**محمد جواد رنجبر**

**۸۱۰۱۰۱۷۳**

**تمرین سوم پردازش گفتار**

**دکتر ویسی**

**بهار ۱۴۰۳**

## فهرست

فهرست اشکال

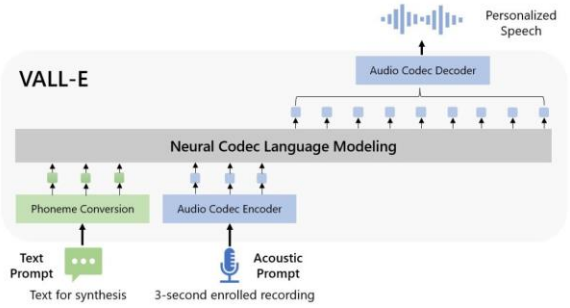
فهرست جداول

## سوال ۱

تولید متن به گفتار:

مدل VALL-E

این مدل رویکرد مدل زبانی برای تبدیل متن به صوت پیشنهاد داد. برای این یک مدل کدک زبانی عصبی آموزش داده شده و تبدیل متن به گفتار را به عنوان یک کار مدل‌سازی زبان شرطی در نظر گرفته شده است. برای آموزش این مدل بیش از ۶۰ هزار ساعت داده‌ی صوتی جمع‌آوری شده است. به عبارت ساده‌تر این مدل ابتدا صوت را به توکن‌های آکوستیک تبدیل می‌کند و سپس مدل زبانی شرطی بر اساس متن ورودی صوت مورد نظر را تولید می‌کند.




شکل ۱ مدل VALL-E

قابلیت‌های این مدل شامل موارد زیر است:

- **یادگیری در زمینه**[1]: مدل قابلیت یادگیری در زمینه را دارد. یعنی با استفاده از سه ثانیه صوت یک گوینده‌ی جدید می‌تواند با صدای آن شخص صوت تولید کند.
- **ترجمه متقابل بدون نمونه**[2]: یکی از ویژگی‌های قابل توجه VALL-E (X) توانایی آن در انجام وظایف ترجمه گفتار به گفتار است.
- **لهجه**: بر اساس توانایی مدل‌سازی گفتار متقابل زبانی آموخته‌شده با شناسه زبان معرفی‌شده، می‌تواند برای هر گوینده‌ای به زبان مادری گفتار تولید کند و می‌تواند مشکل لهجه خارجی را که یک مشکل شناخته شده در زبان متقابل است، به میزان قابل توجهی کاهش دهد.

به صورت کلی این مدل قابلیت تولید صوت با کیفیت بالا و شباهت به انسان را دارد، البته مدل‌هایی با کیفیت بهتر همچون VoiceBox نیز بعد از این مدل ارائه شدند که توضیحات آن در تمرین اول من گذاشته شده است.

تولید موسیقی:

مدل MusicGen:

این مدل نیز برای تولید موسیقی با استفاده از توصیف متنی ورودی استفاده می‌شود. این مدل نیز مانند مدل قبل از مدل زبانی شرطی برای تولید صوت استفاده می‌کند. بخش‌های این مدل به شرح زیر می‌باشد:

[1] In context Learning
[2] Zero-shot speech-to-speech translation

- **رمزگذار متن** :مدل با یک رمزگذار متن شروع می‌شود که توصیفات متنی را به یک دنباله از توکن‌ها تبدیل می‌کند. این امر به مدل اجازه می‌دهد تا زمینه و ویژگی‌های موسیقی که باید تولید کند را درک کند.
- **ترانسفورمر خودرگرسیو** MusicGen :از یک مدل زبان ترانسفورمر استفاده می‌کند که به صورت خودرگرسیو عمل می‌کند. این بدان معناست که بر اساس دنباله‌های قبلی، دنباله بعدی از توکن‌های صوتی را پیش‌بینی می‌کند و یک قطعه موسیقی هماهنگ ایجاد می‌کند.
- **تداخل توکن‌ها** :مدل از الگوهای تداخل توکن‌های کارآمد استفاده می‌کند. این رویکرد نوآورانه نیاز به مدل‌های متعدد متوالی را که پیش از این برای وظایفی مانند تولید سلسله مراتبی یا افزایش نمونه‌برداری لازم بود، حذف می‌کند.
- **توکن‌های صوتی** :توکن‌های صوتی تولید شده، یا کدکهای صوتی، سپس با استفاده از یک مدل فشرده‌سازی صوتی، مانندEnCodec، رمزگشایی می‌شوند تا موج صوتی نهایی تولید شود.
- **تولید کنترل شده** :یکی از ویژگی‌های کلیدی MusicGen توانایی آن در شرطی شدن بر اساس توصیفات متنی یا ویژگی‌های ملودیک است. این به خالقان کنترل بهتری بر سبک، حالت و سایر جنبه‌های موسیقی تولید شده می‌دهد.
- **کیفیت و ارزیابی** :نشان داده شده است که MusicGen قادر به تولید نمونه‌های موسیقی با کیفیت بالا است. این مدل ارزیابی تجربی گسترده‌ای را شامل مطالعات خودکار و انسانی، برای اطمینان از عملکرد برتر آن نسبت به سایر مدل‌ها، تجربه کرده است.

این مدل نسبت به مدل‌های قبلی موسیقی با کیفیت بالاتر و بهتر تولید می‌کند.

## سوال ۲

۲-۱

$$y(t) = a_0 w(t) + a_1 w(t-1) + \cdots + a_N w(t-N)$$

الف)

حال معادله کلی را می‌نویسیم و جاگذاری می‌کنیم:

$$\widehat{X^L} = \mathrm{Cov}(X,Y) \cdot \mathrm{Var}(Y) \cdot (Y - E[Y]) + E[X]$$

با توجه به اینکه $w$ از یک نویز سفید گوسی با توزیع همگن $N(0,1)$ می‌آید، توقع داریم میانگین $y$ها نیز برابر با صفر باشد. $(\bar{y}(t) = 0)$.

$$\hat{y}(t+1) = \bar{y}(t+1) + \Sigma_{y(t+1)y(t)}\Sigma_{y(t)}^{-1}\big(y(t) - \bar{y}(t)\big) = \Sigma_{y(t+1)y(t)}\Sigma_{y(t)}^{-1} y(t)$$

حال واریانس و کورایانس را محاسبه می‌کنیم:

$$\Sigma_{y(t)} = E(a_0 w(t) + a_1 w(t-1) + \cdots + a_N w(t-N) - \bar{y}(t))^2$$

$$= E(a_0 w(t) + a_1 w(t-1) + \cdots + a_N w(t-N))^2 = \sum_{i=0}^{N} a_i^2$$

حال، از آنجایی که $w(t)$ دارای میانگین صفر است و در موارد زمانی مختلف همبستگی ندارد، expected value آن را می‌توان به مجموع مجذورات ضرایب کاهش داد.

همینطور برای کواریانس داریم:

$$\mathrm{Cov}(X,Y) = \mathrm{E}[(X - \mathrm{E}[X])(Y - \mathrm{E}[Y])] = \mathrm{E}[XY] - (\mathrm{E}[X])(\mathrm{E}[Y])$$

$$\begin{aligned}\Sigma_{y(t+1)y(t)} &= \mathrm{E}(y(t+1)y(t)) - \mathrm{E}(y(t+1))(\mathrm{E}(y)) = \mathrm{E}(y(t+1)y(t)) \\ &= \mathrm{E}(a_0 w(t+1) + a_1 w(t) + \cdots + a_N w(t+1-N))(a_0 w(t) + a_1 w(t-1) + \cdots \\ &+ a_N w(t-N)) = \sum_{i=0}^{N} a_{i+1}\, a_i\end{aligned}$$

با جاگذاری در عبارت اولیه داریم:

$$\hat{y}(t+1) = \Sigma_{y(t+1)y(t)} \Sigma_{y(t)}^{-1} y(t) = \frac{\sum_{i=0}^{N} a_{i+1}\, a_i}{\sum_{i=0}^{N} a_i^2} y(t)$$

ب)

ابتدا متغییر $x(t) = [y(t+1) \quad y(t-1)]^T$ را تعریف می‌کنیم و قصد محاسبه‌ی $\hat{y}(t) = \mathrm{E}(y(t)|x(t))$ را داریم.

حال عبارت کلی را بازنویسی می‌کنیم:

$$\hat{y}(t) = \bar{y}(t) + \Sigma_{x(t)y(t)} \Sigma_{x(t)}^{-1} \big(y(t) - \bar{y}(t)\big) = \Sigma_{x(t)y(t)} \Sigma_{x(t)}^{-1} y(t)$$

کواریانس و واریانس را محاسبه می‌کنیم:

$$\begin{aligned}\Sigma_{y(t)x(t)} &= \mathrm{E}[y(t)[y(t+1)^T y(t-1)^T]] = \mathrm{E}[y(t)y(t+1)^T\, y(t)y(t-1)^T] \\ &= \left[\sum_{i=0}^{N-1} a_{i+1}\, a_i \sum_{i=0}^{N-1} a_{i+1}\, a_i\right]\end{aligned}$$

$$\Sigma_{x(t)} = \mathrm{E}[y(t+1) \quad y(t-1)]^T [y(t+1) \quad y(t-1)] = \begin{bmatrix} \sum_{i=0}^{N} a_i^2 & \sum_{i=0}^{N-2} a_{i+2}\, a_i \\ \sum_{i=0}^{N-2} a_{i+2}\, a_i & \sum_{i=0}^{N} a_i^2 \end{bmatrix}$$

بنابراین داریم:

$$\hat{y}(t) = \left[\sum_{i=0}^{N-1} a_{i+1}\, a_i \sum_{i=0}^{N-1} a_{i+1}\, a_i\right] \begin{bmatrix} \sum_{i=0}^{N} a_i^2 & \sum_{i=0}^{N-2} a_{i+2}\, a_i \\ \sum_{i=0}^{N-2} a_{i+2}\, a_i & \sum_{i=0}^{N} a_i^2 \end{bmatrix}^{-1} \begin{bmatrix} y(t+1) \\ y(t-1) \end{bmatrix}$$

بنابراین تخمین به صورت زیر محاسبه می‌شود:

$$\hat{y}(t) = \frac{\sum_{i=0}^{N-1} a_{i+1}\, a_i}{\sum_{i=0}^{N} a_i^2 + \sum_{i=0}^{N-2} a_{i+2}\, a_i}\,(y(t+1) + y(t-1))$$

۲-۲

$$p_x(k) = P(X = k) = e^{-\lambda}\frac{\lambda^k}{k!}, k = 0,1,2,\cdots$$

برای به دست آوردن تخمین بشینه شباهت داریم:

$$l(\lambda; x_1, \cdots x_n) = \ln\left(\prod_{j=1}^{n} e^{-\lambda}\frac{\lambda^{x_j}}{x_j!}\right) = \sum_{j=1}^{n} \ln\left(e^{-\lambda}\frac{\lambda^{x_j}}{x_j!}\right) = \sum_{j=1}^{n}[\ln(e^{-\lambda}) + \ln(\lambda^{x_j}) - \ln(x_j!)]$$

$$= \sum_{j=1}^{n}[-\lambda + x_j \ln(\lambda) - \ln(x_j!)] = -n\lambda + \ln(\lambda)\sum_{j=1}^{n} x_j - \sum_{j=1}^{n}\ln(x_j!)$$

حال نسبت به متغییر $\lambda$ مشتق می‌گیریم:

$$\frac{d}{d\lambda} l(\lambda; x_1, \cdots x_n) = 0 \Rightarrow \frac{d}{d\lambda}\left(-n\lambda + \ln(\lambda)\sum_{j=1}^{n} x_j - \sum_{j=1}^{n}\ln(x_j!)\right) = 0$$

$$\frac{d}{d\lambda} l(\lambda; x_1, \cdots x_n) = -n + \frac{1}{\lambda}\sum_{j=1}^{n} x_j - 0 = 0 \Rightarrow \lambda = \frac{1}{n}\sum_{j=1}^{n} x_j$$

## سوال ۳

برای درست کردن تابع ex3_windowing با توجه به خواسته‌ی سوال پنجره‌های مختلف شامل 'rect','hann','cosine','hamming' را در تابع خود با استفاده از کتابخانه numpy پیاده‌سازی می‌کنیم. حال روی صوت ورودی حرکت می‌کنیم و هر پنجره را در frame اصلی ضرب می‌کنیم. فریم صدادار ۲۳ را برای نمایش انتخاب کرده‌ایم.

نمودارهای بدست آمده برای پنجره‌های مختلف به شکل زیر می‌باشد:

Hamming:

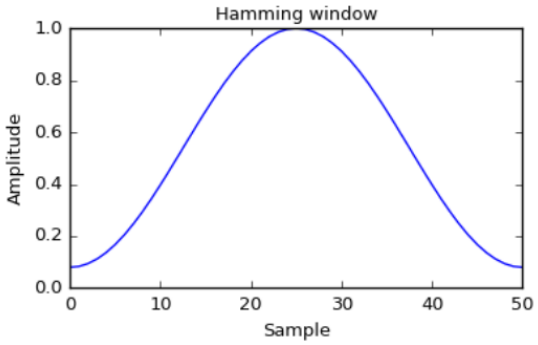
شکل این پنجره به صورت زیر می‌باشد.

- پنجره همینگ توسط فرمول زیر تعریف می‌شود :

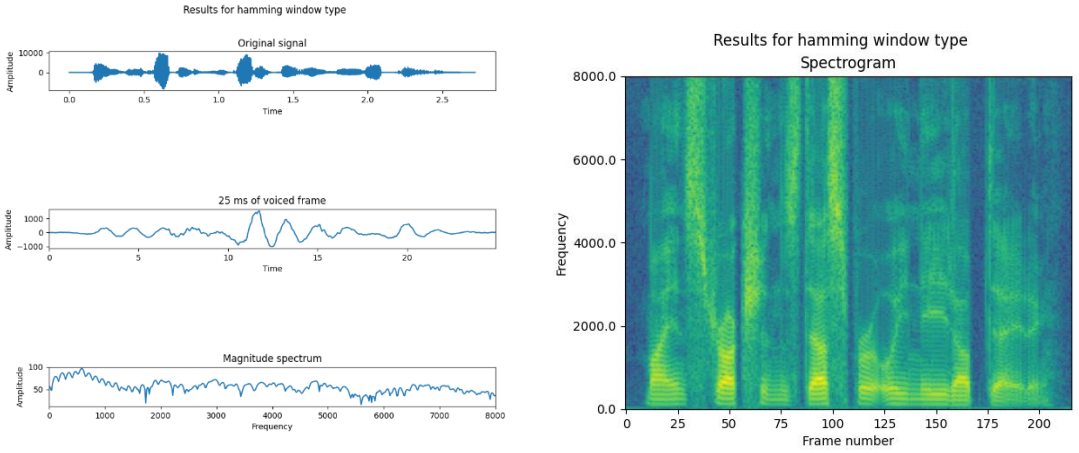
$$w(n) = 0.54 - 0.46\cos\left(\frac{2\pi n}{N}\right)$$

- این پنجره تعادلی بین عرض لوب اصلی و سطح لوبهای جانبی ارائه می‌دهد.
- پنجره همینگ ترنزیشنی نرمتر از لوب اصلی به لوبهای جانبی نسبت به پنجره مستطیلی دارد که منجر به دقت بیشتر فرکانسی می‌شود.
- با این حال، هنوز لوبهای جانبی دارد که می‌تواند در برخی از برنامه‌ها نشت فرکانسی ایجاد کند.

شکل این پنجره به صورت زیر می‌باشد.




شکل ۲ پنجره همینگ



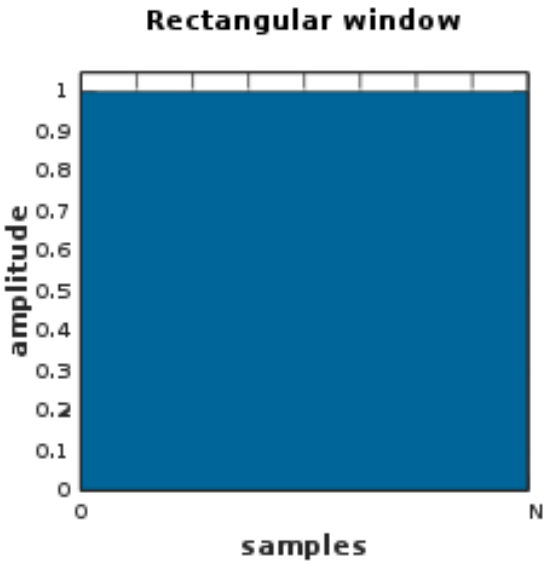

شکل3 نتایج پنجره همینگ

Rectangle:

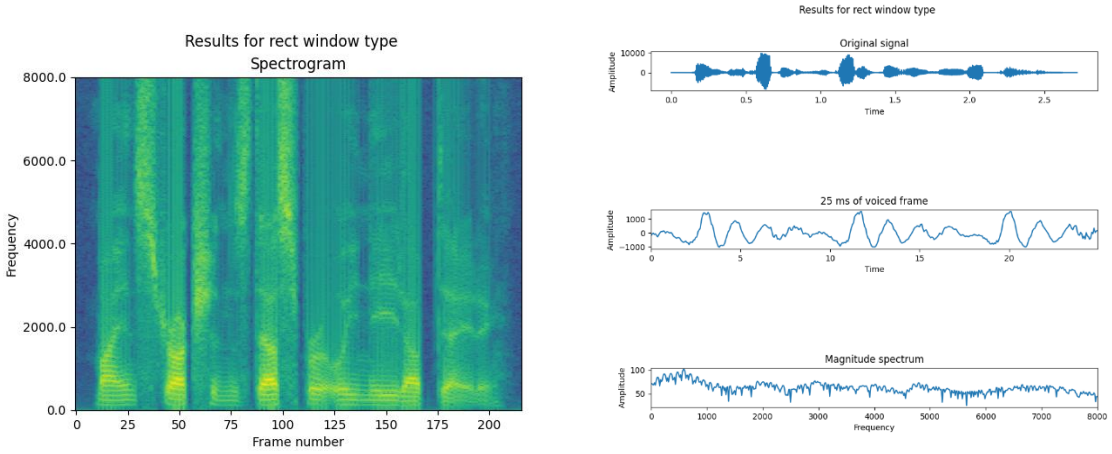
- پنجره مستطیلی ساده‌ترین توابع پنجره است.
- این به عنوان یک مقدار ثابت ۱ در دامنه پنجره تعریف شده است و بیرون از آن صفر است.

- اگرچه این پنجره دارای باریک‌ترین لوب اصلی است، اما از لوبهای جانبی بالایی رنج می‌برد که منجر به دقت فرکانسی نامطلوب و نشت فرکانسی می‌شود.

شکل این پنجره به صورت زیر می‌باشد.




شکل4 پنجره مربعی




شکل5 نتایج پنجره مربعی

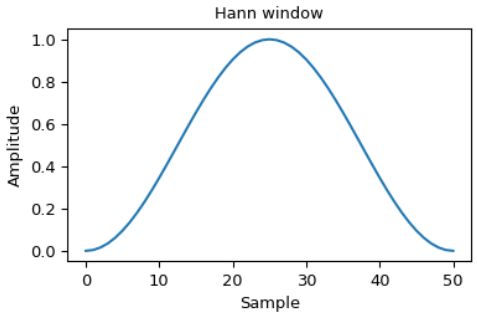
Hann:

- پنجره هان مشابه پنجره همینگ است اما ضریب متفاوتی در جلوی عبارت کوسینوسی دارد. تفاوت بین پنجره‌ی hann و hamming این است که پنجره hann در دو سر آن به صفر می‌رسد و هر گونه وقفه‌ای را برطرف می‌کند. اما پنجره hamming کمی قبل از صفر متوقف می‌شود، به این معنی که سیگنال هنوز دارای یک وقفه کوچک خواهد بود.
- فرمول پنجره هان به صورت زیر است:

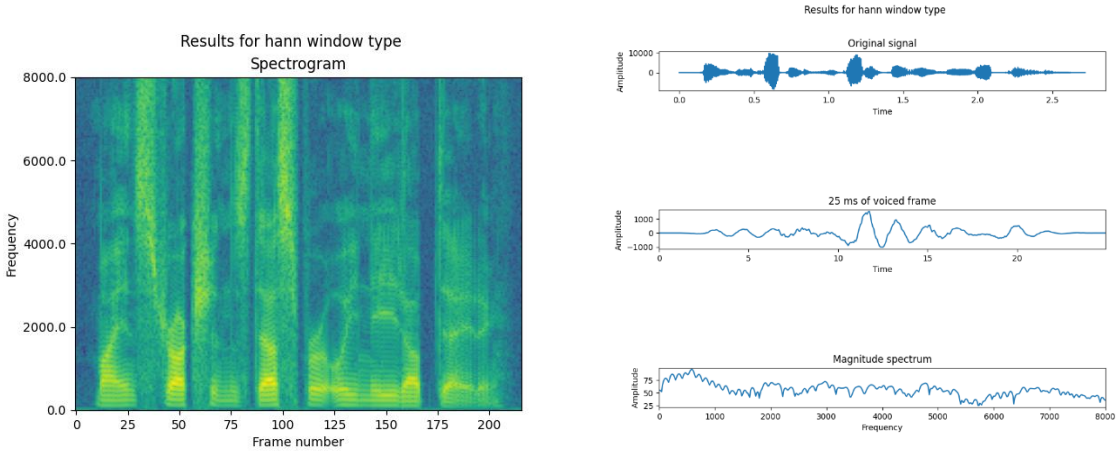
$$w(n) = 0.5\left(1 - \cos\left(\frac{2\pi n}{N}\right)\right)$$

- مانند پنجره همینگ، این پنجره هم تعادلی بین عرض لوب اصلی و سطح لوبهای جانبی ارائه می‌دهد اما با ویژگی‌های کمی متفاوت.
- پنجره هان در دو سر آن به صفر می‌رسد که هر گونه وقفه‌ای را برطرف می‌کند، که به خصوص در مواردی که داده‌ها نزدیک لبه‌های پنجره نیاز به حفظ دارند، مفید است.

شکل این پنجره به صورت زیر می‌باشد.




شکل6 پنجره هان

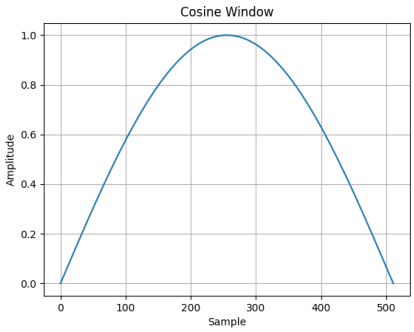



شکل7 نتایج پنجره هان

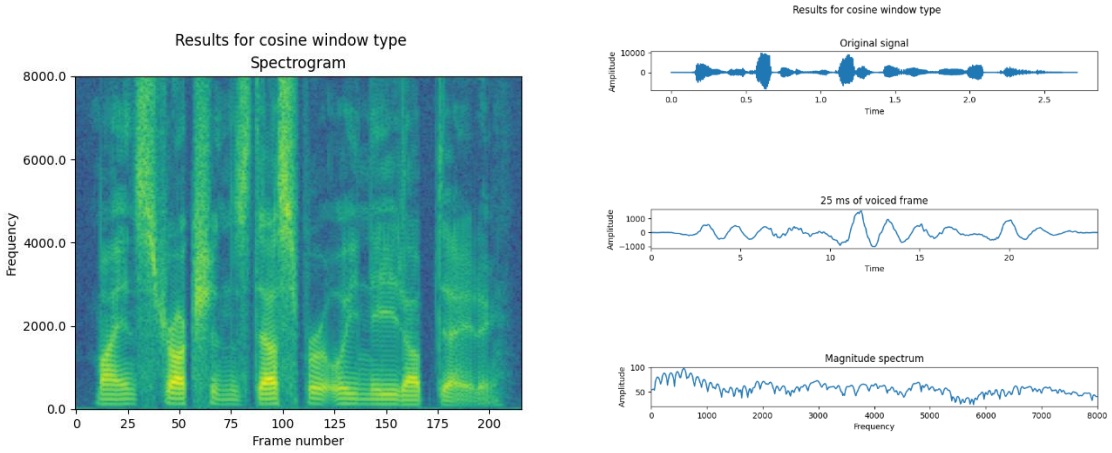
Cosine:

- پنجره کوسینوسی، همچنین به نام پنجره ولچ شناخته می‌شود، نسخه اصلاح شده‌ای از پنجره هان است.
- پنجره کوسینوسی عرض لوب اصلی بهتری و سرکوب لوبهای جانبی نسبت به پنجره هان ارائه می‌دهد.
- با این حال، آن یک زمینه آهسته‌تر را نسبت به پنجره هان دارد، به این معنی که لوب اصلی آن گسترده‌تر است.

شکل این پنجره به صورت زیر می‌باشد.




شکل8 پنجره کوسینوس




شکل9 نتایج پنجره کوسینوس

## سوال ۴

ابتدا کلاس‌های مربوط به KNN و SVM را پیاده‌سازی می‌کنیم.

:KNN

فاز آموزش:

KNN تمام نقاط داده موجود و برچسب های کلاس مربوط به آنها را ذخیره می کند. این مجموعه داده آموزشی را تشکیل می دهد.

فاز پیش بینی:

برای یک نقطه داده ورودی (یا نمونه)، KNN فاصله بین این نقطه و هر نقطه دیگر در مجموعه داده را محاسبه می کند. معیار محاسبه فاصله بسته به کاربرد می‌تواند متفاوت باشد، در اینجا برای سادگی از معیار فاصله اقلیدسی استفاده کرده‌ایم.

پس از محاسبه فواصل، KNN نزدیکترین k همسایه‌های نقطه داده ورودی را شناسایی می کند. اینها نقاط داده با کمترین فاصله تا نقطه ورودی هستند. حال برحسب بیشترین تعداد لیبل مشترک این داده ورودی را دسته‌بندی می‌کنیم.

:SVM

فاز آموزش:

SVM با گرفتن یک مجموعه داده آموزشی شامل نقاط داده ورودی (بردارها) و برچسب های کلاس یا مقادیر هدف مربوطه آنها شروع می شود.

هدف SVM یافتن ابر صفحه بهینه است که نقاط داده کلاس های مختلف را در فضای ویژگی به بهترین نحو از هم جدا می کند.

برای دستیابی به این هدف، SVM زیرمجموعه ای از نقاط داده آموزشی به نام بردارهای پشتیبانی را شناسایی می کند. اینها نقاط داده ای هستند که نزدیکترین نقاط به مرز تصمیم (هیپرپلین) هستند.

هدف SVM به حداکثر رساندن حاشیه است که فاصله بین مرز تصمیم و نزدیکترین نقاط داده (بردارهای پشتیبانی) هر کلاس است. حاشیه بزرگتر به طور کلی منجر به عملکرد تعمیم بهتر می شود.

بهینه سازي:

SVM یک مسئله بهینه سازی را برای یافتن ابر صفحه بهینه حل می کند. هدف آن به حداقل رساندن خطای طبقه بندی (SVM حاشیه نرم) یا به حداکثر رساندن حاشیه (SVM با حاشیه سخت) است، در حالی که طبقه بندی اشتباه را نیز جریمه می کند. مسئله بهینه سازی شامل یافتن ضرایب (وزن) هایپرپلن و عبارت بایاس است. این معمولا با استفاده از تکنیک های بهینه سازی محدب انجام می شود. در شکل زیر مساله کلی بهینه‌سازی قابل مشاهده است.

$$\begin{aligned} \min_{\xi,w,b} \quad & \frac{1}{2}||w||^2 + C\sum_{i=1}^{m}\xi_i \\ \text{s.t} \quad & y^{(i)}\left(w^T x^{(i)} + b\right) \geq 1 - \xi_i, \\ & \xi_i \geq 0;\ i = 1, \ldots, m. \end{aligned}$$

شکل ۱۰ معادله بهینه‌سازی SVM

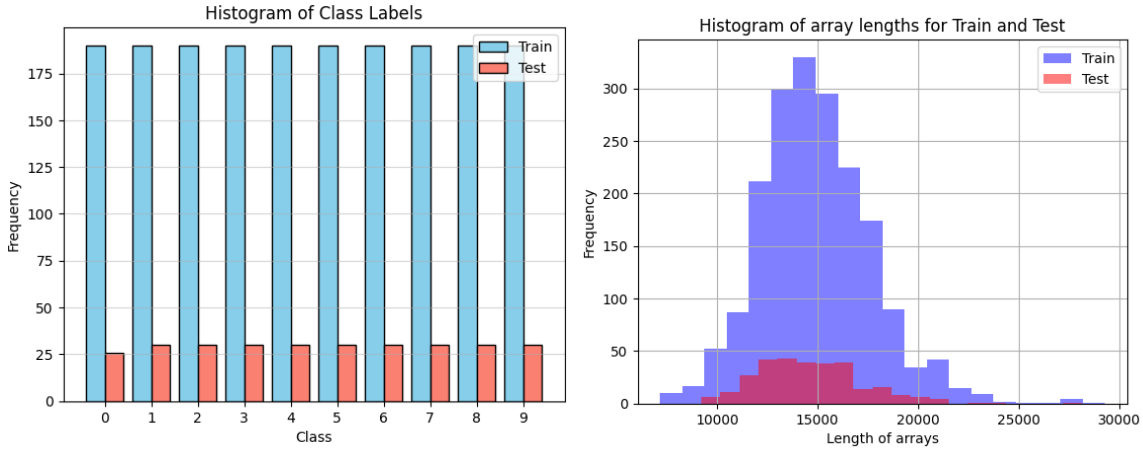
حال به کار با داده‌ها می‌پردازیم، ابتدا داده‌ها را با نرخ نمونه‌برداری ۱۶۰۰۰ هرتز باز می‌کنیم، همچنین برای بهبود عملکرد مدل، دو کار زیر را انجام می‌دهیم:

• نورمالایز کردن: این کار شامل استفاده از روش‌های نرمال‌سازی مانند استانداردسازی برای تبدیل داده‌ها به بازه‌ای مشخص می‌شود، که باعث کاهش تفاوت‌های مقیاس و افزایش استقلال متغیرها می‌شود.

• حذف سکوت: در این مرحله، سکوت‌ها یا قسمت‌هایی از داده‌ها که هیچ اطلاعات مفیدی ندارند، حذف می‌شوند تا داده‌های ورودی به مدل کاهش یابد و اطلاعات موثرتری به مدل ارائه شود. و به بررسی داده‌ها می‌پردازیم.

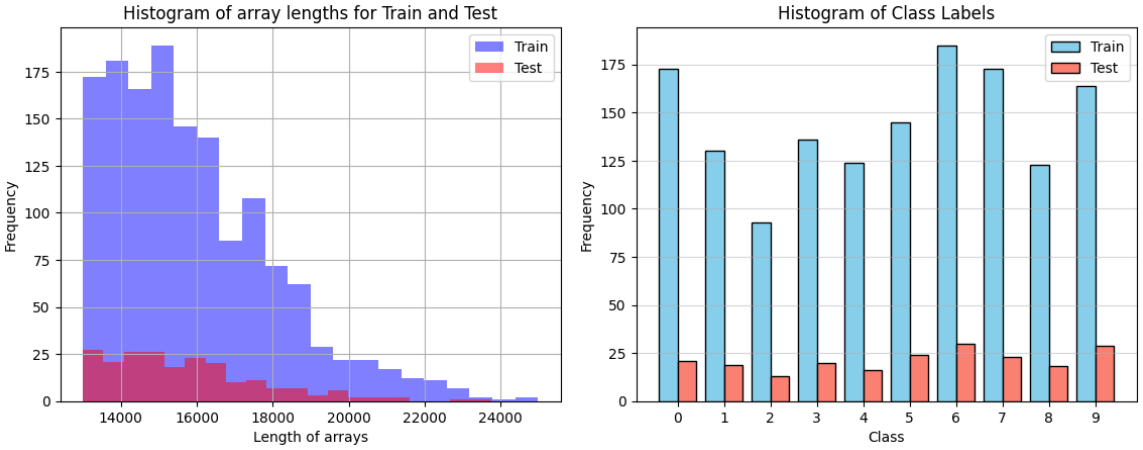
همچنین به بررسی آماری داده‌ها می‌پردازیم، واضح است که تعداد داده‌ها در کلاس‌های مختلف یکسان است بنابراین نیاز به پردازشی

در این زمینه نداریم. با این حال طول داده‌ها با یکدیگر متفاوت است.




شکل ۱۱ توزیع داده‌ها پیش از تمیز کردن

برای حل این مشکل، از padding استفاده می‌کنیم همچنین داده‌هایی که طول بسیار زیاد دارند، یا طول بسیار کم دارند را حذف می‌کنیم، چون هم شامل noise ممکن است باشند و هم باعث می‌شوند که داده‌ها ما تعداد زیادی صفر بی معنی داشته‌باشند. داده‌ها به صورت زیر تغییر می‌کنند.




شکل ۱۲ توزیع داده‌ها بعد از تمیز کردن

شکل بالا بدون استفاده از padding است و طول واقعی داده‌ها را نشان می‌هد. حال به سراغ استخراج ویژگی می‌رویم.

الف)

با توجه به خواسته‌های سوال که شامل:

- طول فریم: ۲۰ میلی‌ثانیه
- تعداد فیلترهای مل: ۲۴
- تعداد ویژگی‌ها: ۱۲
- همراه با مشتقهای اول و دوم

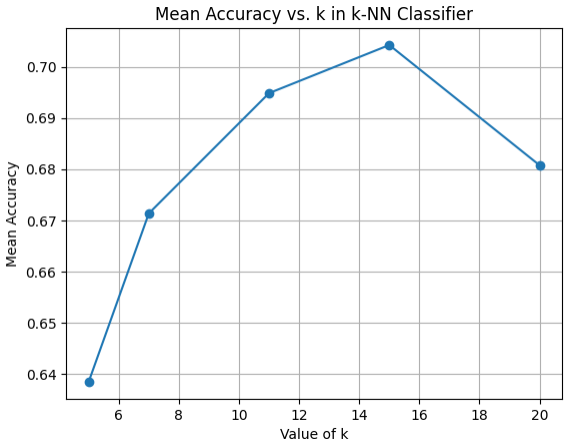
برای ایجاد ویژگی‌های MFCC از کتابخانه librosa استفاده شده است. ابتدا، ویژگی‌های اصلی MFCC با استفاده از تابع **librosa.feature.mfcc**با پارامترهای مناسب از محاسبه می‌شوند. سپس میانگین ویژگی‌های MFCC بر روی محور زمان محاسبه می‌شود تا به MFCC نهایی برسیم.

سپس، مشتقهای اول و دوم MFCC نیز محاسبه می‌شوند با استفاده از تابع **librosa.feature.delta**. سپس میانگین مشتقها بر روی محور زمان محاسبه می‌شوند. سرانجام، تمامی ویژگی‌های MFCC اصلی و مشتق‌های آن‌ها با هم ترکیب می‌شوند تا ویژگی‌های نهایی مورد استفاده تولید شود.

حال از این ویژگی‌های تولید شده برای آموزش مدل KNN استفاده می‌کنیم، البته قبل از آموزش مدل با ویژگی‌ها را نورمالایز نیز می‌کنیم.

نتایج برای مدل KNN به صورت دستی نوشته شده به شرح زیر می‌باشد:

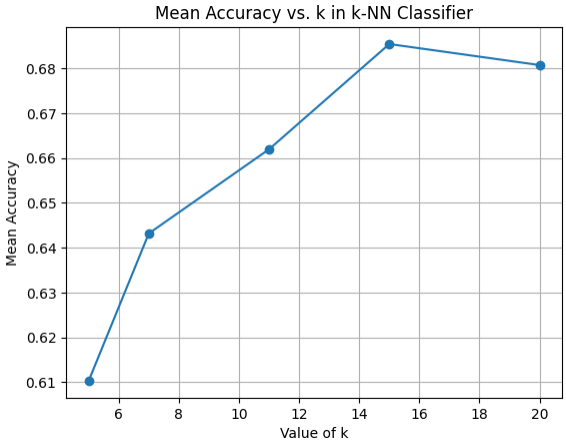
```
Accuracy for the test set with k=5: 0.6384976525821596
Accuracy for the test set with k=7: 0.6713615023474179
Accuracy for the test set with k=11: 0.6948356807511737
Accuracy for the test set with k=15: 0.704225352112676
Accuracy for the test set with k=20: 0.6807511737089202
Mean accuracy: 0.6779342723004694
```




شکل ۱۳ knn دست نویس برای MFCC

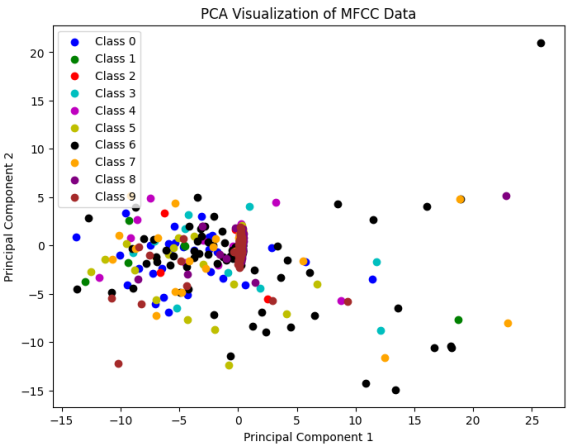
KNN با استفاده از کتابخانه:

```
Accuracy for the test set with k=5: 0.6103286384976526
Accuracy for the test set with k=7: 0.6431924882629108
Accuracy for the test set with k=11: 0.6619718309859155
Accuracy for the test set with k=15: 0.6854460093896714
Accuracy for the test set with k=20: 0.6807511737089202
Mean accuracy: 0.6563380281690142
```




شکل ۱۴ KNN کتابخانه برای MFCC

همچنین رسم PCA برای این داده‌ها به شکل زیر خواهد بود:




شکل ۱۵ نمودار PCA

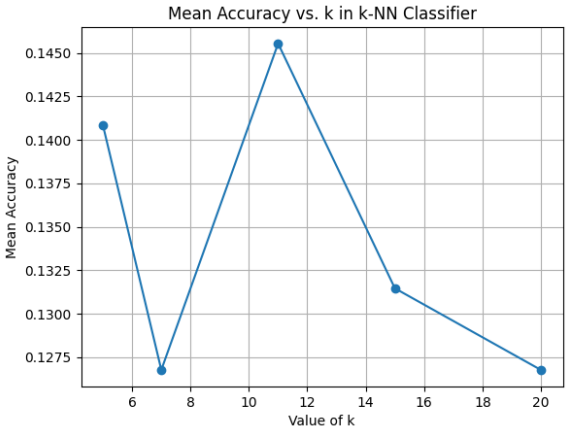
مشخص است که این داده‌ها به صورت خطی جدایی پذیر نمی‌باشند، ولی تا حدی داده‌ها کلاس‌های یکسان نزدیک به یکدیگر واقع شده‌اند.

ب)

حال ویژگی‌های LPC را بدست می‌آوریم. برای این کار نیز از کتابخانه‌ی Librosa استفاده می‌کنیم.

نتایج برای KNN به صورت دستی پیاده‌سازی شده:

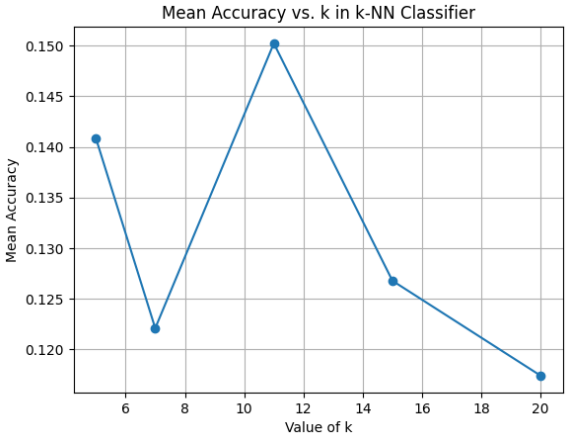
```
Accuracy for the test set with k=5: 0.14084507042253522
Accuracy for the test set with k=7: 0.1267605633802817
Accuracy for the test set with k=11: 0.14553990610328638
Accuracy for the test set with k=15: 0.13145539906103287
Accuracy for the test set with k=20: 0.1267605633802817
Mean accuracy: 0.13427230046948355
```




شکل ۱۶ KNN دست نویس برای LPC

KNN با استفاده از کتابخانه:

```
Accuracy for the test set with k=5: 0.14084507042253522
Accuracy for the test set with k=7: 0.12206572769953052
Accuracy for the test set with k=11: 0.15023474178403756
Accuracy for the test set with k=15: 0.1267605633802817
Accuracy for the test set with k=20: 0.11737089201877934
Mean accuracy: 0.13145539906103287
```




شکل ۱۷ KNN کتابخانه برای LPC

ج)

حال با استفاده از SVM این بخش‌ها را تکرار می‌کنیم که نتایج به شرح زیر می‌باشد:

Linear SVM and MFCC from scratch:

```
Accuracy for the test set with SVM linear kernel: 0.6666666666666666
```

Polynomial SVM and MFCC from scratch:

```
Accuracy for the test set with SVM linear kernel: 0.6525821596244131
```

Linear SVM and MFCC using library:

```
Accuracy for the test set with SVM linear kernel: 0.7182539682539683
```

Polynomial SVM and MFCC using library:

```
Accuracy for the test set with SVM linear kernel: 0.5586854460093896
```

Linear SVM and LPC from scratch:

```
Accuracy for the test set with SVM linear kernel: 0.09121621621621621
```

Polynomial SVM and LPC from scratch:

```
Accuracy for the test set with SVM linear kernel: 0.12837837837837837
```

Linear SVM and LPC using library:

```
Accuracy for the test set with SVM linear kernel: 0.14084507042253522
```

Polynomial SVM and LPC using library:

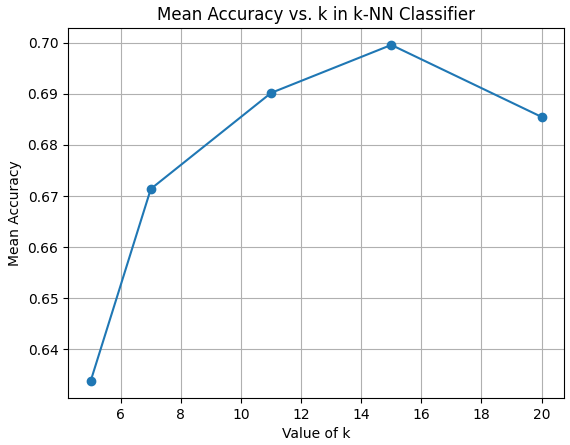
```
Accuracy for the test set with SVM linear kernel: 0.14084507042253522
```

حال نتایج را با ترکیب ویژگی‌ها بررسی می‌کنیم:

[MFCC;LPC]:

KNN from scratch:

```
Accuracy for the test set with k=5: 0.6338028169014085
Accuracy for the test set with k=7: 0.6713615023474179
Accuracy for the test set with k=11: 0.6901408450704225
Accuracy for the test set with k=15: 0.6995305164319249
Accuracy for the test set with k=20: 0.6854460093896714
Mean accuracy: 0.6760563380281691
```

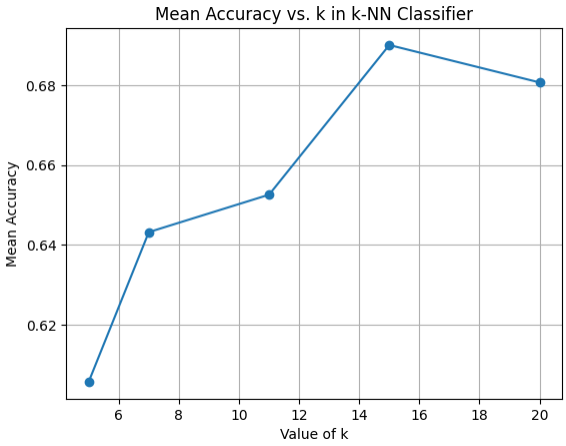



شکل18 KNN دست نویس برای [MFCC;LPC]

KNN using library:

```
Accuracy for the test set with k=5: 0.6056338028169014
```

```
Accuracy for the test set with k=7: 0.6431924882629108
Accuracy for the test set with k=11: 0.6525821596244131
Accuracy for the test set with k=15: 0.6901408450704225
Accuracy for the test set with k=20: 0.6807511737089202
Mean accuracy: 0.6544600938967136
```




شکل19 KNN کتابخانه برای [MFCC;LPC]

Linear SVM from scratch:

```
Accuracy for the test set with SVM linear kernel: 0.5633802816901409
```

Polynomial SVM from scratch:

```
Accuracy for the test set with SVM poly kernel: 0.5633802816901409
```

Linear SVM using library:

```
Accuracy for the test set with combined features: 0.6666666666666666
```

Polynomial SVM using library:

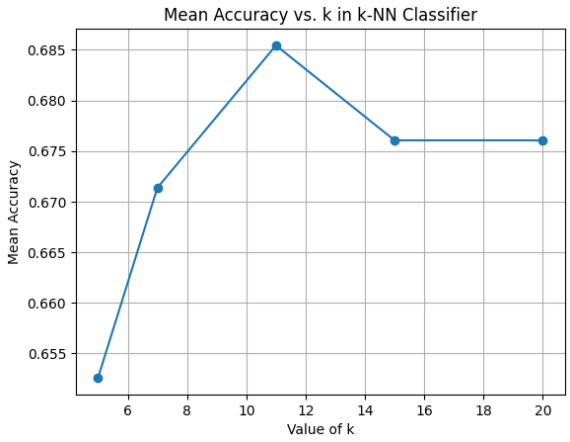
```
Accuracy for the test set with SVM linear kernel: 0.4225352112676056
```

[MFCC;LPC;ZCR]:

[MFCC;LPC;ZCR]

KNN from scratch:

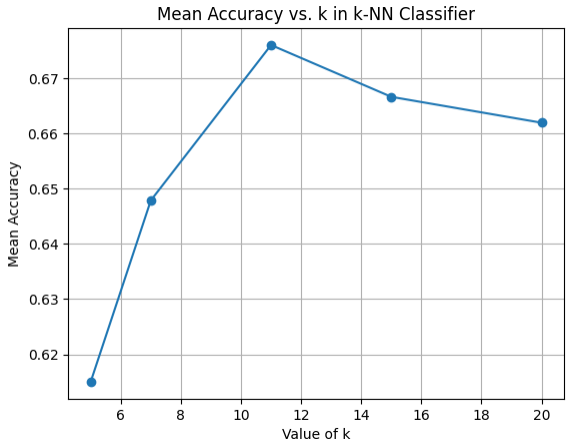
```
Accuracy for the test set with k=5: 0.6525821596244131
Accuracy for the test set with k=7: 0.6713615023474179
Accuracy for the test set with k=11: 0.6854460093896714
Accuracy for the test set with k=15: 0.676056338028169
Accuracy for the test set with k=20: 0.676056338028169
Mean accuracy: 0.6723004694835681
```




شکل20 KNN دست نویس برای [MFCC;LPC;ZCR]

KNN using library:

```
Accuracy for the test set with k=5: 0.6150234741784038
Accuracy for the test set with k=7: 0.647887323943662
Accuracy for the test set with k=11: 0.676056338028169
Accuracy for the test set with k=15: 0.6666666666666666
Accuracy for the test set with k=20: 0.6619718309859155
Mean accuracy: 0.6535211267605634
```




شکل21 KNN کتابخانه برای [MFCC;LPC;ZCR]

Linear SVM from scratch:

```
Accuracy for the test set with SVM linear kernel: 0.5868544600938967
```

Polynomial SVM from scratch:

```
Accuracy for the test set with SVM linear kernel: 0.5586854460093896
```

Linear SVM using library:

```
Accuracy for the test set with combined features: 0.6901408450704225
```

Polynomial SVM using library:

```
Accuracy for the test set with SVM linear kernel: 0.4507042253521127
```

| ویژگی | KNN(K=7) | SVM-linear | SVM-Poly |
|---|---|---|---|
| **MFCC** | 64% | 71% | 55% |
| **LPC** | 12% | 14% | 14% |
| **[MFCC;LPC]** | 64% | 66% | 42% |
| **[MFCC;LPC;ZCR]** | 64% | 69% | 45% |

جدول ۱ نتایج الگوریتم‌ها و ویژگی‌های مختلف

۱. **K-Nearest Neighbors (KNN) با K=7:**

- با استفاده از ویژگیMFCC ، دقت تشخیص ٪۶۴ است. این نتیجه نشان می‌دهد که ویژگی MFCC دقت نسبتا خوبی می‌دهد با این حال جای بهتر کردن دارد.
- با استفاده از ویژگیLPC ، دقت فقط ٪۱۲ است که بسیار پایین است و نشان می‌دهد که این ویژگی به تنهایی برای تشخیص صوت مفید نیست.
- استفاده از ترکیب ویژگی‌های MFCC و LPC نیز بهبود محسوسی به وجود نمیاورد و دقت دارد و به ٪۶۴ است.
- افزودن ویژگی ZCR به MFCC و LPC نیز بهبود محسوسی به وجود نمیاورد و دقت دارد و به ٪۶۴ است.

۲. **Support Vector Machine (SVM)با هسته خطی:**

- استفاده از ویژگیMFCC ، دقت ٪۷۱ را به دست می‌دهد که نسبتاً بهتر از KNN است، اما همچنان قابل بهتر کردن است.
- ویژگی LPC همچنان با دقت ٪۱۴ نتایج ضعیفی ارائه می‌دهد و اضافه کردن آن به MFCC بهبود زیادی نمی‌آورد.
- ترکیب MFCC و LPC باعث افزایش دقت به ٪۶۶ می‌شود.
- با اضافه کردن ZCR به ویژگی‌ها، دقت به ٪۶۹ افزایش می‌یابد که نشان از اثر مثبت این ویژگی است.

۳. **Support Vector Machine (SVM)با هسته چند جمله‌ای:**

- استفاده از MFCC با دقت ٪۵۵ نشان می‌دهد که این هسته برای این مسئله بهتر از خطی نیست.
- LPCبا دقت ٪۱۴ همچنان نتایج ضعیفی ارائه می‌دهد.
- ترکیب MFCC و LPC باعث افزایش دقت به ٪۴۲ می‌شود، اما این همچنان نسبت به دیگر روش‌ها ضعیف است.
- افزودن ZCR به MFCC و LPC به دقت ٪۴۵ منجر می‌شود که همچنان نشان از عملکرد ضعیف این هسته برای این مسئله است.

به طور کلی، نتایج نشان می‌دهد که SVM با هسته خطی بهترین عملکرد را ارائه می‌دهد در حالی که KNN و SVM با هسته چند جمله‌ای عملکرد ضعیف‌تری دارند. همچنین، ویژگی MFCC به تنهایی بهترین نتایج را ارائه می‌دهد و افزودن ویژگی‌های دیگر به آن بهبود محسوسی در دقت نمی‌آورد، به جز برای SVM با هسته خطی که افزودن ZCR به MFCC و LPC منجر به افزایش دقت می‌شود.