به نام خدا

گزارش پروژه نهایی درس یادگیری ماشین

آتیه آرمین (۸۱۰۱۹۷۶۴۸)

معین شیردل (۸۱۰۱۹۷۵۳۵)

بهمن‌ماه ۱۴۰۰

بخش ۱) شناسایی داده ها

بخش ۱ - ۱) Extract کردن داده ها

در این بخش ابتدا باید folder داده ها را extract کرده تا بتوانیم از تمام داده ها استفاده کنیم. پس از این کار با کمک کتابخانه os به فایل داده ها می توانیم دست یابیم. در این بخش نیاز به load کردن همه داده ها نخواهیم داشت؛ به همین دلیل تنها اسم فایل ها را در کنار آدرس فولدری که در آن قرار دارند را در یک لیست نگهداری میکنیم. همچنین در لیستی جداگانه label هر داده را با همان ترتیبی که داده ها را وارد لیست اول کرده‌ایم، نگهداری میکنیم. در ادامه پروژه با فایل هایی برخورد کردیم که به اصطلاح damaged بودند و خوانده نمی‌شدند. برای آنکه در ادامه این داده ها مشکل‌ساز نشوند، با کتابخانه‌ی mutagen و تابع MP3 که فایل های mp3 را با سرعت خیلی بالا میتواند بخواند، تمام فایل ها را میخوانیم و با یک try-except می توانیم فایل هایی که damaged هستند را شناسایی کرده و آن ها را وارد لیست داده ها نکنیم. همانطور که در notebook مشخص است،فایل های ۸۹ و ۱۵۶ آهنگ های لری، ۲۲ بندری، ۲۳۱ و ۱۰۴ ترکی همگی damaged هستند که آنها را وارد لیست داده ها نمی کنیم.

برای آنکه با داده هایمان بیشتر آشنا شویم، تعداد نمونه ها از هر ژانر نیز چاپ شده است. همانطور که مشخص است تعداد نمونه ها تقریبا در یک range هستند و دیتاست ما تقریبا balanced است و لازم نیست نگران این موضوع باشیم.

بخش ۱ - ۲) Visualization

در این بخش به visualization یک داده از هر ژانر می پردازیم. برای این کار از ۳ نمودار ،Waveform Spectrogram و Spectrum استفاده میکنیم.

لازم به ذکر است که برای تصویرسازی داده ها ابتدا باید هر نمونه را توسط تابع load از کتابخانه librosa بخوانیم. این تابع برای داده هایی با فرمت wav به نسبت سریع و خوب عمل میکند ولی برای داده هایی با فرمت

mp3 بسیار کند است. به همین دلیل در این بخش تمام داده ها را load نکرده و تنها بخشی از آن ها که نیاز داریم (یک نمونه از هر ژانر) load میکنیم. تابع load در واقع با خواندن یک فایل mp3، دو مقدار را به عنوان خروجی میدهد. در ادامه در بخش خواندن داده ها این تابع بیشتر توضیح داده شده است اما در اینجا به آرایه‌ی y که خروجی این تابع است و در واقع سری زمانی آن آهنگ را نشان می‌دهد، احتیاج داریم (توضیح این مقدار در بخش خواندن داده ها با جزئیات بیشتر ذکر شده است).

- نمودار Waveform:

این نمودار در واقع سری زمانی ذکر شده که در بازه های زمانی مشخص sr نمونه برداری شده است را به تصویر می‌کشد و شکل موج آهنگ را نشان می‌دهد.

برای کشیدن این نمودار از تابع display.waveshow() از کتابخانه librosa استفاده شده است.

- نمودار Spectrogram:

این نمودار نمایش بصری طیف فرکانس های صدا یا سیگنال های دیگر است و با زمان تغییر می‌کند. در این نمودار محور عمودی فرکانس (بر حسب Hz) و محور افقی زمان است. همچنین در این نمودار از رنگ های مختلف برای نشان دادن دامنه هر فرکانس استفاده شده است به طوری که هر چه رنگ روشن‌تر باشد، انرژی سیگنال بیشتر است.

Spectrogram از سیگنال های صوتی که با استفاده از تبدیل فوریه تولید می‌شود. به طوری که تبدیل فوریه سیگنال را به فرکانس های تشکیل دهنده آن تجزیه کرده و دامنه هر فرکانس را نمایش می‌دهد. به عبارت دیگر مدت زمان یک سیگنال را به بخش های زمانی کوچک‌تر تقسیم می‌کند و سپس روی هر کدام از این بخش ها، تبدیل فوریه را اعمال می‌کند تا فرکانس های موجود در آن بخش مشخص شود و در نهایت تمام آن تبدیل ها را ترکیب می‌کند.

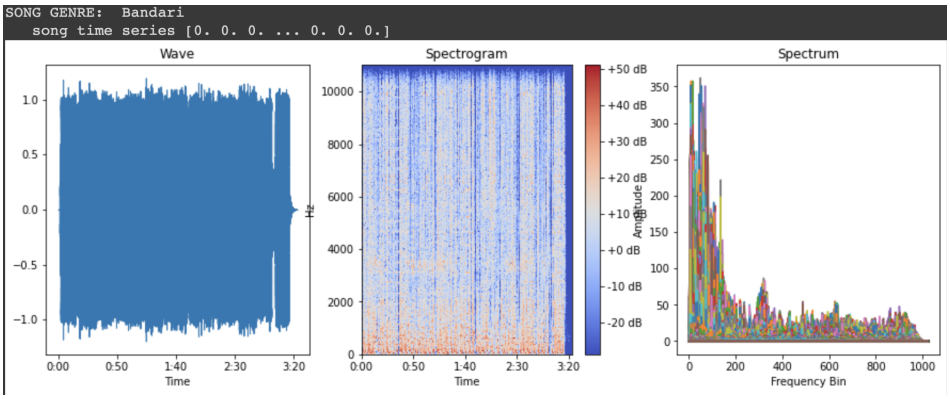
برای کشیدن این نمودار ابتدا توسط تابع stft() از کتابخانه librosa، تبدیل فوریه y را بدست آورده و سپس با تابع amplitude_to_db()، مقادیر دسیبل را به دست آورده که نشان دهنده انرژی سیگنال است و در نهایت با تابع display.specshow() این نمودار را نمایش می‌دهیم.

- نمودار Spectrum :

همانطور که در بخش قبل ذکر شده است، تبدیل فوریه به ما این امکان را می‌دهد که یک سیگنال را به فرکانس ها و دامنه آن ها تجزیه کنیم. به عبارت دیگر این تبدیل سیگنال را از حوزه زمان به حوزه فرکانس تبدیل میکند که نمودار حاصل از آن را Spectrum می‌نامند.

برای کشیدن این نمودار کافیست قدر مطلق مقادیر به دست آمده از تابع stft() که تبدیل فوریه y را میدهد را بدست آورده و رسم کنیم.

حال در این بخش این نمودار ها ممکن است اطلاعات زیادی درباره تفاوت ژانر ها در اختیار ما قرار ندهند زیرا ژانر ها بسیار کلی هستند و ممکن است بسیاری از آهنگ ها از ژانر های متفاوت Spectrogram و Spectrum شبیه به هم داشته باشند. این visualization بیشتر به ما کمک خواهد کرد تا ویژگی هایی که در ادامه از سری زمانی به‌دست آمده استخراج می‌کنیم را بهتر بفهمیم. به طور کلی از این نمودار ها میتوان اطلاعات مختلفی را درباره یک آهنگ بدست آورد اما درباره تفاوت بین ژانر ها اطلاعات زیادی نخواهیم یافت مگر آنکه چندین نمونه مختلف را از هر ژانر نمایش دهیم که در اینجا هدف ما تنها نمایش چند نمونه از آهنگ‌ها و فهم بهتر ویژگی های استخراج شده بوده است. نمونه ای از این ۳ نمودار در ادامه قابل مشاهده است.



بخش ۲) خواندن داده ها

حال نیاز به خواندن تمام داده ها داریم. در این بخش توسط تابع load از کتابخانه librosa هر آهنگ را می‌خوانیم. این تابع آدرس فایل را به عنوان ورودی گرفته و دو مقدار y و sr را به عنوان خروجی باز می‌گرداند. Y در واقع سری زمانی آهنگ است. سری زمانی را می توان اینگونه تعریف کرد که به یک دنباله یا توالی از متغیرهای تصادفی که در فاصله های زمانی ثابت نمونه برداری شده‌اند می‌گویند. به بیانی دیگر، منظور از یک سری زمانی، مجموعه‌ای از داده های آماری است که در فواصل زمانی مساوی و منظمی جمع‌آوری شده باشند.

به طور کلی می‌توان گفت که داده‌های صوتی با نمونه‌برداری از موج صوتی در فواصل زمانی معین و اندازه‌گیری شدت یا دامنه موج در هر نمونه به دست می‌آید. این نمونه ها همان y هستند که در خروجی تابع load() میبینیم. همچنین sr، تعداد نمونه در هر ثانیه یا به عبارت دیگر sample rate نمونه ها (y) را به ما می‌دهد. در واقع زمانی که یک آهنگ یا صدا را به صورت یک فایل در اختیار داریم، آن فایل یک فرمت فشرده است. هنگامی که این فایل load میشود، از حالت فشرده خارج شده و تبدیل به یک آرایه (numpy array) می‌شود. لازم به ذکر است که این آرایه به فرمت فایل ارتباطی ندارد و در همه حالات یکسان است.

در ادامه برای آنکه ویژگی های بدست آمده دقیق تر باشند، بهتر است کل آهنگ را به قسمت های ۱ دقیقه‌ای تقسیم کنیم و هر کدام از این قسمت ها را به عنوان یک نمونه جدا در نظر گرفته و از آن ویژگی ها را استخراج کنیم. به طور حدودی هر ۱۳۰۰۰۰۰ نمونه نشان دهنده یک دقیقه خواهد بود.

پس از جدا کردن هر آهنگ به بخش های ۱ دقیقه‌ای آن بخش را به کلاس feature extraction میدهیم تا ویژگی های مورد نظر را استخراج کند. ویژگی‌های استخراج شده در بخش۳ - ۱ به صورت کامل توضیح داده شده‌اند.

به علت زیاد بودن داده‌ها و برای آنکه ram مان crash نکند، مجبور شدیم تا آهنگ‌های هر ژانر را به صورت جداگانه load کنیم، ویژگی‌ها را استخراج کرده و تمام اطلاعات به‌دست آمده را در یک فایل csv مجزا ذخیره کنیم. برای اینکار ابتدا توسط تابع create_file، فایل csv مربوط به آن کلاس با ستون‌های مد نظر ساخته‌شده و سپس در کلاس feature extraction پس از استخراج ویژگی‌ها، ویژگی های هر بخش ۱ دقیقه‌ای به صورت یک ردیف در این فایل نوشته می‌شوند. این روند برای هر ژانر به صورت جداگانه انجام شده‌است. پس از بدست آمدن هر ۵ دیتاست، همه آن ها را به کمک کتابخانه pandas خوانده و سپس به کمک تابع sample() آن ها را shuffle کرده‌ایم تا نمونه‌هایی از یک ژانر همگی پشت سر هم نباشند. در نهایت این فایل که شامل تمام اطلاعات است را ذخیره کرده تا در ادامه پروژه تمام داده‌ها را تنها از این فایل بخوانیم.

بخش ۳) پیش‌پردازش

بخش ۳ - ۱) استخراج ویژگی

به طور کلی، نیازمندیم که قطعه‌های آهنگ‌های مورد بررسی در پروژه را با اعداد توصیف کنیم. برای داده‌های از نوع صدا و صوت، ویژگی‌های زیادی قابل استخراج هستند اما 5 مورد از آن‌ها شاخص ترین‌شان هستند که برای توصیف داده‌های ما انتخاب شدند.

- ویژگی Zero Crossing:

این ویژگی، یکی از ساده‌ترین و در دسترس‌ترین ویژگی‌های داده‌هایی از جنس سیگنال یا Audio است و بیان‌گر تعداد دفعاتی است که یک سیگنال یا یک موج خاص، تغییر علامت می‌دهد. برای تشخیص و جداسازی یک سری از سبک‌ها و ژانرهای موسیقی، این ویژگی بسیار کاربردی است چون به طور مثال، مقادیر بالاتری در موسیقی‌های سبک راک و . . . دارد. همچنین به علت سادگی محاسبه، تبدیل به یک ویژگی پراستفاده شده‌است.

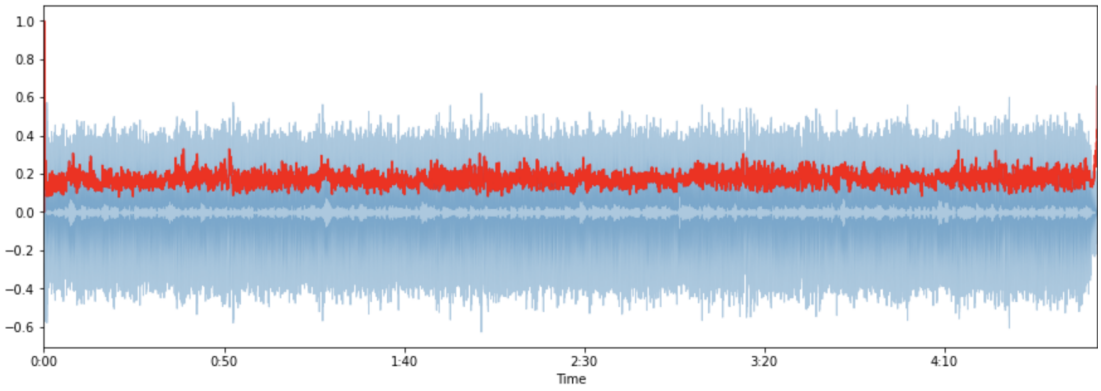
خروجی ویژگی Zero Crossing یک عدد به ازای هر فریم از قسمت‌های یک‌دقیقه‌ای آهنگ‌هاست. برای توصیف کل یک دقیقه‌ی آهنگ، تعداد Zero Crossing های کل فریم‌ها را جمع می‌زنیم و بدین صورت تکه آهنگ‌های یک دقیقه‌ای را از لحاظ Zero Crossing توصیف می‌کنیم.

- ویژگی Spectral Centroid:

این ویژگی به نوعی مرکز جرم یک صوت را از لحاظ فرکانسی پیدا می‌کند و بیان‌گر مرکز طیف فرکانسی آن است. به طور مثال، برای آهنگ‌های متال در مقایسه با آهنگ‌های کلاسیک و ملایم‌تر، مرکز طیف فرکانسی (Spectral Centroid) نزدیک‌تر به انتهای بازه (بزرگ‌تر) است چون در زمان‌های بیشتری از آهنگ، اصوات دارای مقادیر فرکانسی بالاتر هستند.

خروجی این ویژگی هم یک عدد به ازای هر فریم است اما جمع زدن مقادیر Spectral Centroid متعلق به هر فریم به نظر کار معناداری نیست. برای همین، فرض می‌کنیم که این اعداد برای فریم‌های مختلف از توزیع نرمال پیروی می‌کنند. پس برای توصیف Spectral Centroid های هر یک از بخش‌های یک دقیقه‌ای، از اعداد حاصل از فریم‌هایش یک میانگین و واریانس نگه‌داری می‌کنیم تا بیان‌گر توزیع نرمال Spectral Centroid های تکه‌های یک‌دقیقه‌ای آهنگ‌های هر ژانر باشد.

مثالی از visualization این ویژگی در ادامه قابل مشاهده است.

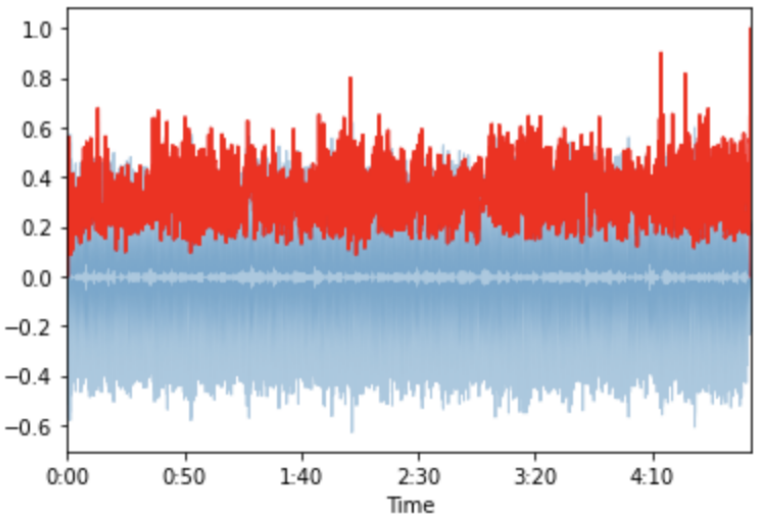


- ویژگی Spectral Roll-off:

این ویژگی، معیاری از شکل نمودار آن سیگنال یا صوت است. به نوعی نشان‌دهنده‌ی میزان فرکانسی است که در بازه‌ی آن میزان از فرکانس و کمتر از آن، بخش زیادی از انرژی صوت پوشش داده می‌شود. (به طور مثال، عددی بین ۸۰ تا ۹۰ درصد از انرژی آن). این ویژگی برای هر فریم از موج ما محاسبه می‌شود.

خروجی این ویژگی نیز همانند ویژگی Spectral Centroid است. یعنی یک عدد برای هر فریم که جمع‌زدن آن‌ها خیلی معنادار نیست. برای این ویژگی نیز، فرض می‌کنیم که مقادیر خروجی برای هر قسمت یک‌دقیقه‌ای از توزیع نرمال پیروی می‌کنند و مشابه با Spectral Centroid، یک میانگین و واریانس برای توصیف آن نگه می‌داریم.

مثالی از visualization این ویژگی در ادامه قابل مشاهده است.

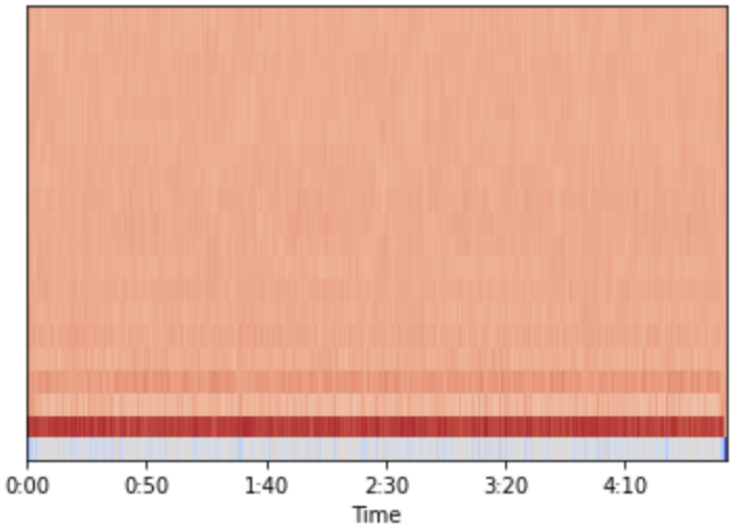


- ویژگی Mel-Frequency Cepstral Coefficients یا MFCC ها:

این ویژگی‌ها، مجموعه‌ای از ۲۰ تا ضریب است که سعی می‌کنند طیف فرکانسی صدا یا موج ما را از جنبه‌های مختلف توصیف کنند. این ضرایب، اطلاعاتی در مورد تغییرات موج در هر یک از قسمت‌های طیف فرکانسی به ما می‌دهند و در مجموع می‌توان اطلاعاتی در مورد قدرت و انرژی آن موج از این ضرایب استخراج کرد. اگر یک ضریب MFCC دارای مقدار منفی باشد، بیانگر این است که عمده‌ی انرژی طیفی در قسمت‌های کم‌فرکانس جمع شده است و آن بخش از موج ما، به طور عمده فرکانس‌های پایین داشته است. به طور مشابه، اگر مقدار آن ضریب مثبت باشد نیز بیانگر این است که عمده‌ی انرژی طیفی از قسمت‌های پرفرکانس آن موج حاصل می‌شود.

برای هر یک از ۲۰ ویژگی حاصل از ضرایب MFCC نیز مانند دو ویژگی بالا عمل می‌کنیم. یعنی فرض می‌کنیم که مقدار هر ضریب در فریم‌های موجود در یک قسمت یک‌دقیقه‌ای از آهنگ از توزیع نرمال پیروی می‌کنند. یعنی در مجموع ۲۰ میانگین و ۲۰ واریانس برای هر قسمت یک‌دقیقه‌ای ذخیره می‌کنیم که حاصل آن، ۴۰ ستون متفاوت است.

مثالی از visualization این ویژگی در ادامه قابل مشاهده است.

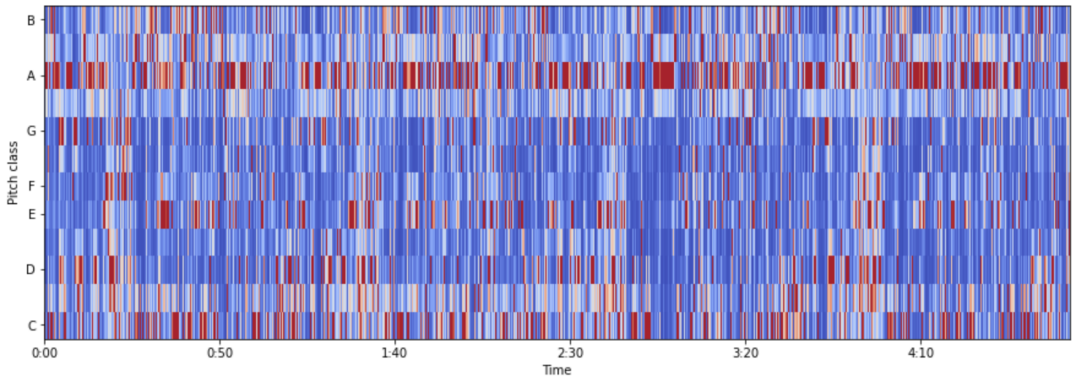


- ویژگی Chroma Frequencies:

در ویژگی‌های Chroma Frequencies، هر قطعه آهنگ روی ۱۲ تا bin مختلف که نماینده‌ی ۱۲ تا زیر صدا از اکتاو های موسیقی هستند تصویر می‌شوند. تشخیص تفاوت این اکتاوهای نزدیک به هم برای گوش انسان به راحتی صورت نمی‌گیرد. ویژگی‌های مربوط به این زیر صدا (chroma) ها، ۱۲ ویژگی هستند که خواص مربوط به هارمونی‌های تکرار شونده و ملودی آهنگ‌ها را نشان می‌دهند.

برای استخراج این ویژگی‌ها نیز مانند MFCC ها عمل می‌کنیم و برای هر یک از این ۱۲ مقدار حاصل از این ویژگی، میانگین و واریانس را برای تمامی فریم‌های موجود در قطعه آهنگ یک‌دقیقه‌ای حساب می‌کنیم و برای آن قطعه آهنگ، این ۲۴ مقدار را ذخیره می‌کنیم.

مثالی از visualization این ویژگی در ادامه قابل مشاهده است.



پس در مجموع برای توصیف هر قطعه‌ی یک‌دقیقه‌ای، ۶۹ تا ویژگی انتخاب شد که 1 ستون برای ویژگی اول، ۲ ستون برای هر یک از ویژگی های دوم و سوم، ۴۰ ستون برای ویژگی چهارم و ۲۴ ستون برای ویژگی پنجم اختصاص داده شده‌است.

بخش ۳ - ۲) Standardization و Label Encoding

مقیاس مقادیر در همه‌ی ویژگی‌ها یکسان نیست. مثلا مقادیر ویژگی zero crossing در حدود ۹۰۰۰۰ تا ۱۷۰۰۰۰ است در حالی که مقادیر ویژگی Mfcc_mean1 مثلا در حدود ۴۰ تا ۱۳۰ است. این تفاوت می‌تواند در مدل‌های طبقه بند ما تاثیر بگذارد و ویژگی‌ای همچون zero crossing تاثیر بیشتری در طبقه بند بگذارد تا ویژگی‌ای مانند Mfcc_mean1. به همین دلیل نیاز است تا تمام مقادیر را scale کنیم. برای این کار دو تابع standardization و normalization پیاده سازی شده اند. Standardization در واقع داده ها را با فرض گوسی بودن آن‌ها، استاندارد می‌کند به این معنا که میانگین آن‌ها را ۰ و واریانس آن‌ها را به ۱ تغییر می‌دهد. اما Normalization داده‌ها را بر اساس مینیمم و ماکزیمم آن ها scale می‌کند به طوری که هر مقدار را از مینیمم آن ویژگی کم کرده و تقسیم بر ماکزیمم منهای مینیمم می‌کند. فرض ما بر این است که توزیع ویژگی‌ها به صورت گوسی است، به همین دلیل از standardization استفاده می‌کنیم (standardization و normalization از لحاظ عملکرد تست شده‌اند و مشخصا فرض گوسی بودن داده‌ها تا حدودی درست است و standardization نتیجه بهتری دارد).

در ادامه باید توجه داشته باشیم که ژانرهای ما به صورت یک سری label از نوع string هستند که برای استفاده از آن ها لازم است آن ها را به عدد تبدیل کنیم. در اینجا از کلاس LabelEncoder() از کتابخانه sklearn استفاده می‌کنیم تا label ها را به اعداد بین ۰ تا ۴ تبدیل کند.

بخش ۳ - ۳) انتخاب ویژگی

در این بخش از کلاس SelectKbest() از کتابخانه sklearn استفاده میکنیم. الگوریتم آن به این صورت است که بر اساس یک function که در اینجا f_classif انتخاب شده است، k تا بهترین feature را انتخاب کرده و آن ها را جدا میکند. F_classif حاصل پراکندگی بین میانگین داده ها تقسیم بر پراکندگی میان داده ها را که f_value در ANOVA نام دارد، می‌سنجد.

برای انتخاب k ابتدا این feature selection را در یک پایپ به همراه تک تک طبقه بند ها پیاده سازی کرده و grid search را روی آنها اجرا کرده‌ایم. در واقع این بخش در بخش طبقه بند ها و به همراه آن ها پیاده‌سازی شده است. در تمامی طبقه بند ها البته این مقدار برابر با ۶۰ بدست آمده است.

بخش ۴) مدل ها

بخش ۴ - ۱) طبقه بندها

برای طبقه بند ها از ۳ الگوریتم svm، knn و polynomial logistic regression استفاده شده است. دلیل انتخاب هر کدام از این مدل ها و شرح پیاده سازی انجام شده در ادامه آمده است. در هر کدام از این مدل ها با استفاده از gridsearch، مقدار مناسب هایپر پارامتر ها را پیدا کرده و با آن مقدار ها ادامه داده‌ایم. همچنین برای پیدا کردن مقدار مناسب k در selectKbest که برای انتخاب ویژگی ها هست نیز در هر بخش با استفاده از یک pipe همراه با مدل به gridsearch داده شده است تا مقدار مناسب برای k نیز بدست آید.

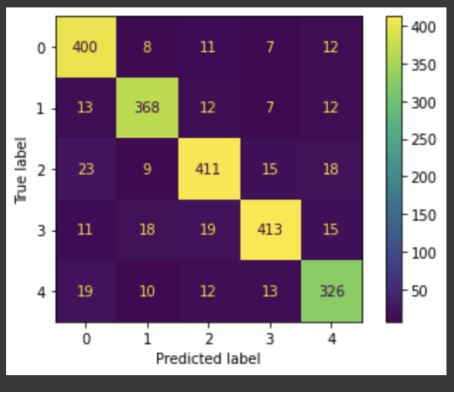
برای سنجش این مدل ها از تابع classification_report و confusion matrix استفاده شده است. تابع classification_report در واقع اطلاعاتی از قبیل recall، f1_score، accuracy و precision را به ما می دهد. همچنین با استفاده از confusion matrix نشان می توانیم به صورت بصری ببینیم که تعداد دیتا هایی که درست طبقه بندی شده اند و تعداد دیتا هایی که برچسب آن ها چیز دیگری بوده و برچسب پیشبینی شده آن ها چیز دیگری، چقدر است.

بخش ۴ - ۱ - ۱) طبقه بند knn

طبقه بند knn به صورت کلی برای اکثر داده ها به خوبی عمل میکند زیرا معیار طبقه بندی آن فاصله بین داده ها است و بر اساس k تا همسایه نزدیک به یک داده، برچسب آن داده را مشخص میکند. به همین سبب از این طبقه بند در این پروژه استفاده شده است.

در این بخش ابتدا با استفاده از gridsearch تعداد همسایه های مناسب برای تصمیم گیری و همچنین مقدار k مناسب برای انتخاب k تا بهترین ویژگی، به ترتیب ۱ و ۶۰ به دست آمده است. دقت حاصل از طبقه بند knn با این هایپر پارامتر ها روی داده های آموزش و تست به ترتیب ۱۰۰ و ۸۷ بوده و عملکرد آن روی داده های تست به صورت زیر است.

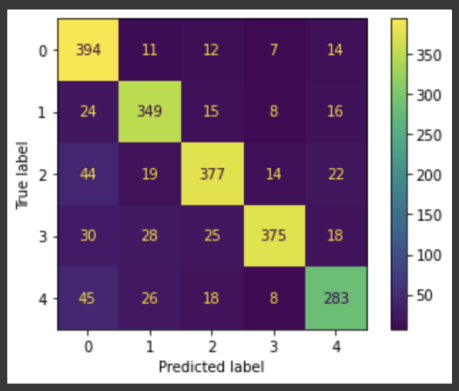
	precision	recall	f1-score	support
0	0.86	0.91	0.88	438
1	0.89	0.89	0.89	412
2	0.88	0.86	0.87	476
3	0.91	0.87	0.89	476
4	0.85	0.86	0.85	380
accuracy			0.88	2182
macro avg	0.88	0.88	0.88	2182
weighted avg	0.88	0.88	0.88	2182



همانطور که مشخص است، این نتایج به نسبت خوب هستند و با توجه به مقدار precision و recall مشخص است عملکرد این طبقه بند به نسبت خوب است. اما به صورت کلی می دانیم که طبقه بند knn با هایپر پارامتر k = ۱ در واقع باعث overfit می‌شود و باعث می شود تا طبقه بند به صورت کامل بر روی داده های آموزش fit

شود و عملکرد آن روی داده های تستای که به طور کلی کمی متفاوت با داده های آموزش باشند، کم شود. به همین دلیل این طبقه بند را با هایپر پارامتر k = ۳ نیز تست می کنیم. دقت آموزش و تست با این مدل به ترتیب ۹۲ و ۸۱ است. همانطور که مشخص است کمی از دقت تست نیز کاسته شده است اما در نهایت می توانیم بگوییم کمی از overfit شدن مدل جلوگیری کردهایم. عملکرد این طبقه بند در ادامه آمده است.

	precision	recall	f1-score	support
0	0.73	0.90	0.81	438
1	0.81	0.85	0.83	412
2	0.84	0.79	0.82	476
3	0.91	0.79	0.84	476
4	0.80	0.74	0.77	380
accuracy			0.81	2182
macro avg	0.82	0.81	0.81	2182
weighted avg	0.82	0.81	0.82	2182



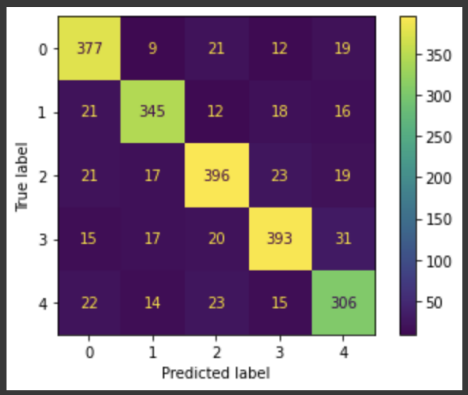
بخش ۴ - ۱ - ۲) طبقه بند SVM

طبقه بند svm طبقه بندی خطی است. می دانیم برای داده هایی که به صورت خطی جدا پذیر نیستند، با استفاده از kernel trick می توان داده ها را به ابعاد بالاتر برده تا جایی که بتوان آن ها را به صورت خطی جدا کرد. این طبقه بند داده ها را بسیار خوب و به نسبت سریع یاد میگیرد و این یک دلیل برای انتخاب svm در این پروژه است. همچنین با استفاده از kernel های غیر خطی، قطعا عملکرد این طبقه بند در این پروژه خوب خواهد بود.

در این بخش با استفاده از gridsearch مقدار مناسب برای C (پارامتر regularization)، کرنل مناسب برای این طبقه بند و همچنین مقدار k مناسب برای تعداد فیچر ها به ترتیب ۱۰، rbf و ۶۰ بدست آمده است.

در نهایت با داشتن این هایپر پارامتر ها دقت آموزش و تست به ترتیب ۹۸ و ۸۳ بوده و عملکرد این طبقه بند بر داده های تست به صورت زیر است. همانطور که در عکس مشخص است عملکرد این طبقه بند نیز بسیار خوب است. تنها کمی بر روی داده ها با برچسب ۴ مقدار precision و f1 score آن کم است. این به این معنا است که در این داده ها نسبت آن هایی که به درستی برچسب ۴ به آن ها نسبت داده شده است به تمام آن هایی که برچسب ۴ به آن ها نسبت داده شده است کمتر است و در نتیجه مقدار f1 score آن که به بیانی میانگین بین precision و recall است کمتر شده است. به نظر میرسد این طبقه بند در تشخیص داده ها با برچسب ۴ کمی به مشکل خرده و داده هایی که به اشتباه برچسب ۴ را به آن ها نسبت داده است زیاد تر از طبقه بند های دیگر است.

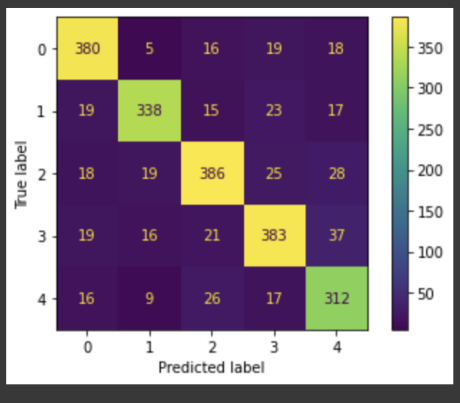
	precision	recall	f1-score	support
0	0.83	0.86	0.84	438
1	0.86	0.84	0.85	412
2	0.84	0.83	0.84	476
3	0.85	0.83	0.84	476
4	0.78	0.81	0.79	380
accuracy			0.83	2182
macro avg	0.83	0.83	0.83	2182
weighted avg	0.83	0.83	0.83	2182



بخش ۴ - ۱ - ۳) طبقه بند polynomial logistic regression

همانطور که مشخص است داده های ما به صورت خطی جدا پذیر نیستند. در نتیجه باید از روش هایی استفاده کنیم که بتواند مرز تصمیم غیر خطی را به خوبی ایجاد کند. یکی از این روش ها polynomial logistic regression است که مرز های تصمیم پیچیده را نسبتا خوب میتواند ایجاد کند. در اینجا ابتدا برای polynomial کردن ویژگی ها درجه ۳ را در نظر گرفته ایم و سپس با استفاده از greadsearch مقدار مناسب برای k در انتخاب k تا بهترین feature و مقدار مناسب برای C (عکس پارامتر regularization) به ترتیب ۶۰ و ۱ انتخاب شده اند. دقت حاصل از این طبقه بند بر روی داده های آموزش و تست به ترتیب ۱۰۰ و ۸۲ بدست آمده است. همچنین عملکرد این طبقه بند بر روی داده های تست نیز در ادامه قابل مشاهده است.

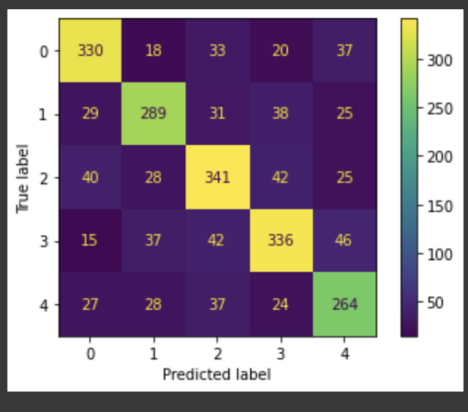
	precision	recall	f1-score	support
0	0.84	0.87	0.85	438
1	0.87	0.82	0.85	412
2	0.83	0.81	0.82	476
3	0.82	0.80	0.81	476
4	0.76	0.82	0.79	380
accuracy			0.82	2182
macro avg	0.82	0.82	0.82	2182
weighted avg	0.83	0.82	0.82	2182



در این جا به نظر میرسد این طبقه بند نیز با مشکل overfit رو به رو باشد. دقت آموزش آن ۱۰۰ شده و ممکن است عملکرد آن بر روی داده هایی از نوع دیگر خیلی خوب نباشد. به همین دلیل این طبقه بند را با ویژگی های

polynomial از درجه ۲ نیز آزمایش کرده‌ایم. در این حالت مقدار مناسب برای C ۰.۱ بدست آمد. طبقه بند جدید دقت آموزش ۹۶ و دقت تست ۷۱ میدهد. همچنین عملکرد آن بر روی داده های تست به صورت زیر است. همانطور که مشخص است با کمی کم شدن دقت آموزش شاید کمی از overfit جلوگیری کرده باشیم اما در مقابل دقت مان در داده های تست به صورت چشمگیری پایین آمده است. البته تشخیص overfitting تنها با این عوامل کار بسیار سختی است و نمی‌توان از آن مطمئن بود. به نظر میرسد در این جا با کم کردن درجه ویژگی های polynomial هزینه زیادی در پیشبینی داده های تست پرداخت میکنیم و این عمل خیلی جوابگو نخواهد بود.

	precision	recall	f1-score	support
0	0.75	0.75	0.75	438
1	0.72	0.70	0.71	412
2	0.70	0.72	0.71	476
3	0.73	0.71	0.72	476
4	0.66	0.69	0.68	380
accuracy			0.71	2182
macro avg	0.71	0.71	0.71	2182
weighted avg	0.72	0.71	0.72	2182



بخش ۴ - ۲) خوشه بندها

برای خوشه‌بندی داده‌ها از ۳ الگوریتم KMedoids، KMeans و Hierarchical با معیار فاصله‌ی Ward Linkage استفاده شد. تعداد ویژگی‌های داده‌ها ۶۹ تاست که بسیار زیاد است. به همین علت، برای راحت تر شدن خوشه‌بندی و همچنین قابلیت دیدن توزیع تصویری داده‌ها نیاز بود که ابعاد را کمتر کنیم. از آنجایی که

خوشه‌بندی مخصوص داده‌های بدون برچسب و روشی unsupervised است، برای Reduction از روش PCA استفاده شد و به کمک آن ابعاد داده را به ۲ بعد کاهش دادیم.

همچنین برای ارزیابی این مدل‌های خوشه‌بندی از دو معیار ارزیابی Silhouette Score و Purity استفاده شد که برای پیاده سازی اولی از توابع آماده استفاده شد ولی دومی به صورت دستی پیاده‌سازی شد. Purity به این‌گونه محاسبه شده که میزان دقت و خلوص هر خوشه، برابر با تعداد نمونه‌های کلاس غالب در آن خوشه تقسیم بر کل نمونه‌های موجود در آن خوشه است. در نتیجه Purity کل خوشه‌بندی را می‌توان برابر با مجموع تعداد نمونه‌های کلاس غالب در هر خوشه تقسیم بر تعداد کل نمونه‌ها در نظر گرفت.

البته به طور کلی، دقت و عملکرد خوشه‌بندها به قدر کافی خوب نبود و با وجود اعمال کاهش بعد روی داده‌ها، باز هم میزان جدایی پذیری خوشه‌ها و شباهت لیبل‌های درون خوشه‌ها بالا نبود.

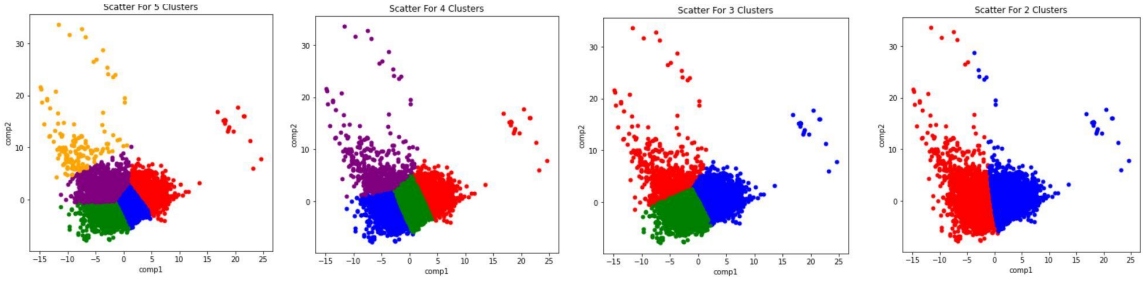
در نوت‌بوک کدهای پروژه، هر یک از این ۳ الگوریتم برای تعداد خوشه‌های ۲ تا ۵ اجرا شده‌اند. در هر ۴ حالت، معیارهای Purity و Silhouette محاسبه شده‌اند و توزیع داده‌های هر کلاس (ژانر) در هر خوشه نمایش داده شده‌است. همچنین نمایش دو بعدی داده‌ها نیز به ازای هر تعداد خوشه رسم شد تا بهتر متوجه شویم که خوشه‌های جدید نتیجه‌ی تجزیه کدام یک از خوشه‌های پیشین هستند.

دو نکته قابل توجه در مورد تفاوت‌های خوشه‌بندی در این ۳ روش قابل مشاهده است. مورد اول این‌که **مرز تصمیم‌ها در دو روش KMeans و KMedoids تقریبا نزدیک به خط است** و خوشه‌ها با مرزهای خطی جدا شده‌اند در حالی‌که **در روش Hierarchical مرز تصمیم‌ها پیچیده‌تر و غیر خطی هستند**. مورد دوم نیز نوع جدا شدن و ایجاد شدن خوشه‌ی جدید است که **در روش Hierarchical، خوشه جدید معمولا از یکی از خوشه‌های قبلی مشتق می‌شود** و به عنوان بخشی از آن خوشه‌ی قبلی جدا و مستقل می‌شود. اما **در دو روش KMeans و KMedoids خوشه‌های جدید لزوما از یک خوشه‌ی قبلی جدا نمی‌شوند و ممکن است تکه‌هایی از تمام خوشه‌های قبلی را در بر بگیرند.**

بخش ۴ - ۲ - ۱) خوشه بند KMeans

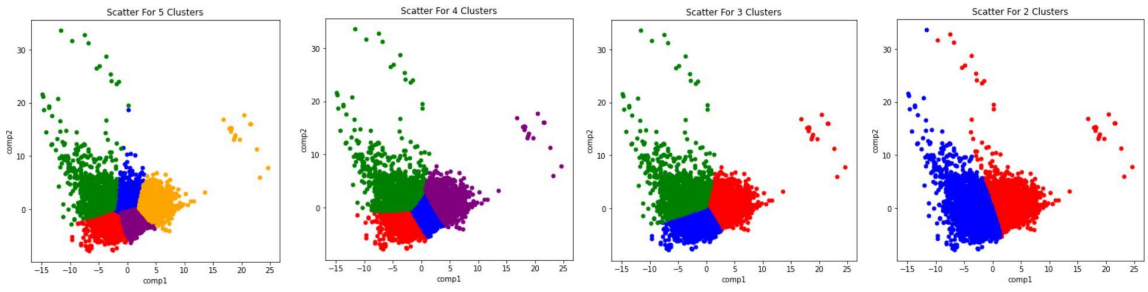
در خوشه‌بندی حاصل از روش KMeans، به نظر از همان ابتدا و با داشتن دو خوشه، ترانه‌های بندری از ترانه‌های گیلکی و لری بهتر از سایر دسته‌ها جدا شده‌اند و ترانه‌های ترکی و کردی به طور تقریبا مساوی تری بین دو خوشه پخش شده‌اند. با اضافه شدن خوشه بعدی، طبق انتظار، ترانه‌های بندری عمدتا در همان خوشه خودشان باقی مانده‌اند و خوشه‌ی دیگر بیشتر تجزیه شده‌است. در این تجزیه، به نظر بیشتر ترانه‌های کردی و ترکی از بقیه جدا شده‌اند و در خوشه‌ی جدیدتر قرار گرفته‌اند، هرچند خوشه‌ی جدید به نظر از هر دو خوشه‌ی قبلی نمونه‌هایی برداشته‌است. با چهار خوشه، همچنان بخشی از ترانه‌های بندری در همان خوشه قبلی خودشان مانده‌اند ولی بخشی از آن‌ها نیز وارد خوشه جدید شده‌اند که خیلی مطلوب ما نیست و خوشه جدید (سبزرنگ)، از ترکیب قسمت‌هایی خوشه‌ی قبلی حاصل شده‌است. خوشه آخر نیز عمدتا داده‌های نویزی خوشه بنفش رنگ در حالت چهارخوشه‌ای را گرفته‌است که بیشترشان ترانه‌های لری هستند. ترانه‌های ترکی که به نظر عمدتا در بخش‌های پایینی نمودار هستند نیز به نظر دورترین داده‌ها به این خوشه‌ی جدید (نارنجی رنگ) هستند.

همچنین از لحاظ دقت و عملکرد، این ۴ حالت به نظر خیلی تفاوتی از لحاظ Parity ندارند و معیار مربوط به فاصله (Silhouette) نیز با توجه به تصویر و قرارگیری خوشه‌ها خیلی معنادار نیست و آن هم تقریبا تغییر خاصی نداشته‌است. می‌توان گفت عملکرد در همه حالات تقریبا مشابه بوده‌است.



بخش ۴ - ۲ - ۲) خوشه بند KMedoids

در خوشه‌بندی حاصل از روش KMedoids، در حالت دو خوشه‌ای، اتفاق مشابه با خوشه‌بندی به روش KMeans رخ داده‌است و ترانه‌های بندری و لری (و تا حدی گیلکی) بهتر از بقیه جدا شده‌اند و در cluster های جداگانه قرار گرفته‌اند. خوشه سوم نیز همانند بخش قبل، بخش‌هایی از هر دو خوشه‌ی حالت ۲ خوشه‌ای را گرفته‌است که عمدتا ترانه‌های لری را پوشش می‌دهد و همچنان در تمامی حالات، خوشه‌ی سمت راست شامل اکثر ترانه‌های بندری است. با اضافه شدن خوشه چهارم ترانه‌های بندری بیشتر در این خوشه قرار گرفته‌اند و خوشه پنجم نیز، به نظر آنقدر خوشه مورد اطمینان و جالبی نیست چون به تعداد تقریبا مساوی از تمامی ژانرهای آهنگ‌ها نماینده دارد.

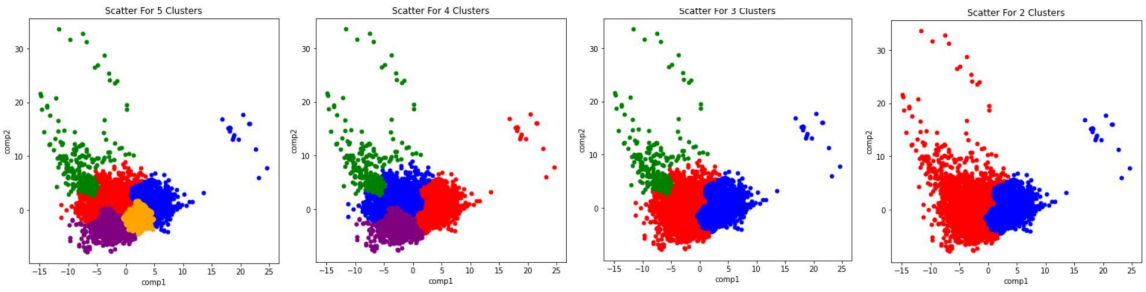


بخش ۴ - ۲ - ۳) خوشه بند Hierarchical با معیار فاصله‌ی Ward Linkage

در خوشه‌بندی حاصل از این روش، همانگونه که گفته‌شد مرز تصمیم‌ها برعکس حالات قبلی، خطی نیستند و اشکال پیچیده‌تری دارند. در حالت دو خوشه‌ای، به نظر خوشه‌ی کوچکتر بیشتر ترانه‌های بندری را در بر گرفته‌است و همانند دو طبقه‌بند قبلی، ترانه‌های لری و گیلکی عمدتا در خوشه‌ی دیگر قرار گرفته‌اند. همین‌جا با توجه به مشاهده شدن این اشتراک بین هر سه روش می‌توان متوجه شد که **جداسازی داده‌های ترانه‌های بندری از ترانه‌های گیلکی و لری به نظر کار راحت‌تری بوده‌است.** خوشه سوم، بخش کوچکی شامل داده‌های نویزی خوشه‌ی بزرگتر را در بر گرفته‌است. در این قسمت به نظر باز هم ترانه‌های لری و گیلکی بیشتر هستند با این تفاوت که

بیشتر داده ها حالت نویز و دور از تراکم اصلی داده‌ها هستند. خوشه‌ی چهارم نیز بخش دیگری از خوشه بزرگتر است که به نظر، توانسته است ترانه‌های ترکی را مقداری از این خوشه جدا کنه و کمتر ترانه‌ی ترکی بوده که وارد این خوشه چهارم شده باشد. در نهایت، خوشه پنجم اضافه شده‌است که این بار، از خوشه‌ی کوچکتر در حالت دو خوشه‌ای (خوشه‌ی در بر گیرنده‌ی عمده‌ی ترانه‌های بندری) جدا شده‌است. البته، چیزی که مشخص است، این است که این فرآیند ها در اصل به طور برعکس انجام گرفته‌اند. یعنی در اجرای الگوریتم Hierarchical، ابتدا تمامی data point ها جدا از هم قرار گرفته‌اند و آنقدر با یکدیگر join شده‌اند که تعداد خوشه‌ها تک‌رقمی شود. یعنی ابتدا به ۵ خوشه رسیده‌است و سپس به ۴ و ... . پس تمامی مراحل گفته شده، در اصل در جهت عکس انجام شده‌اند.

همچنین مورد عجیبی که در این الگوریتم دیده شد، مقدار برابر Purity در حالات مختلف آن بود که به نظر، به علت تغییر نکردن کلاس غالب (ژانر غالب) در حالات مختلف در خوشه‌هاست.



بخش ۵) کارهای آینده

- اختصاص دادن ضریب اهمیت به داده‌های موجود در ستون‌ها:

یکی از مواردی که در ابتدا گفته شد، این مورد بود که ۵ ویژگی مثل مجموع تعداد Zero Crossing ها، ضرایب MFCC و ... را برای توصیف قطعات یک‌دقیقه‌ای بر می‌گزینیم. اما چیزی که پس از استخراج این

ویژگی‌ها می‌بینیم آن است که به ویژگی ضرایب MFCC در مجموع ۴۰ ستون و به ویژگی Zero Crossing تنها ۱ ستون اختصاص داده شده‌است، در حالی که هر دو یک فایل صوتی یک‌دقیقه‌ای را به طور کامل توصیف می‌کنند، در حالی‌که اطلاعات MFCC ها از مجموعه‌ی ۴۰ ستون به دست می‌آید. به طور خلاصه، به نظر باید به Zero Crossing اهمیت بیشتری نسبت یک ستون تنها از ضرایب MFCC (مثل میانگین ضریب پنجم MFCC) داد، که احتمالا وزن دادن به داده‌های موجود در هر ستون می‌تواند حلال این مسئله باشد و به طور مثال **شاید**، کار Clustering را کمی بهبود ببخشد و به فیچرهای مهم‌تر و پر اطلاعات‌تر، وزن بیشتری بدهد.

- استفاده کردن از شبکه‌های عصبی برای طبقه‌بندی:

به طور کلی، شبکه‌های عصبی ابزاری عمومی برای کلاس‌بندی هر نوع داده‌ای هستند و می‌توان با استفاده از آن‌ها، مدل‌های قدرتمندتری برای طبقه‌بندی طراحی کرد.