامی ضرایب MFCC استخراج شده(۱ تا ۱۳) استفاده شده و بررسی اثر اعمال تکنیک‌های کاهش بعد PCA و LDA نیز بر دقت طبقه‌بندی بررسی شده است.

۴_۳_۲ بصری‌سازی ویژگی‌های Spectral Contrast و کاربرد در تشخیص هویت

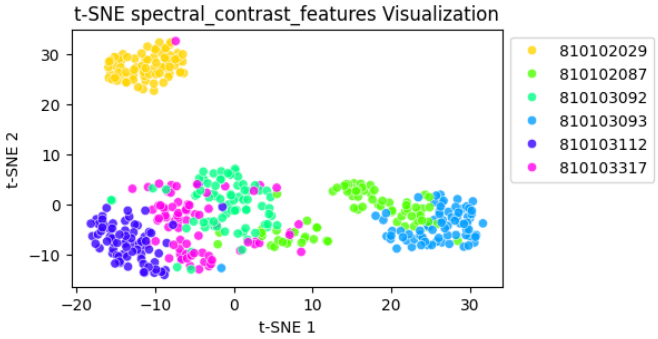
شکل زیر، توزیع نمونه‌های ۶ دانشجو در فضای ویژگی‌های Spectral Contrast را نشان می‌دهد. این ویژگی‌ها، تفاوت در دامنه بین قله‌ها و دره‌ها یک طیف صدا در باندهای فرکانسی مختلف را نشان می‌دهد.



Spectral Contrast_1 تفاوت‌های طیفی در فرکانس‌های پایین را اندازه‌گیری می‌کند و اغلب نمایانگر تفاوت‌های طیفی گسترده‌تر است. Spectral Contrast_7 تفاوت‌های طیفی در فرکانس‌های بالا سیگنال صوتی را اندازه‌گیری می‌کند. برای تشخیص هویت و تفکیک بین نمونه‌ها، ویژگی‌های تضاد طیفی مرتبه پایین‌تر (مانند Spectral Contrast در باندهای ۱ تا ۳) ممکن است بهتر باشند. این ویژگی‌ها تفاوت‌های بزرگ‌تر و مهم‌تر در طیف را نشان میدهند که معمولاً قابل تمایزتر و کمتر تحت تأثیر نویز هستند.

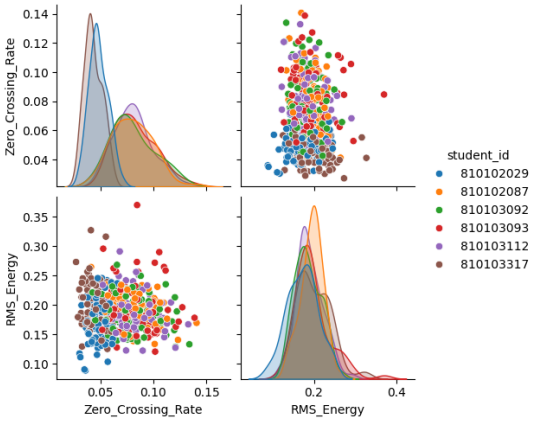
ویژگی‌های تضاد طیفی مرتبه بالاتر (مانند Spectral Contrast در باندهای۶ تا ۷) ممکن است برای نشان دادن جزئیات دقیق‌تر، مناسب باشند؛ اما می‌توانند به نویز حساستر باشند و برای کاربرد تشخیص هویت و یا جنسیت کمتر مفید باشند.

داده‌های Spectral Contrast مربوط به نمونه‌های دانشجویان مختلف توسط تکنیک کاهش بعد TSNE، در دو بعد نمایش داده شده است. همانطور که این شکل هم نشان میدهد؛ ویژگی‌های Spectral Contrast جداپذیری نسبتا خوبی بین نمونه‌های دانشجویان مختلف، ایجاد می‌کند. اما به خوبی جداسازی به کمک ویژگی‌های MFCC نمی‌باشد. در ادامه تمامی ویژگی‌های Spectral Contrast که برای ۶ باند فرکانسی مختلف می‌باشد؛ برای احراز هویت استفاده شده و ترکیب آن با سایر ویژگی‌ها و تکنیک‌های کاهش بعد بررسی شده است.



۴_۳_۳ بصری‌سازی ویژگی‌های حوزه زمانی

شکل زیر توزیع نمونه‌ها در فضای ویژگی‌های زمانی مانند میانگین rms انرژی و نرخ عبور از صفر را نشان می‌دهد. همانطور که در شکل نیز مشخص است؛ این ویژگی‌ها به دلیل حساسیت به نویز و محتوای سیگنال صوتی، تفکیک‌پذیری خوبی بین نمونه‌های افراد مختلف نمی‌تواند ایجاد کند. در ادامه، عملکرد این ویژگی‌ها نیز در تشخیص هویت بررسی شده است که دقت پایین آن‌ها را در کاربردهای تشخیص‌هویت نشان می‌دهد.



۴_۴ ملاحظات لازم برای انجام CrossValidation

برای انجام CrossValidation و در نظر گرفتن مدل‌ها و تکنیک‌های مختلف کاهش ابعاد بردار ویژگی تابعی تعریف شده است که روشی کارآمد برای مقایسه مدل‌های طبقه‌بندی و ارزیابی ویژگی‌های مختلف و تکنیک‌های کاهش ویژگی ارائه می‌کند.

بخش‌های مهم این تابع:

۱) با استفاده از کلاس KFold از `sklearn.model_selection`، داده‌ها به k فولد تقسیم می‌شوند (پیش‌فرض ۵ است). این تضمین می‌کند که هر فولد یک بار به عنوان مجموعه تست و به عنوان بخشی از مجموعه تمرینی k-1 بار استفاده می‌شود.

۲) تابع نرمالسازی به مجموعه ویژگی‌ها با استفاده از `StandardScaler` برای نرمال‌سازی داده‌ها اعمال می‌شود.

۳) اگر تابع کاهش ویژگی ارائه شده باشد، ابتدا با استفاده از داده‌های آموزشی برازش (fit) می‌شود و سپس بر روی داده‌های تست اعمال می‌گردد. این مرحله تضمین می‌کند که تکنیک کاهش ویژگی به درستی در مجموعه آموزشی آموزش دیده است و سپس به داده‌های تست اعمال می‌شود.

۴) میانگین امتیاز دقت در همه فولدها برای ارائه یک معیار عملکرد کلی محاسبه می‌شود. علاوه بر این، ماتریس آشفتگی میانگین برای تجسم عملکرد ترسیم می‌شود.

۴_۵ نتایج الگوریتم‌های طبقه‌بند

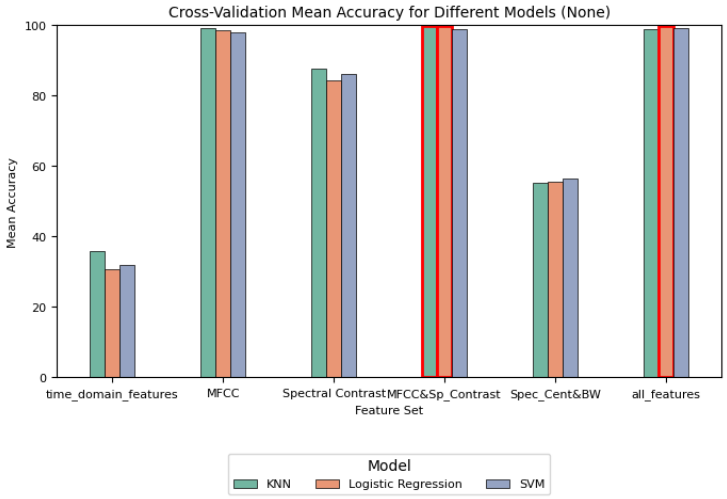
در این بخش با استفاده از توابع نوشته‌شده، دقت ۳ مدل طبقه‌بند: KNN(k=5)، Logistic Regression و SVM مورد بررسی قرار گرفته‌است. در هر یک از این مدل‌ها، مجموعه‌های ویژگی‌های مختلف و نیز تاثیر اعمال تکنیک‌های کاهش ابعاد PCA و LDA بررسی شده است. برای هر یک از این حالت‌ها، میانگین ماتریس آشفتگی حاصل از K-fold-CrossValidation(k = 4)، میانگین دقت و انحراف معیار مربوط به آن محاسبه شده است؛ اما به دلیل تعداد زیاد آن‌ها، در هر یک از بخش‌های زیر نمودار

میله‌ای برای مقایسه‌ی میانگین دقت حاصل از CrossValidation آورده شده است. علاوه بر این، جدول شامل ۵ مدل و تکنیکی که به بیشترین دقت طبقه‌بند با استفاده از Crossvalidation رسیده اند ارائه شده است.

۴_۵_۱ مجموعه اول از نمونه‌های صوتی انتخاب تصادفی ۶ دانشجو

۱) نتایج بدون اعمال تکنیک‌های کاهش بعد:

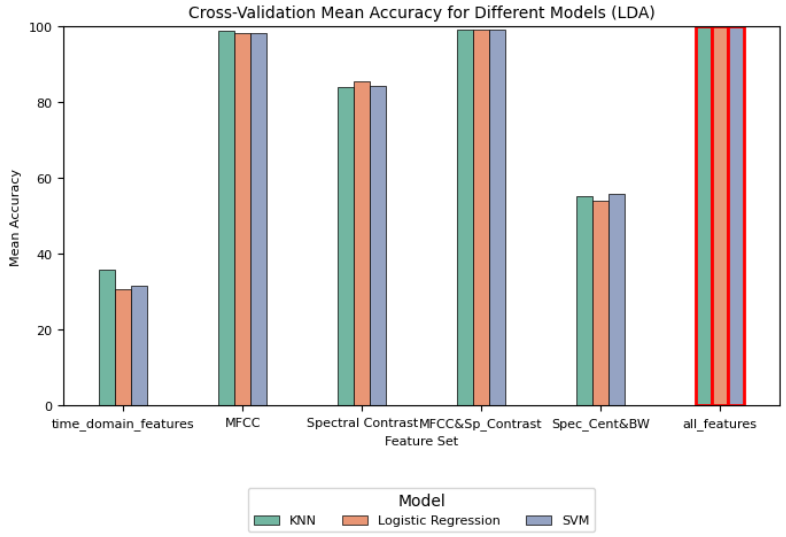
همانطور که نتایج نشان می‌دهد طبقه‌بندی با استفاده از ویژگی‌های MFCC و هر ترکیبی از ویژگی‌ها که شامل ویژگی‌های MFCC باشد؛ دقت بالایی دارد. بالاترین میانگین دقت حاصل از CrossValidation در این نمودار با رنگ قرمز مشخص شده است.



۲) نتایج با اعمال تکنیک کاهش بعد LDA:

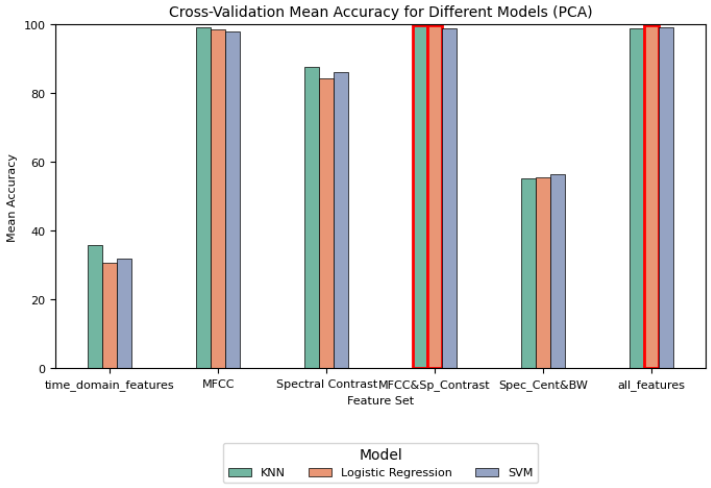
با توجه به اینکه ۶ کلاس مختلف داریم، با اعمال روش کاهش ابعاد LDA، ۵ ویژگی جدید از ترکیب خطی ویژگی‌های ورودی تولید می‌شود که قابلیت جداسازی خوبی بین کلاس‌های مختلف ایجاد می‌کند. همانطور که در شکل زیر نشان‌داده شده است، با اعمال تکنیک کاهش بعد LDA به تمامی ویژگی‌های زمانی و فرکانسی (MFCCها، Spectral Contrastها، Spectral Centroid، Spectral BandWidth و Zero Crossing Rate و mean rms energy)، میانگین دقت CrossValidation طبقه‌بندهای مختلف در صورت به ۱۰۰ درصد رسیده است. این نمودار نیز نشان می‌دهد که استفاده از ویژگی‌های MFCC و اعمال LDA

به آن می‌توان به دقت نزدیک به ۱۰۰ رسید. از طرفی، ناکارآمدی برخی از ویژگی‌های حوزه زمان مانند Zero crossing rate و mean rms energy و همین‌طور برخی از ویژگی‌های فرکانسی نظیر Spectral Centroid و Spectral Bandwidth را نشان می‌دهد. زیرا این ویژگی‌های صوتی، تمایز خوبی بین صوت افراد مختلف ایجاد نمی‌کنند. اما ویژگی‌های MFCC، جزئیات طیف‌های صوتی افراد مختلف را به خوبی استخراج می‌کنند.



۳) نتایج با اعمال تکنیک کاهش بعد PCA:

با اعمال تکنیک کاهش بعد PCA به مجموعه‌ی ویژگی‌ها، ویژگی‌های جدیدی تولید می‌شوند که پراکندگی داده‌ها در آن زیاد باشد. شکل زیر، نتایج حاصل از دقت طبقه‌بندی با اعمال تکنیک PCA به مجموعه ویژگی‌ها برای طبقه‌بندهای مختلف را نشان می‌دهد. در صورت استفاده از تمامی ویژگی‌ها، در برخی از مدل‌ها اندکی دقت کاهش پیدا می‌کند اما همچنان دقت مدل logistic regression حفظ شده و دقت مدل KNN مشابه با وضعیتی که از تکنیک کاهش بعد استفاده نمی‌شود، بیشینه است.



۴) مقایسه کلی:

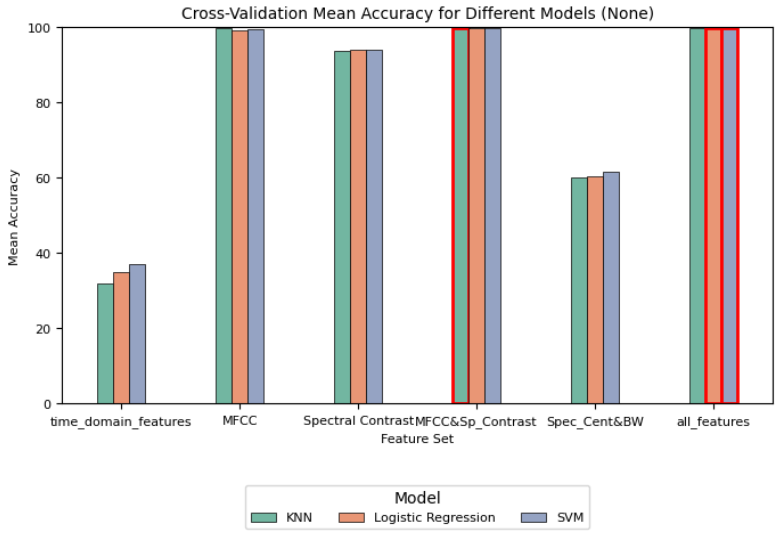
جدول زیر ۵ مدل و تکنیکی که به بیشترین دقت طبقه‌بند با استفاده از Crossvalidation رسیده اند را نشان می‌دهد. نتایج نشان می‌دهد که طبقه‌بندی با استفاده از تمام ویژگی‌های حوزوه زمانی و فرکانسی(۲۴ ویژگی) و با اعمال روش کاهش ابعاد ویژگی LDA به دقت بسیار خوبی می‌رسد. علاوه‌بر این، روش LDA باعث ساده شدن مدل می‌شود زیرا با داشتن ۶ کلاس مختلف، ۵ ویژگی از ترکیب خطی ویژگی‌ها تولید می‌شود و با استفاده از این ۵ ویژگی، برای این مجموعه از نمونه‌ها، دقت طبقه‌بندی به ۱۰۰ رسیده است.

	Model	Feature Set	Feature Reduction	Mean Accuracy	Std Accuracy
15	KNN	all_features	LDA	100.000000	0.000000
51	SVM	all_features	LDA	100.000000	0.000000
33	Logistic Regression	all_features	LDA	100.000000	0.000000
35	Logistic Regression	all_features	None	99.774775	0.390102
34	Logistic Regression	all_features	PCA	99.774775	0.390102

۴_۵_۲ مجموعه دوم از نمونه‌های صوتی انتخاب تصادفی ۶ دانشجو

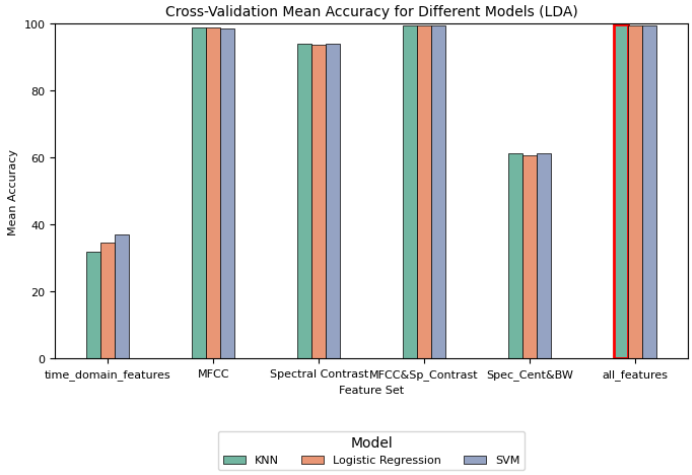
۱) نتایج بدون اعمال تکنیک‌های کاهش بعد:

همانطور که نتایج نشان می‌دهد برای این مجموعه از نمونه‌ها نیز، طبقه‌بندی با استفاده از ویژگی‌های MFCC و هر ترکیبی از ویژگی‌ها که شامل ویژگی‌های MFCC باشد؛ دقت بالایی دارد. بالاترین میانگین دقت حاصل از CrossValidation در این نمودار با رنگ قرمز مشخص شده است.



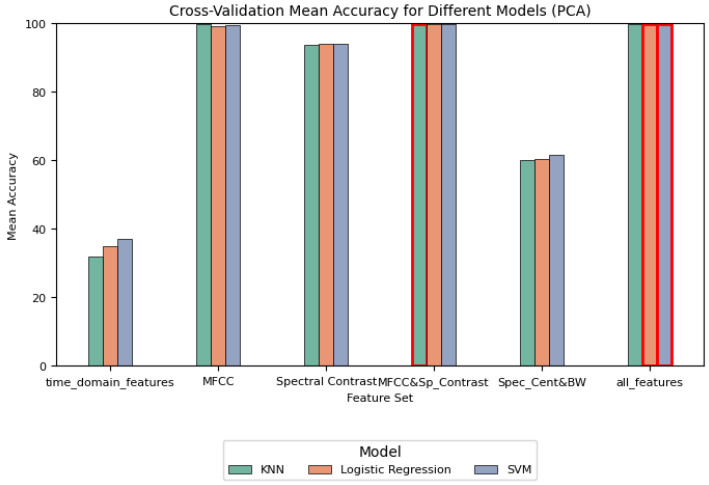
۲) نتایج با اعمال تکنیک کاهش بعد LDA:

با توجه به اینکه ۶ کلاس مختلف داریم، با اعمال روش کاهش ابعاد LDA، ۵ ویژگی جدید از ترکیب خطی ویژگی‌های ورودی تولید می‌شود که قابلیت جداسازی خوبی بین کلاس‌های مختلف ایجاد می‌کند. همانطور که در شکل زیر نشان‌داده شده است، با اعمال تکنیک کاهش بعد LDA به تمامی ویژگی‌های زمانی و فرکانسی (MFCCها، Spectral Contrastها، Spectral Centroid، Spectral BandWidth و Zero Crossing Rate و mean rms energy)، میانگین دقت CrossValidation طبقه‌بند KNN به نزدیک ۱۰۰ درصد رسیده است.



۳) نتایج با اعمال تکنیک کاهش بعد PCA:

با اعمال تکنیک کاهش بعد PCA به مجموعه‌ی ویژگی‌ها، ویژگی‌های جدیدی تولید می‌شوند که پراکندگی داده‌ها در آن زیاد باشد. شکل زیر، نتایج حاصل از دقت طبقه‌بندی با اعمال تکنیک PCA به مجموعه ویژگی‌ها برای طبقه‌بندهای مختلف را نشان می‌دهد. دقت مدل‌ها با استفاده از ویژگی‌های MFCC و یا ترکیب آن با سایر ویژگی‌ها همچنان نسبت به ویژگی‌های زمانی بیشینه است.



۴) مقایسه کلی:

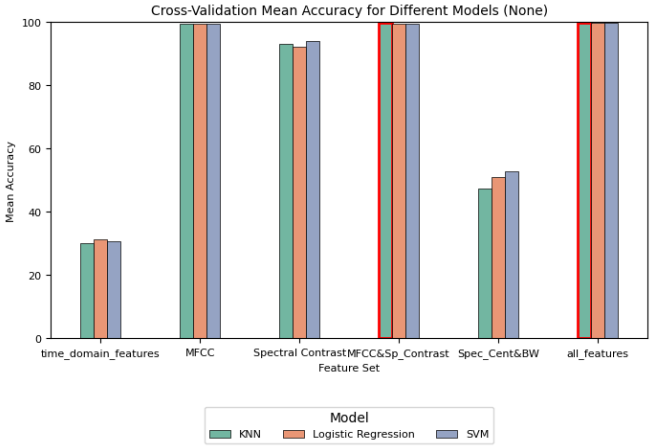
جدول زیر ۵ مدل و تکنیکی که به بیشترین دقت طبقه‌بند با استفاده از Crossvalidation رسیده اند را نشان می‌دهد. نتایج نشان می‌دهد که با این مجموعه از نمونه‌ها(که تعداد نمونه‌ها بیشتر بود) به ازای

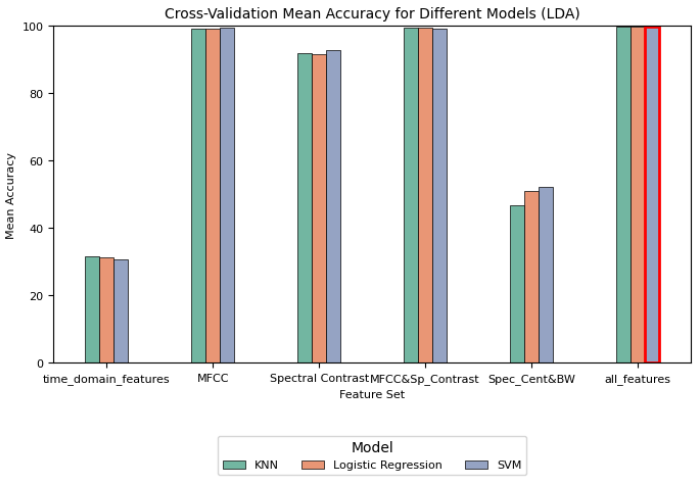
مدل‌های مختلف طبقه‌بندی با استفاده از تمام ویژگی‌های حوزوه زمانی و فرکانسی(۲۴ ویژگی) و با اعمال روش کاهش ابعاد ویژگی PCA و حتی بدون هر گونه کاهش ویژگی، به دقت بسیار خوبی می‌رسد.

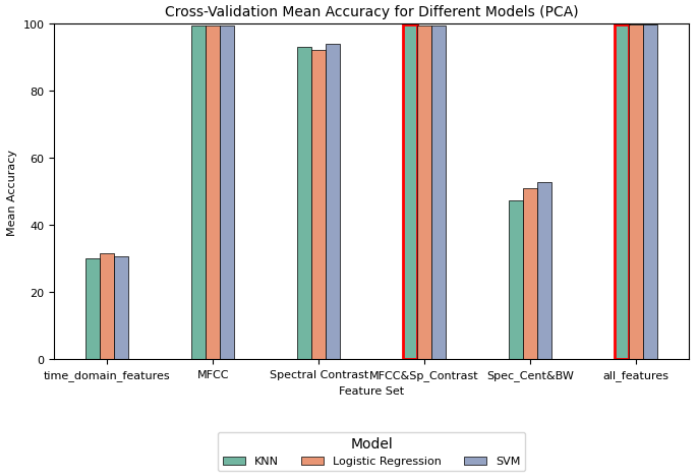
	Model	Feature Set	Feature Reduction	Mean Accuracy	Std Accuracy
53	SVM	all_features	None	99.839744	0.092524
52	SVM	all_features	PCA	99.839744	0.092524
35	Logistic Regression	all_features	None	99.839744	0.177170
34	Logistic Regression	all_features	PCA	99.839744	0.177170
10	KNN	MFCC&Sp_Contrast	PCA	99.839744	0.177170

۳_۵_۴ مجموعه سوم از نمونه‌های صوتی انتخاب تصادفی ۶ دانشجو

نمودارهای زیر نتایج حاصل برای طبقه‌بندهای مختلف و تکینیک‌های طبقه‌بندی را برای مجموعه سوم از داده‌ها، نشان می‌دهد.







جدول زیر ۵ مدل و تکنیکی که به بیشترین دقت طبقه‌بند با استفاده از Crossvalidation رسیده اند را برای این مجموعه از نمونه‌ها نشان می‌دهد.

	Model	Feature Set	Feature Reduction	Mean Accuracy	Std Accuracy
17	KNN	all_features	None	99.906367	0.162177
10	KNN	MFCC&Sp_Contrast	PCA	99.906367	0.162177
11	KNN	MFCC&Sp_Contrast	None	99.906367	0.162177
16	KNN	all_features	PCA	99.906367	0.162177
51	SVM	all_features	LDA	99.812734	0.187266

۶_۴ نتیجه‌گیری

با توجه به نتایج حاصل از بررسی‌ها، می‌توان مشاهده کرد که ویژگی MFCC دارای دقت بالایی در تفکیک نمونه‌های صوتی مختلف است. این ویژگی به تنهایی و یا در ترکیب با سایر ویژگی‌ها، می‌تواند دقت طبقه‌بندهای مختلف را به طور قابل توجهی افزایش دهد. اما ویژگی‌های حوزه زمان به دلیل حساسیت زیاد به نویز و محیط برای انجام طبقه‌بندی مناسب نبوده و دقت بسیار کمی دارند.

۵ خوشه‌بندی

هدف از این بخش انجام خوشه‌بندی نمونه‌های صوتی دانشجویان می‌باشد. ویژگی‌های MFCC و Spectral Contrast دو دسته از ویژگی‌های اصلی صوتی هستند که در این بخش بر اساس آن‌ها خوشه‌بندی انجام شده است. در فایل Clustering.ipynb تمامی این مراحل به ترتیب همراه با خروجی آورده شده است.

۵_۱ آماده‌سازی داده

دیتاست استفاده شده در این بخش، دیتاست آماده شده از صدای دانشجویان است که در آن نسبت دانشجویان دختر و پسر برابر است. به همین دلیل تعداد کل افراد موجود در این دیتاست ۵۶ نفر است که ۲۸ نفر از آنان دانشجویان پسر و ۲۸ نفر دیگر دانشجویان دختر هستند. اما از آنجایی که فایل‌های صوتی مختلفی از یک دانشجو وجود دارد؛ تعداد نمونه‌های صوتی متعلق به هر فرد با هم متفاوت است. این تکه کد، داده‌ها را بر اساس شماره دانشجویی گروه‌بندی می‌کند و به طور تصادفی تعداد مساوی از نمونه‌های صوتی را برای هر فرد نمونه‌برداری می‌کند تا تعداد نمونه‌های منصفانه‌ای از تمامی افراد و گروه‌ها داشته باشیم. این مجموعه داده متعادل برای انجام تجزیه و تحلیل خوشه‌بندی بدون بایاس و دقیق بر روی نمونه های صوتی ضروری است.

```
balanced_data = data.groupby('student_id').apply(lambda x:
x.sample(n=data['student_id'].value_counts().min(),
random_state=42)).reset_index(drop=True)
balanced_data = shuffle(balanced_data, random_state=42)
balanced_data.reset_index(drop=True, inplace=True)
```

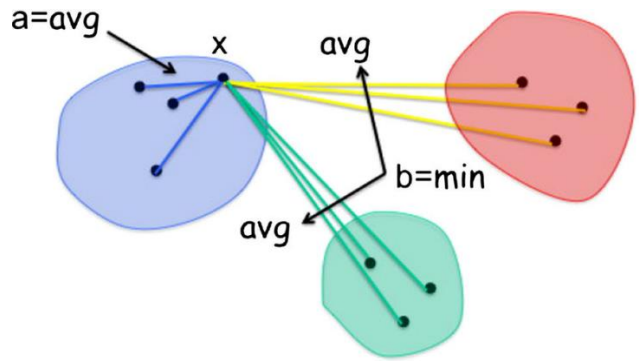
۵_۲ رویکرد خوشه‌بندی

برای انجام خوشه بندی از دو روش Kmeans++ و روش GMM(Gaussian Mixture Model) استفاده شده است. در هر یک از این مدل‌ها ویژگی‌های MFCC و Spectral Contrast به صورت ترکیبی و نیز جداگانه

ارزیابی شده است. برای ارزیابی خوشه‌بندی از Silhouette Score استفاده شده است. علاوه‌بر این تاثیر استفاده از روش‌های کاهش ابعاد بردار ویژگی بر کیفیت خوشه‌بندی مورد بررسی قرار گرفته است.

۵_۲_۱ معیار Silhouette Score

معیار Silhouette Score معیاری است که برای ارزیابی کیفیت خوشه بندی استفاده می شود. این معیار انسجام خوشه ها را اندازه‌گیری می‌کند و می‌تواند مقداری بین ۱- تا ۱+ داشته باشد. هر چه مقدار این معیار بالاتر باشد، نشان‌دهنده خوشه‌های منسجم‌تر است. مقادیر نزدیک به ۱+ نشان می دهد که یک نمونه از خوشه های همسایه دور است و مقادیر منفی نشان می دهد که ممکن است نمونه ها به خوشه اشتباهی اختصاص داده شده باشند. ضریب بر اساس انسجام خوشه و جداسازی خوشه محاسبه می‌شود که به ترتیب میانگین فاصله بین نمونه‌ها و نقاط داده در داخل و بین خوشه ها است. این معیار به صورت زیر محاسبه می‌شود که در آن a میانگین فاصله داخل خوشه‌ای و b میانگین فاصله‌ی بین خوشه‌ای است. شکل زیر مثالی از محاسبه این معیار را نشان میدهد.



$$Silhouette\ Score = \frac{b - a}{\max(a, b)}$$

خوشه‌بندی با مقدار متوسط بیش از ۰.۷ قوی، با مقدار بیش از ۰.۵ معقول و با مقدار بیش از ۰.۲۵ ضعیف در نظر گرفته می‌شود. اما با بزرگ شدن ابعاد بردار ویژگی، رسیدن به خوشه‌بندی با مقدار Silhouette بالا بسیار دشوار می‌شود. زیرا در ابعاد بالا، خوشه‌ها ممکن است شکل‌های پیچیده‌تری به خود بگیرند که

خوشه‌بندی آن‌ها با الگوریتم‌های خوشه‌بندی سخت‌تر می‌شود که منجر به امتیاز Silhouette پایین‌تر می‌شود. از طرفی، با بزرگ شدن ابعاد، داده‌ها نیز sparse شده و فاصله‌ی بین داده‌ها معنای خود را از دست میدهند.

به همین دلیل، تکنیک‌های کاهش ابعاد مانند PCA برای بهبود عملکرد خوشه‌بندی و بررسی آن‌ها می‌توانند مفید باشند.

۵_۲_۲ تابع محاسبه Silhouette برای مدل‌ها و داده‌های مختلف

تابع calculate_silhouette_scores برای محاسبه امتیازات silhouette برای تعداد مختلف خوشه با استفاده از روش‌های خوشه‌بندی K-Means یا مدل مخلوط گاوسی (GMM) طراحی شده است. این به ارزیابی کیفیت خوشه‌بندی با اندازه‌گیری میزان تفکیک خوشه‌ها کمک می‌کند تا بهترین تعداد خوشه‌بندی را به ازای مدل‌ها و ورودی‌های مختلف انتخاب کنیم. این تابع آرایه امتیازهای Silhouette متناظر با خوشه‌بندی با تعداد مختلف خوشه‌ها را برمی‌گرداند.

```
from sklearn.preprocessing import StandardScaler, MinMaxScaler
from sklearn.cluster import KMeans
from sklearn.mixture import GaussianMixture
from sklearn.metrics import silhouette_score

# Function to calculate Silhouette Scores
def calculate_silhouette_scores(model_type, X, n_clusters_range):
    scores = np.zeros(len(n_clusters_range))
    for i, n_clusters in enumerate(n_clusters_range):
        if model_type == 'kmeans':
            model = KMeans(n_clusters=n_clusters, init='k-means++', random_state=42)
        elif model_type == 'gmm':
            model = GaussianMixture(n_components=n_clusters, random_state=42)
        else:
            raise ValueError("Unsupported model type. Use 'kmeans' or 'gmm'.")
        model.fit(X)
        if model_type == 'kmeans':
            labels = model.labels_
        elif model_type == 'gmm':
            labels = model.predict(X)
        scores[i] = silhouette_score(X, labels)
    return scores
```

۵_۲_۳ بررسی تعداد خوشه‌بندی بهینه بر اساس معیار Silhouette

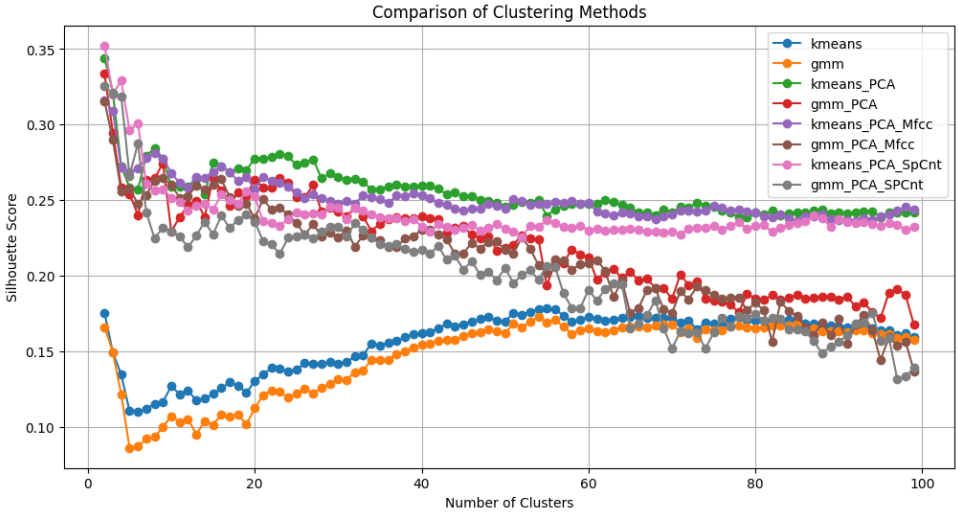
برای انجام خوشه‌بندی ترکیب‌های مختلفی از ویژگی‌ها و تابع کاهش بعد PCA در نظر گرفته شده است. برای تمامی حالت‌ها، نمودار معیار Silhouette به ازای تعداد خوشه‌ها، در بازه‌ی ۲ تا ۱۰۰ با استفاده از تابع `calculate_silhouette_scores(model_type, X, n_clusters_range)` برای هر دو مدل Kmeans++ و GMM محاسبه شده است.

- استفاده از همه‌ی فیچرهای MFCC و Spectral_Contrast (در مجموع ۲۰ ویژگی) بدون استفاده از تابع کاهش بعد
- استفاده از همه‌ی فیچرها به همراه تابع کاهش بعد PCA با ۳ مؤلفه
- استفاده از فیچرهای MFCC به همراه تابع کاهش بعد PCA با ۳ مؤلفه
- استفاده از فیچرهای Spectral Contrast به همراه تابع کاهش بعد PCA با ۳ مؤلفه

قطعه کد زیر نحوه استفاده از تابع و ذخیره اطلاعات آن در dictionary برای رسم را نشان می‌دهد.

```
scaler = StandardScaler()
X = scaler.fit_transform(balanced_data[combined_features])
n_clusters_range = range(2, 100)

# Calculate Silhouette Scores for KMeans and GMM
kmeans_scores = calculate_silhouette_scores('kmeans', X, n_clusters_range)
gmm_scores = calculate_silhouette_scores('gmm', X, n_clusters_range)

# Store scores in a dictionary for plotting
scores_dict = {
    'kmeans': np.array(kmeans_scores),
    'gmm': np.array(gmm_scores)
}
```

شکل زیر نتایج معیار Silhouette به ازای مدل‌های مختلف را نشان می‌دهد. در صورت استفاده از تابع PCA امتیاز مربوط به خوشه‌بندی نیز افزایش یافته است. مقدار بهینه خوشه‌ها با استفاده از روش‌های کاهش بعد برابر ۲ خوشه است و به ازای استفاده از تمامی ویژگی‌ها بدون کاهش بعد برای مدل kmeans++ برابر ۵۵ و برای مدل GMM برابر ۵۴ است.



با توجه به اینکه، امتیاز خوشه‌بندی با استفاده از ترکیب ویژگی‌های MFCC و Spectral Contrast در مقایسه با سایر حالت‌ها، دارای امتیاز Silhouette بیشتری می‌باشد؛ در ادامه، خوشه‌بندی‌ها با استفاده از ترکیب این ویژگی‌ها انجام شده است. با توجه به بالا بودن امتیاز خوشه‌بندی با دو خوشه با استفاده بدون استفاده از PCA نیز بالا است، در ادامه خوشه‌بندی با دوخوشه، تعداد خوشه‌های بهینه(۵۵ برای kmeans و ۵۴ برای GMM) و نیز ۳۰ خوشه انجام شده‌است.

۵_۳ ارزیابی خوشه‌بندی

پایه و اساس خوشه‌بندی، شباهت بین نمونه‌ها است و نمونه‌هایی که ویژگی‌های صوتی آن‌ها شبیه به هم باشند؛ در یک خوشه قرار می‌گیرند. هدف از این بخش، ارزیابی دقیق‌تر نتایج خوشه‌بندی با بررسی ترکیب هر خوشه است. به طور خاص، هدف ما درک توزیع دانشجویان، بررسی جنسیت نمونه‌های صوتی قرار گرفته در یک خوشه و استدلال پشت قرار گرفتن نمونه‌های خاص با هم درون یک خوشه است. برای این منظور تابع Cluster_Analysis نوشته شده که با دریافت فریم داده‌ها که شامل برچسب مربوط به خوشه‌ی داده‌ها می‌باشد؛ اطلاعات زیر را به ازای هر خوشه در خروجی میدهد:

- تعداد دانشجویان(با توجه به اینکه چندین نمونه صوتی از هر فرد در دیتاست وجود دارد باید تعداد متمایز دانشجویان موجود در کلاستر را بدست آوریم.)
- تعداد دانشجویان مونث

- تعداد دانشجویان مذکر
- تعداد کل نمونه‌های مونث در خوشه
- تعداد کل نمونه‌های مذکر در خوشه

شکل زیر، کد مربوط به این تابع را نشان می‌دهد:

```
def Cluster_Analysis(df, cluster_label_column):

  cluster_analysis = []
  Num_Cluster = df[cluster_label_column].nunique()

  for cluster in range(Num_Cluster):
      cluster_data = df[df[cluster_label_column] == cluster]
      unique_students = cluster_data['student_id'].nunique()
      male_students = cluster_data[cluster_data['label']==
'male']['student_id'].nunique()
      female_students = cluster_data[cluster_data['label'] ==
'female']['student_id'].nunique()
      cluster_analysis.append({
          'Cluster': cluster,
          'Unique_Students': unique_students,
          'Male_Students': male_students,
          'Female_Students': female_students,
          'male_Samples': cluster_data['label'].value_counts().get('male', 0),
          'Female_Samples': cluster_data['label'].value_counts().get('female', 0)
      })

  cluster_analysis_df = pd.DataFrame(cluster_analysis)
  return cluster_analysis_df
```

۵_۳_۱ خوشه‌بندی به ازای ۲ خوشه

با توجه به اینکه امتیاز خوشه‌بندی (Silhouette Score) با استفاده از ترکیب ویژگی‌های MFCC و Spectral Contrast در مقایسه با سایر روش‌ها بالاتر است، خوشه‌بندی‌ها در این بخش، با استفاده از این ویژگی‌ها انجام شده‌اند. در این بخش، خوشه‌بندی با دو خوشه (تعداد بهینه تعیین شده در صورت استفاده از PCA) انجام شده است.

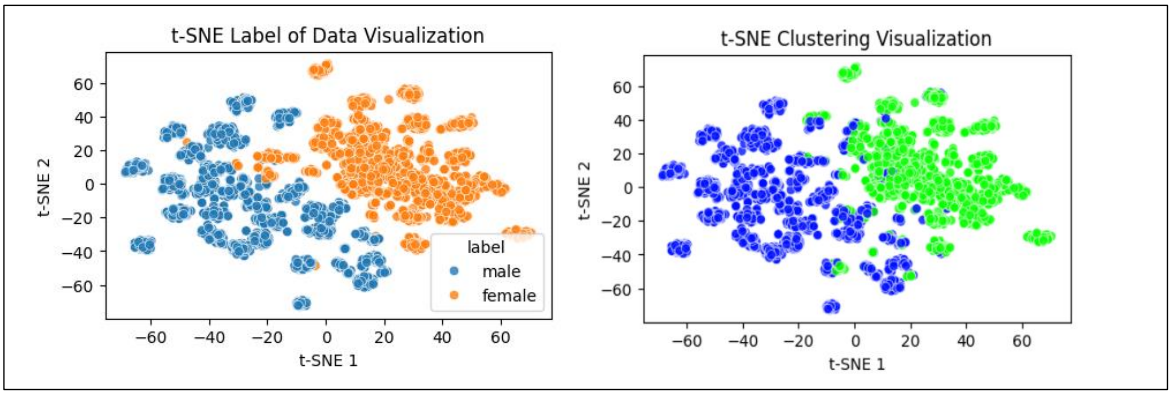
با تحلیل نتایج خوشه‌بندی با استفاده از تابع Cluster_Analysis مشاهده می‌شود که تعداد نمونه‌های female در یک خوشه بسیار بیشتر از تعداد نمونه‌های male است. به این ترتیب، خوشه‌بندی با مدل‌های Kmeans++ و GMM به نوعی معادل طبقه‌بندی بر اساس جنسیت است. در واقع، با استفاده از PCA، ویژگی‌های جدیدی از ترکیب ویژگی‌ها تولید می‌شوند که در راستای بیشترین پراکندگی داده‌ها هستند. از طرفی، باید توجه داشت که ویژگی‌های صوتی مربوط به نمونه‌های مونث متفاوت از ویژگی‌های صوتی نمونه‌های مذکر هستند و به نوعی می‌توان گفت که روش PCA ویژگی‌های جدید در جهت بیشتر کردن این تفاوت‌ها عمل کرده است و در صورت خوشه‌بندی با دو خوشه، نمونه‌های مونث و مذکر در خوشه‌های جداگانه قرار می‌گیرند. شکل زیر خروجی مربوط به مدل Kmeans++ را نشان می‌دهد.

```
kmeans = KMeans(n_clusters=optimal_clusters['kmeans_PCA'], init='k-means++', random_state=42)
balanced_data['Cluster_Kmeans_2'] = kmeans.fit_predict(X)
```

```
df_Kmeans_2 = Cluster_Analysis(balanced_data, 'Cluster_Kmeans_2')
df_Kmeans_2.head()
```

	Cluster	Unique_Students	Male_Students	Female_Students	male_Samples	Female_Samples
0	0	35	7	28	105	1853
1	1	43	28	15	1967	219

علاوه بر این، نمونه‌ها یکبار بر اساس برچسب جنسیت نمونه‌ها و یکبار نیز بر اساس برچسب خوشه‌بندی (با استفاده از TSNE در فضای دو بعدی) رسم شده‌اند. همانطور که از شکل‌ها مشخص است، خوشه‌بندی انجام شده بسیار شبیه به برچسب جنسیت داده‌ها می‌باشد.



با استفاده از مدل GMM نیز خوشه‌بندی مشابه با روش Kmeans++ منجر به طبقه‌بندی جنسیت شده است.

```
gmm = GaussianMixture(n_components=optimal_clusters['gmm_PCA'], random_state=42)
balanced_data['Cluster_gmm_2'] = gmm.fit_predict(X)
```

```
df_gmm_2 = Cluster_Analysis(balanced_data, 'Cluster_gmm_2')
df_gmm_2.head()
```

	Cluster	Unique_Students	Male_Students	Female_Students	male_Samples	Female_Samples
0	0	33	5	28	27	2006
1	1	33	28	5	2045	66

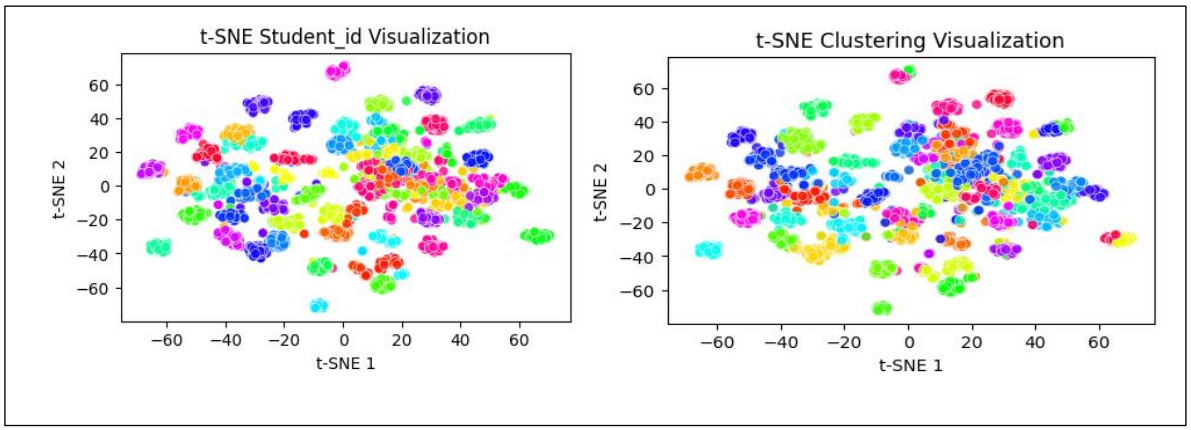
۵_۳_۲ خوشه‌بندی به ازای تعداد خوشه بهینه در Kmeans و GMM

مقدار بهینه خوشه‌ها با استفاده از روش‌های کاهش بعد برابر ۲ خوشه شد و به ازای استفاده از تمامی ویژگی‌ها بدون کاهش بعد برای مدل kmeans++ برابر ۵۵ و برای مدل GMM برابر ۵۴ است که نزدیک به تعداد دانشجویان موجود در دیتاست (۵۶) می‌باشد. به عبارت دیگر در صورتی که کاهش بعد انجام ندهیم و با استفاده از تعداد زیادی از ویژگی‌ها بخواهیم خوشه‌بندی را انجام دهیم، تقریبا بسیاری از نمونه‌های صوتی هر فرد در خوشه جداگانه قرار می‌گیرد. با تحلیل نتایج خوشه‌بندی با استفاده از تابع Cluster_Analysis نیز همین امر مشاهده می‌شود. در شکل زیر اطلاعات ۵ خوشه از ۵۵ خوشه مدل Kmeans آورده شده است. در بسیاری از خوشه‌ها، نمونه‌ها به یک فرد و یا به یک جنسیت متعلق هستند.

```
df_Kmeans_opt = Cluster_Analysis(balanced_data, 'Cluster_Kmeans_optimum')
df_Kmeans_opt.head()
```

	Cluster	Unique_Students	Male_Students	Female_Students	male_Samples	Female_Samples
0	0	6	6	0	83	0
1	1	8	0	8	0	94
2	2	9	9	0	101	0
3	3	4	1	3	60	7
4	4	5	4	1	76	1

علاوه بر این، نمونه‌ها یکبار بر اساس برچسب شماره دانشجویی نمونه‌ها و یکبار نیز بر اساس برچسب خوشه‌بندی انجام‌شده، با تعداد خوشه بهینه در مدل Kmeans++ (با استفاده از TSNE در فضای دو بعدی) رسم شده‌اند. همانطور که از شکل‌ها مشخص است، خوشه‌بندی انجام شده بسیار شبیه به برچسب شماره دانشجویی داده‌ها می‌باشد.



شکل زیر نیز خروجی مربوط به مدل GMM را نشان می‌دهد. در این مدل نیز بسیاری از خوشه‌ها تنها شامل نمونه‌های یک دانشجو یا یک جنسیت هستند.

```
df_gmm_opt = Cluster_Analysis(balanced_data, 'Cluster_gmm_opt')
df_gmm_opt.head()
```

	Cluster	Unique_Students	Male_Students	Female_Students	male_Samples	Female_Samples
0	0	2	2	0	93	0
1	1	4	0	4	0	57
2	2	4	3	1	76	6
3	3	3	3	0	37	0
4	4	5	5	0	134	0

۵_۳_۳ خوشه بندی با ۳۰ خوشه

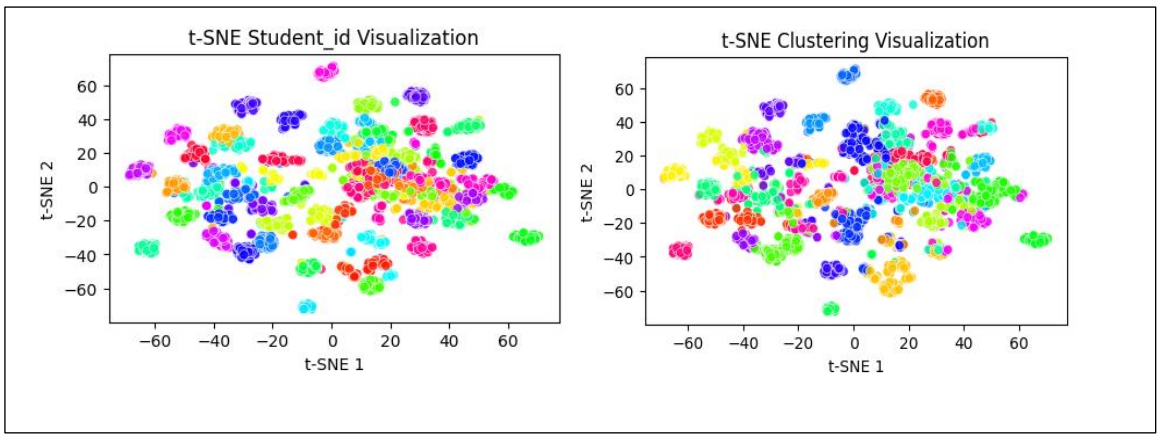
با خوشه‌بندی به ازای ۳۰ خوشه، نمونه‌های مشابه به هم در یک خوشه قرار می‌گیرند. با بررسی خوشه‌ها در این حالت، مشاهده می‌شود که بیشتر خوشه‌ها شامل نمونه‌های مربوط به یک جنسیت خاص هستند. اما

تعداد نمونه‌ها در هر خوشه بیشتر از خوشه‌بندی با ۵۵ خوشه در روش KMeans++ (و بیشتر از خوشه‌بندی با ۵۴ خوشه در روش GMM) است.

```
df_Kmeans_30 = Cluster_Analysis(balanced_data, 'Cluster_Kmeans_30')
df_Kmeans_30.head()
```

	Cluster	Unique_Students	Male_Students	Female_Students	male_Samples	Female_Samples
0	0	10	9	1	161	2
1	1	8	0	8	0	84
2	2	10	7	3	87	9
3	3	8	6	2	207	18
4	4	14	12	2	147	27

برای خوشه‌بندی با مدل Kmeans++ با ۳۰ خوشه نیز، نمونه‌ها یکبار بر اساس برچسب شماره دانشجویی نمونه‌ها و یکبار نیز بر اساس برچسب خوشه‌بندی (با استفاده از TSNE در فضای دو بعدی) رسم شده‌اند. همانطور که از شکل‌ها مشخص است، خوشه‌بندی انجام شده بسیار شبیه به برچسب جنسیت داده‌ها می‌باشد و شبیه به حالت قبل است با این تفاوت که تعداد خوشه‌ها کمتر و حجم آن‌ها بزرگتر از خوشه‌بندی با ۵۵ خوشه است.



شکل زیر نیز اطلاعات ۵ خوشه از ۳۰ خوشه با استفاده از مدل GMM را نشان میدهد.

```
df_gmm_30 = Cluster_Analysis(balanced_data, 'Cluster_gmm_30')
df_gmm_30.head()
```

	Cluster	Unique_Students	Male_Students	Female_Students	male_Samples	Female_Samples
0	0	3	3	0	146	0
1	1	1	0	1	0	74
2	2	4	2	2	90	9
3	3	6	5	1	213	1
4	4	8	7	1	130	28

۵_۴ نتیجه‌گیری

به نظر می‌رسد خوشه‌بندی با ویژگی‌های MFCC و Spectral Contras به گونه‌ای است که در صورت استفاده از دو خوشه، خوشه‌بندی منجر به طبقه‌بندی جنسیت می‌شود. هر چه تعداد خوشه‌ها افزایش یابد، هر یک از این دو خوشه به خوشه‌های جداگانه‌ای تقسیم می‌شوند. اگر تعداد خوشه‌ها نزدیک به تعداد نفرات باشد (در حالت بهینه بر اساس امتیاز Silhoutte به این تعداد خوشه رسیدیم)، به حالتی می‌رسیم که بیشتر نمونه‌های مربوط به هر شخص در یک خوشه قرار می‌گیرند و در نهایت، در حالت حدی، هر نمونه در یک خوشه قرار می‌گیرد. از طرفی، با افزایش تعداد خوشه‌ها از حالت بهینه به دلیل کمتر شدن فاصله بین خوشه‌ها، امتیاز Silhouette نیز کاهش میابد و نیز در صورتی که تعداد خوشه‌ها، کمتر از حالت بهینه باشد؛ به دلیل بیشتر شدن فاصله درونی در خوشه‌ها، امتیاز Silhouette کاهش می‌یابد.

در خوشه‌بندی با دو خوشه، خوشه‌بندی شبیه به طبقه‌بندی جنسیت شد و تمایزی بین نمونه‌های صوتی افراد مختلف ایجاد نشده بود. از طرفی نیز، استفاده از PCA برای تولید ویژگی‌های جدید، تفاوت‌های جنسیتی در داده‌ها را برجسته‌تر کرد و منجر به افزایش امتیاز خوشه‌بندی با دو خوشه، با استفاده از ویژگی‌های کاهش‌یافته، شد. این نتایج نشان می‌دهند که انتخاب ویژگی‌ها و تکنیک‌های کاهش ابعاد می‌تواند تأثیر قابل‌توجهی بر مقدار معیار خوشه‌بندی داشته باشد.

۶ مراجع

[1] N. Chandolikar, C. Joshi, P. Roy, A. Gawas, and M. Vishwakarma, “Voice recognition: A comprehensive survey,” in *2022 International Mobile and Embedded Technology Conference (MECON)*, 2022.

[2] N. H. Tandel, H. B. Prajapati, and V. K. Dabhi, “Voice recognition and voice comparison using machine learning techniques: A survey,” in *2020 6th International Conference on Advanced Computing and Communication Systems (ICACCS)*, 2020.

[3] B. Yelure, S. Patil, A. Nayakwadi, C. Raut, K. Joshi, and A. Nadaf, “Machine Learning based Voice Authentication and Identification,” in *2023 3rd International Conference on Innovative Mechanisms for Industry Applications (ICIMIA)*, 2023, pp. 936–940.

[4] “Feature extraction — librosa 0.10.2.post1 documentation,” *Librosa.org*. [Online]. Available: https://librosa.org/doc/main/feature.html. [Accessed: 30-Dec-2024].

[5] *Kaggle.com*. [Online]. Available: https://www.kaggle.com/code/gopidurgaprasad/mfcc-feature-extraction-from-audio. [Accessed: 30-Dec-2024].

[6] M. S. Ahmad, “Deep learning 101: Lesson 23: The basics of audio signal processing with FFT,” *Medium*, 02-Sep-2024. [Online]. Available: https://muneebsa.medium.com/deep-learning-101-lesson-23-the-basics-of-audio-signal-processing-with-fft-ffef65689c1d. [Accessed: 30-Dec-2024].

[7] B. A. Alsaify, H. S. Abu Arja, B. Y. Maayah, M. M. Al-Taweel, R. Alazrai, and M. I. Daoud, “Voice-Based Human Identification using Machine Learning,” in *2022 13th International Conference on Information and Communication Systems (ICICS)*, 2022.

[8] R. Sharma, D. Govind, J. Mishra, A. K. Dubey, K. T. Deepak, and S. R. M. Prasanna, “Milestones in speaker recognition,” *Artif. Intell. Rev.*, vol. 57, no. 3, 2024.

[9] K. W. Cheuk, H. Anderson, K. Agres, and D. Herremans, “NnAudio: An on-the-fly GPU audio to spectrogram conversion toolbox using 1D convolutional neural networks,” *IEEE Access*, vol. 8, pp. 161981–162003, 2020.

[10] C. Li *et al.*, “Deep Speaker: An end-to-end neural speaker embedding system,” *arXiv [cs.CL]*, 2017.

[11] V. K. Pande, V. K. Kale, and S. Tharewal, “Audio data feature extraction for speaker diarization,” in *Proceedings of the NIELIT’s International Conference on Communication, Electronics and Digital Technology*, Singapore: Springer Nature Singapore, 2024, pp. 243–255.

[12] W. N. Jasim, S. A. W. Saddam, and E. J. Harfash, “Wind sounds classification using different audio feature extraction techniques,” *Informatica (Ljubl.),* vol. 45, no. 7, 2022.

[13] S. Vijayputra, *Gender Recognition Using Fast Fourier Transform With Ann*. 2019.

[14] S. Furui, “Speaker Recognition in Smart Environments,” in *Human-Centric Interfaces for Ambient Intelligence*, Elsevier, 2010, pp. 163–184.