به نام او

یادگیری آماری

گزارش پروژه

دانشجو: زینب شریفی ۴۰۰۲۰۵۵۶۶

استاد: دکتر محمدزاده

بهار ۱۴۰۱

فاز ۱:

برای پردازش و آنالیز فایل های صوتی در پایتون از شیوه های متفاوتی میتوان بهره برد، من جمله کتابخانه scipy و پکیج librosa

کتابخانه scipy جامع تر و به طور کلی برای پردازش سیگنال به کار میرود. اما پکیج librosa به صورت اختصاصی برای پردازش و آنالیز صوت و موسیقی ایجاد شده است و نتیجتا توابع کاربردی تری دارد و استفاده از آن راحتتر است.

فایل .wav خوانده شده از این ۲ طریق یک تفاوت اساسی دارد: librosa دامنه سیگنال را بین ۱ و ۱- اسکیل میکند و نوع داده float32 است. اما scipy همان مقادیر اصلی دامنه را حفظ میکند و نوع داده int16 است. به طور دقیق تر:

$$librosa\ wave = \frac{scipy\ wave}{2^{\#bits-1}}\ (\#bits = 16)$$

در انجام این پروژه به طور کلی از scipy استفاده شده و تنها در فاز ۲ برای ایجاد spectrogram از librosa و توابع مربوطه آن استفاده شده است.

برای پیش پردازش فایل ها ۴ کار باید انجام شود:

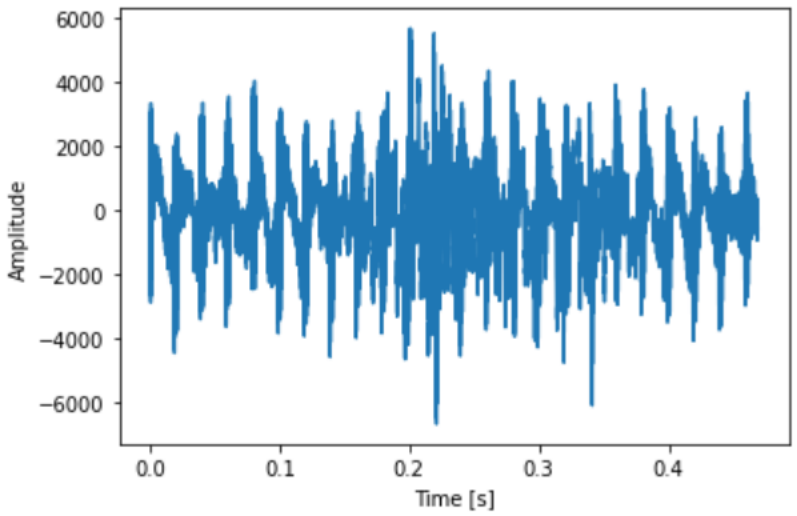
۱. حذف نویز های فرکانس بالا و پایین
۲. حذف نویز سفید
۳. حذف بخش های سکوت
۴. یکسان کردن طول نمونه ها

برای حذف نویزهای فرکانس بالا و پایین از فیلتر butterworth با فرکانس های ۵۰ و ۳۵۰۰ استفاده شد. با توجه به اینکه فرکانس نمونه برداری ۸۰۰۰ هرتز است، حداکثر فرکانس موجود ۴۰۰۰ است. پس اعمال این فیلتر bandpass فرکانس های ۰تا۵۰ و ۳۵۰۰ تا ۴۰۰۰ هرتز را حذف میکند.

برای حذف نویز سفید، الگوریتم pypi noisereduce که از روش spectral gating برای حذف نویز ثابت و غیرثابت استفاده میکند، مورد استفاده قرار گرفت. نوع نویز در دیتاست ما از نوع ثابت است. یعنی از ابتدا تا انتهای فایل صوتی بدون تغییر و توقف حضور دارد.

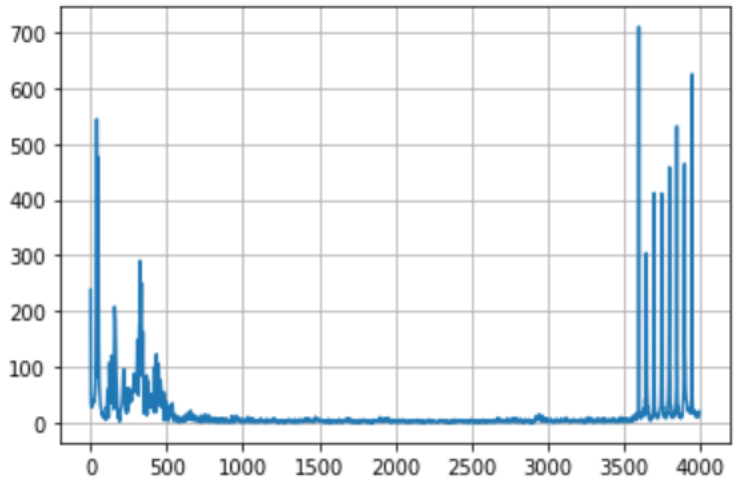
ترتیب انجام مراحل ۱ و۲ گفته شده میتواند متفاوت باشد یعنی ابتدا نویز ثابت را حذف کنیم و سپس از فیلتر میانگذر عبور دهیم. برای رسیدن به جواب بهینه و تعیین پارامتر های توابع این ۲ مرحله هر دو حالت بر روی یکی از فایل ها (فایل ۰ از پوشه ۰) بررسی و نتایج حوزه زمان و فرکانس مقایسه شدند که در ادامه آورده میشوند: (این نتایج با استفاده از scipy.io.wavfile گرفته شده اند و همانطور که واضح است دامنه بین ۱ و ۱- اسکیل نشده)

```
[ ]  time = np.linspace(0., length, audio.shape[0])
     plt.plot(time, audio)
     plt.xlabel("Time [s]")
     plt.ylabel("Amplitude")
     plt.show()
```



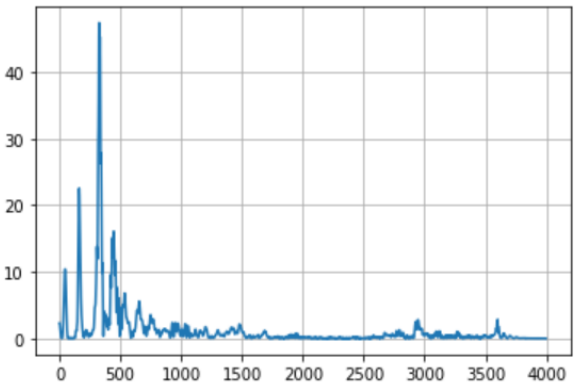
در تصویر بالا نمودار حوزه زمان سیگنال نویزی را میبینیم. واضح است که نویزی ثابت با دامنه حدودا ۳۰۰۰ از ابتدا در سیگنال وجود دارد.

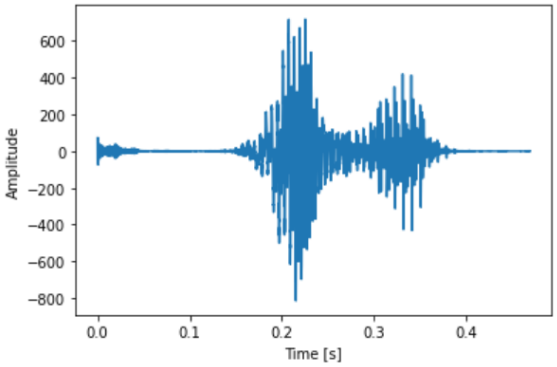
```
[ ]  N = audio.shape[0]
     T = 1.0 / rate
     x = np.linspace(0.0, N*T, N, endpoint=False)
     yf = fft(audio)
     xf = fftfreq(N, T)[:N//2]
     import matplotlib.pyplot as plt
     plt.plot(xf, 2.0/N * np.abs(yf[0:N//2]))
     plt.grid()
     plt.show()
```



در اینجا fft سیگنال را نیز میبینم. با توجه به نمودار در فرکانس های پایین و بالا نویز زیادی داریم. برای راحتی تنها سمت راست fft رسم شده. واضح است که برای مقادیر منفی فرکانس این نمودار قرینه میشود.

ابتدا سعی میکنیم با استفاده از تابع noise_reduce نویز ثابت را حذف کنیم. در مستندات این الگوریتم آورده شده است که برای پردازش سیگنال صحبت (پروژه ما) هر چقدر ورودی n_fft ،که مشخص میکند در پنجره های چه اندازه ای fft برای حذف نویز محاسبه شود، کوتاه تر انتخاب شود، دقت محاسبات و حذف نویز بهتر انجام میشود. اعداد پیشنهادی در مستندات برای فرکانس نمونه برداری پایه librosa پیشنهاد شده بودند که با توجه به فرکانس ۸۰۰۰ هرتز در پروژه ما مقدار n_fft ۱۲۸ انتخاب شد. خروجی این تابع در حوزه زمان و فرکانس را مشاهده میکنیم:





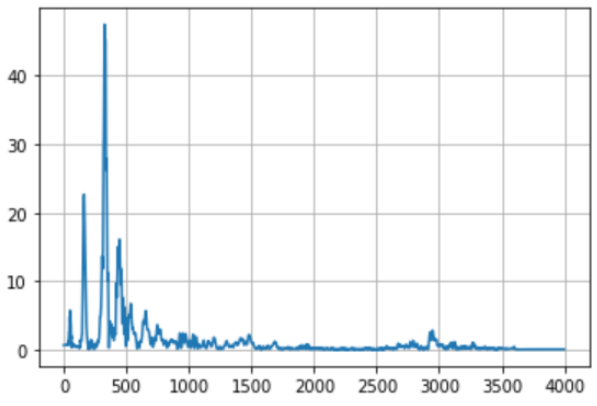
نویز ثابتی که از ابتدای فایل صوتی وجود داشت به خوبی حذف شده. البته کماکان قدری نویز در ابتدا وجود دارد که قابل اغماض است. حال فیلتر میانگذر را اعمال میکنیم:

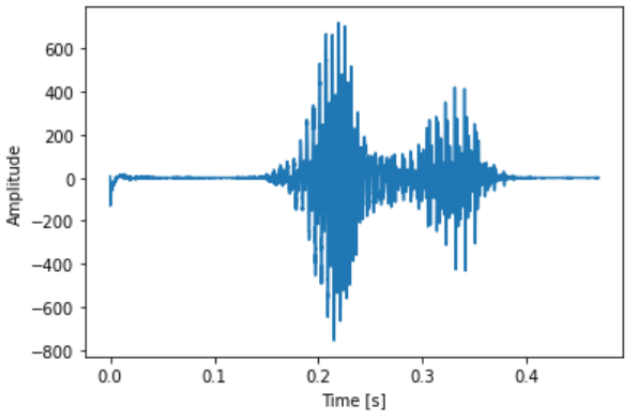
```
[ ]  sos = butter(4, [50, 3500], analog=False,fs=rate, btype='band', output='sos')
     filtered = sosfiltfilt(sos, reduced_noise)
```

در اینجا از sosfiltfit به دو دلیل استفاده شده است:

۱. فاز سیگنال را به درستی تغییر میدهد
۲. برای اردر های بالای فیلتر موجب ناپایداری خروجی نمیشود.

خروجی این فیلتر:

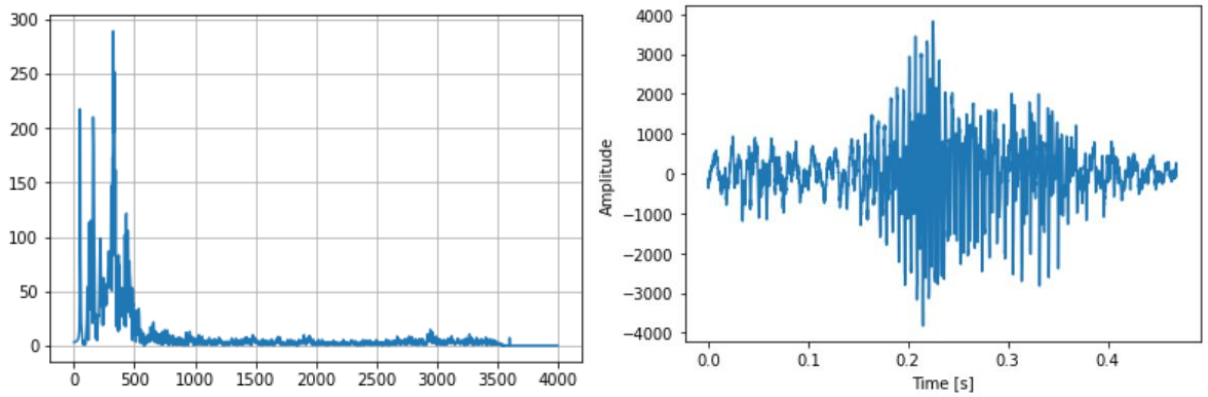




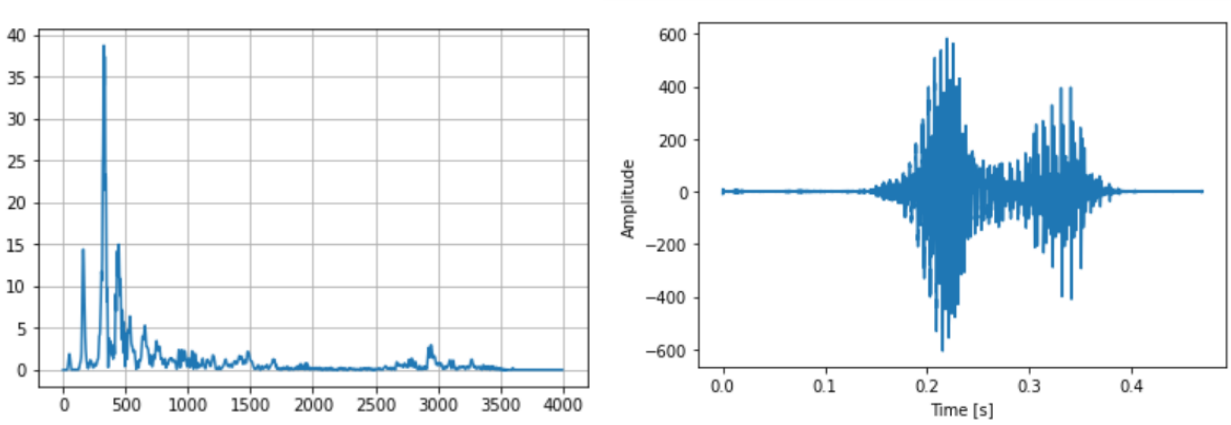
همانطور که مشاهده میشود در این حالت فیلتر میانگذر تاثیر زیادی ندارد. یعنی همان تابع noise_reduce تا حد خوبی نویز های فرکانس بالا و پایین را هم حذف کرده. نویز ابتدای سیگنال هم گرچه تا حد خوبی حذف

شده اما به دلیل پیک ناخواسته کوچکی که ایجاد شده، در مراحل بعد برای انتخاب ترشولد مناسب برای حذف بخش های سکوت کار قدری سخت میشود.

حال ترتیب اعمال فیلتر ها را عوض میکنیم. ابتدا با فیلتر میانگذر که اینبار اردر آن را تا ۱۰ بالا بردیم فرکانس های بالا و پایین را حذف میکنیم:



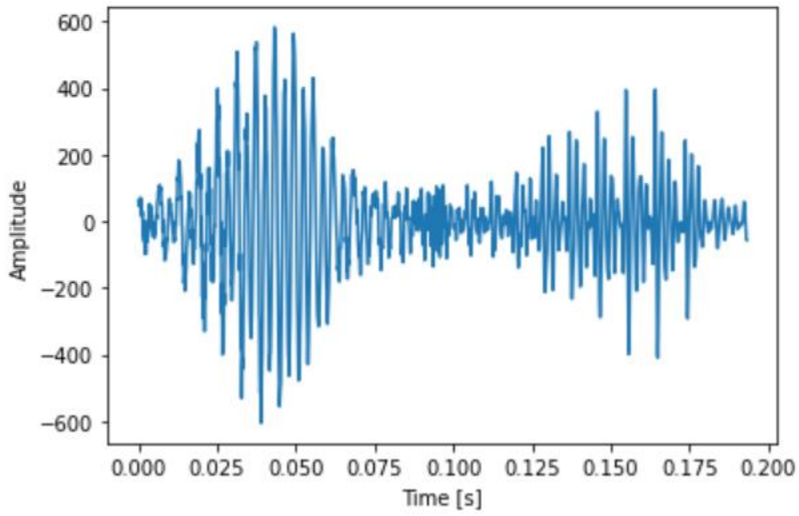
سپس نویز سفید را حذف میکنیم:



در این حالت نویز ابتدای سیگنال هم کاملا حذف شده و پیک ناخواسته ای به وجود نیامده. به همین دلیل در ادامه ابتدا فیلتر میانگذر را اعمال میکنیم و سپس نویز سفید را حذف میکنیم.

برای حذف بخش های سکوت هم میتوان از توابع آماده librosa یعنی effects.trim استفاده کرد. هم میتوان با انتخاب یک ترشولد مناسب در حوزه زمان، مقادیر کمتر از ترشولد را به عنوان بخش های سکوت از ابتدا و انتهای سیگنال حذف کرد. در ادامه نتایج روش دوم آورده شده است:

```
start = 0
end = len(reduced_noise2)

for idx, point in enumerate(reduced_noise2):
    if abs(point) > 50: #first sample which is greater that 50 is the start point
        start = idx
        break

# Reverse the array for trimming the end
for idx, point in enumerate(reduced_noise2[::-1]):
    if abs(point) > 50: #last sample which is greater that 50 is the end point
        end = len(audio) - idx
        break
final2 = reduced_noise2[start:end]

time = np.linspace(0., (end-start)/rate, final2.shape[0])
plt.plot(time, final2)
plt.xlabel("Time [s]")
plt.ylabel("Amplitude")
plt.show()
```




```
final2.shape[0]
```

```
1545
```

مراحل گفته شده برای فایل های رندم از میان داده های سایر اعداد ۱تا۹ نیز انجام شدند تا در صورت نیاز ترشولد نویز و فرکانس های فیلتر میانگذر تغییر کنند. پس از بررسی چندین فایل دیگر تغییرات زیر انجام شدند:

۱. میزان n_fft از ۱۲۸ به ۲۵۶ افزایش پیداکرد. به نظر میرسید مقدار بسیار کم آن بعضا باعث از بین رفتن اطلاعات مفید سیگنال اصلی نیز میشد.
۲. فرکانس عبور پایین فیلتر میانگذر از ۵۰ به ۷۵ افزایش یافت تا اطمینان حاصل شود فرکانس ۵۰ حتما از بین میرود.
۳. آنچه در عمل مشاهده شد این بود که ترشولد نویز ثابت با توجه به متغیر بودن دامنه سیگنال های متفاوت پاسخگو نیست و باید به دامنه خود آن سیگنال وابسته باشد. نتیجتا نقاط شروع و پایان غیر سکوت با ۰.۱ حداکثر دامنه مقایسه شدند
۴. به دلیل وجود پیک های ناخواسته در ابتدا یا انتهای سیگنال پس از فیلتر کردن، ۱۰۰ سمپل اول و ۲۰۰ سمپل آخر برای یافتن اولین و آخرین نقطه سیگنال بدون سکوت درنظر گرفته نمیشوند.
۵. عبور خروجی فیلتر میانگذر از تابع noise_reduce در بعضی موارد باعث ایجاد نویز در ابتدا یا انتهای سیگنال میشود که راه حلی برای این مورد پیدا نشد.

تمام بررسی های انجام شده در این بخش در نوت بوک Phase1_preparation در پوشه فاز ۱ آمده اند. توابع اصلی فاز ۱ مورد استفاده در نوت بوک کلی پروژه در ادامه توضیح داده میشود:

برای این فاز با توجه به نوع خواندن دیتا در فاز دوم، پیش پردازش به صورت یک تابع اصلی و ۲ تابع کمکی نوشته شده است.

۱. تابع preprocess: این تابع بدنه اصلی کد را تشکیل میدهد که در ورودی خود ۱. آدرس پوشه داده های آموزش یا تست و ۲. طول مشخص داده ها را دریافت میکند. در حالت پیش پردازش داده های آموزش، از قبل طول مشخص نیست و این ورودی صفر است اما در پیش پردازش داده های تست که باید با داده های آموزش هم طول شوند طول داده های آموزش در ورودی به تابع داده میشود. این تابع در خروجی ۱. داده های هم طول پیش پردازش شده، ۲. لیبل های آن ها و ۳. طول داده ها را تحویل میدهد. طول یکسان شده داده های آموزش بعدا به عنوان ورودی به تابع برای پیش پردازش داده های تست داده میشود.

در این تابع ابتدا فایل ها تک تک از پوشه های ۰ تا ۹ خوانده میشوند و با توجه به نام پوشه، لیبل مناسب در آرایه y ذخیره میشود.(بخش ب)

سپس نویز های سیگنال تا حد امکان گرفته میشود و همچنین بخش های سکوت نیز حذف میشوند.(بخش پ)

تا به اینجا هر سیگنال طول خاص خود را دارد و به همین جهت تمام سیگنال ها در یک لیست ذخیره میشوند.

در مرحله بعد باید حداکثر طول سیگنال ها را بدست آورد تا بقیه سیگنال ها نیز با zero-pad به همین طول برسند. البته همانطور که بالاتر گفته شد سیگنال های تست به طول سیگنال های آموزش میرسند نه ماکزیمم طول خودشان.(بخش ج)

```
[79] def preprocess(root, padd=0):
       X = [] #list to save trimmed signals with different length
       y = [] #list to assign labels according to subdirectory
       #j = 0
       #fig = plt.figure(figsize=(14,45 ))
       for i in range(1):
         path = os.path.join(root,str(i))
         for file in glob.glob(os.path.join(path,'*.wav')):
           wave = noise_reduction(file)
           trimmed = silence_trim(wave)
           X.append(trimmed.tolist())
           y.append(i)
```

```
  length = len(max(X,key=len)) #padding train data with max length
  if(padd != 0): #padding test data with maximum length of "train" data
    length = padd
  print(length)
  print()
  X_pad = np. zeros((len(X),length)) #2D array for equal size train(test) data
  for j in range(len(X)):
    audio = np.asarray(X[j]) #convert list back to numpy
    #print(audio.shape[0])
    #pad the end of signal with zeros
    audio = np.pad(audio,(0,length-audio.shape[0]),'constant',constant_values=(0,0))
    X_pad[j] = audio

  y = np.asarray(y)

  return X_pad,y, length
```

۲. تابع noise_reduction: این تابع متشکل از یک فیلتر میانگذر و تابع مورد نیاز برای حذف نویز سفید است.

```
def noise_reduction(path):
  rate, audio = wavfile.read(path)
  sos = butter(10, [75, 3500], analog=False,fs=rate, btype='band', output='sos')
  filtered = sosfiltfilt(sos, audio)
  wave = nr.reduce_noise(y = filtered, sr=rate,stationary=True,n_fft=256)
  return wave
```

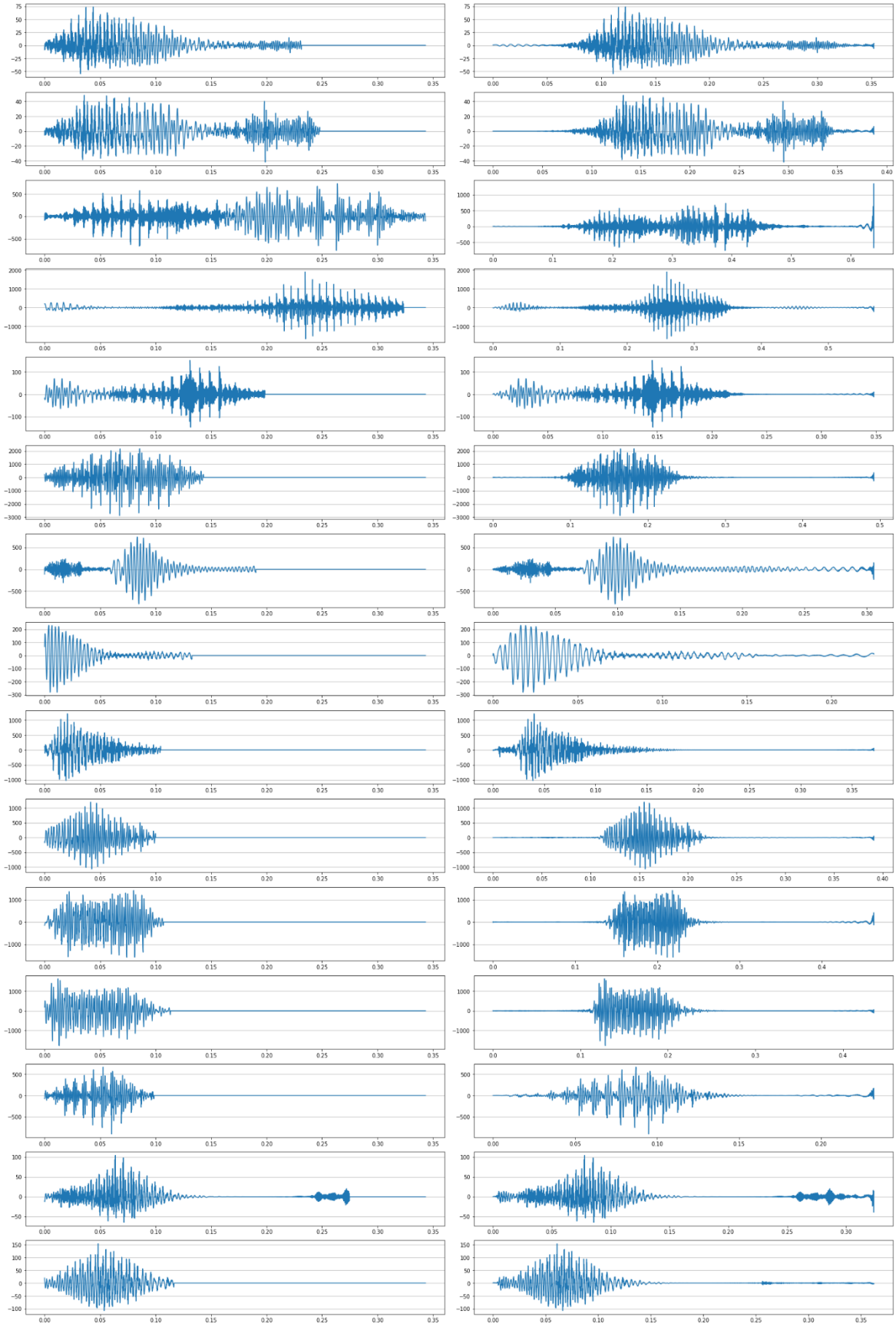
۳. تابع silence_trim: مشابه آنچه پیش تر گفته شد، این تابع با توجه به پیک سیگنال، مقادیر کمتر از ۰.۱ این مقدار را به عنوان نویز از ابتدا و انتهای سیگنال دور میریزد.

```
def silence_trim(wave):
  start = 0
  end = len(wave)
  #print(abs(max(wave[50:-50]))) #ignore unwanted picks at the end and beginning
  for idx, point in enumerate(wave):
    #first sample which is greater than 0.1*max is the start point
    if abs(point) > 0.1*abs(max(wave[50:-50])):
      if idx > 100:
        start = idx
        break

  # Reverse the array for trimming the end
  for idx, point in enumerate(wave[::-1]):
    #last sample which is greater than 0.1*max is the end point
    if abs(point) > 0.1*abs(max(wave[50:-50])):
      if idx > 200:
        end = len(wave) - idx
        break
  trimmed = wave[start:end]
  return trimmed
```

خروجی های این بخش یعنی سیگنال های پیش پردازش شده آموزش و تست به همراه لیبل های آن ها به فرمت .npy در پوشه فاز ۱ ذخیره میشوند. (سایز نهایی تمام سیگنال ها 4783 شد)

نمونه ای از خروجی های این فاز در صفحه آینده آورده شده اند. این سیگنال ها اعداد ۰و۱و۲و۷و۸ هستند که به ترتیب از هرکدام ۳ نمونه پشت هم آورده شده است.

نمودارهای سمت راست کل سیگنال پیش پردازش شده(بدون trim) و نمودارهای سمت چپ سیگنال های zero-pad شده (بعد از trim) هستند.



فاز ۲: بخش اول

الف)ابتدا داده های فاز قبل را لود میکنیم و سپس برای اینکه PCA تحت تاثیر رنج متفاوت داده ها قرار نگیرد داده ها را به میانگین ۰ و واریانس ۱ میرسانیم. (داده های تست را با همان میانگین و واریانس های داده های آموزش تغییر میدهیم)

```
[13] X_train = np.load(os.path.join(phase1_path,'phase1_Xtrain.npy'))
     y_train = np.load(os.path.join(phase1_path,'phase1_ytrain.npy'))
     print(X_train.shape)
     print(y_train.shape)
```

```
(2000, 4783)
(2000,)
```

```
[7]  X_test = np.load(os.path.join(phase1_path,'phase1_Xtest.npy'))
     y_test = np.load(os.path.join(phase1_path,'phase1_ytest.npy'))
     print(X_test.shape)
     print(y_test.shape)
```

```
(500, 4783)
(500,)
```

```
scalar = StandardScaler()
X_train_scale = scalar.fit_transform(X_train)
X_test_scale = scalar.transform(X_test)
```

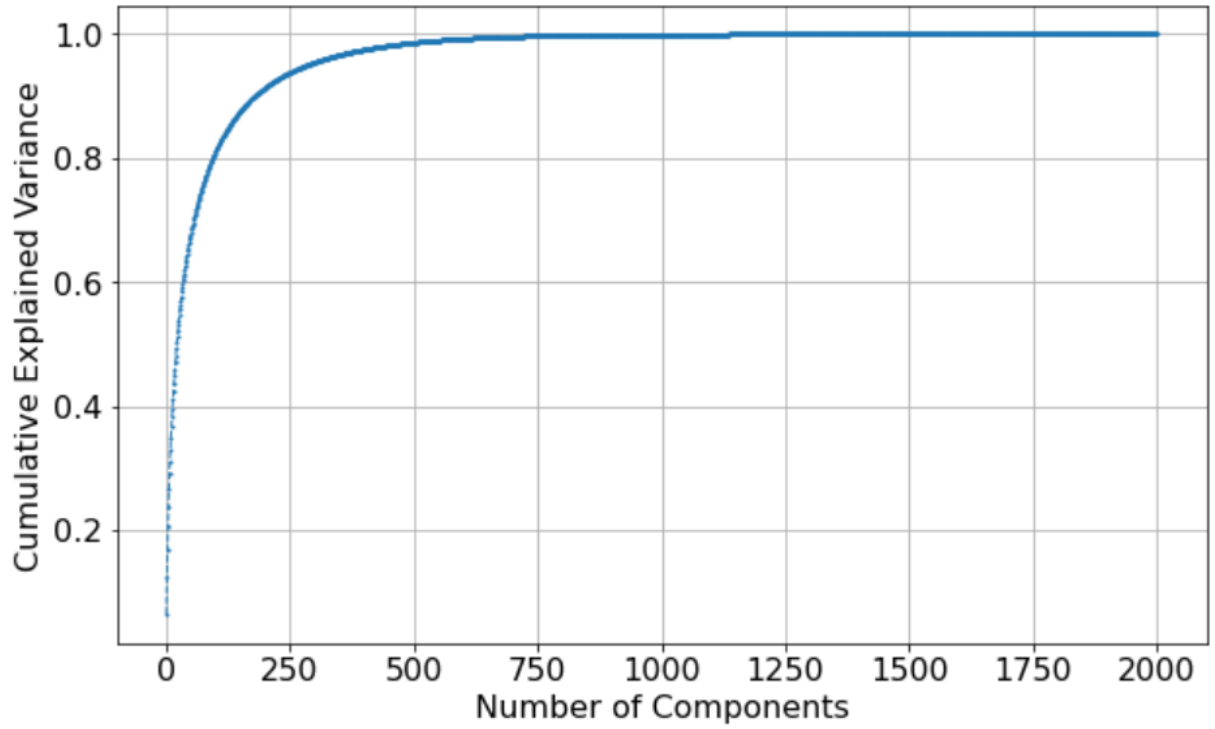
حال میخواهیم بررسی کنیم که با PCA چقدر سایز هر سمپل را میتوانیم کاهش دهیم. همانطور که در تصویر بالا مشخص است، سایز سمپل ها ۴۷۸۳ است که محاسبات را به شدت بالا میبرد. با توجه به تعداد نمونه ها که ۲۰۰۰ است با PCA حداکثر ابعاد را تا ۲۰۰۰ بعد میتوانیم کاهش دهیم. با استفاده از explained_variance_ratio میخواهیم ببینیم که هر کدام از این ابعاد چه واریانسی نسبت به هم دارند. یعنی هر کدام چقدر اطلاعات مفید در خود دارند. جمع تمام این مقادیر ۱ میشود، پس میتوان مثلا اولین نقطه ای که جمع ۰.۹۹ میشود یعنی ۹۹٪ اطلاعات حفظ شده را به عنوان تعداد بعد بهینه انتخاب کنیم.

```
pca = PCA() #n_components = n_samples = 2000
pca.fit(X_train_scale);
evr = pca.explained_variance_ratio_ #how much information each feature hold compared to others
```

```
print(evr.cumsum()[0:2000:50])
print(evr.shape)
```

```
[0.13719439 0.67818515 0.80013019 0.86458341 0.90434454 0.93085616
 0.94914157 0.96219883 0.97174105 0.97883095 0.98410242 0.98805202
 0.99104354 0.99330342 0.99502232 0.99633506 0.99733359 0.99808029
 0.99862994 0.99903372 0.99932738 0.99954077 0.99968975 0.999794
 0.99986508 0.99991269 0.99994439 0.99996577 0.99997954 0.99998819
 0.99999348 0.99999659 0.99999834 0.99999925 0.9999997  0.99999989
 0.99999997 0.99999999 1.         1.        ]
(2000,)
```

همانطور که در تصویر بالا آمده بعد ۲۰۰ اولین جایی است که به مقدار بالای ۰.۹ رسیده ایم و در نهایت با ۶۰۰ بعد به عدد بالای ۰.۹۹ میرسیم. برای اینکه درک بهتری داشته باشیم، نمودار تجمعی را هم بررسی میکنیم:



با توجه به اینکه ۶۰۰ کماکان ابعاد بالایی است، برای کاهش محاسبات ابعاد را به ۳۰۰ میرسانیم.

```
pca = PCA(n_components=300)
X_train_pca = pca.fit_transform(X_train)
X_test_pca = pca.transform(X_test)
print(X_train_pca.shape)
```

```
(2000, 300)
```

برای طبقه بندی چند کلاسه از روش های متفاوتی میتوان استفاده کرد:

- k-Nearest Neighbors.
- Decision Trees.
- Naive Bayes.
- Random Forest.
- Gradient Boosting.
- Logistic Regression.
- Support Vector Machine.

در این قسمت از random forest که طبقه بند قوی میباشد استفاده میکنیم. برای یافتن هایپر پارامترهای مناسب از RandomizedSearchCV استفاده میکنیم که cross validation نیز به همراه دارد. از این جهت از GridSearchCV استفاده نمیکنیم که تعداد هایپر پارامترها زیاد هستند و با توجه به سایز سمپل ها بررسی تک تک حالات زمان بسیار زیادی میبرد.

```
param_grid = {
    'max_depth': [20, 30, 40, 50, 60],
    'min_samples_leaf': [3, 4, 5, 6],
    'min_samples_split': [8, 10, 12],
    'n_estimators': [300, 400, 500],
    'max_features': [70, 90, 110, 130, 150]
}

rf = RandomForestClassifier()

clf = RandomizedSearchCV(estimator = rf, param_distributions = param_grid,
                         cv = 5, scoring='accuracy', n_iter=50, n_jobs = -1,
                   verbose=2, random_state=0,return_train_score=True)
clf.fit(X_train_pca, y_train)
pd.DataFrame({'parameters': clf.cv_results_["params"],
              'validation_score': clf.cv_results_["mean_test_score"],
              'train_score': clf.cv_results_["mean_train_score"],
              'rank': clf.cv_results_["rank_test_score"]})
```

ب)در جدول زیر پارامترهای مدل های آموزش داده شده و خطای آموزش و ارزیابی هر کدام آورده شده است. آنچه واضح است این است که مدل به شدت overfit شده است. دلیل آن تعداد کم نمونه ها برای یادگیری هر کلاس و تعداد زیاد ویژگی ها است. اگر قرار باشد قاعده ۱ به ۱۰ رعایت شود، برای یادگیری = 2000*0.8/10 160 نمونه هر کلاس، حداکثر باید ۱۶ ویژگی در آموزش دخیل باشد که با توجه به سایز اولیه داده ها یعنی ۴۷۸۳ اطلاعات بسیار زیادی دور ریخته میشود.

parameters	validation_score	train_score	rank
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 130, 'max_depth': 40}	0.3055	0.992125	41
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 6, 'max_features': 130, 'max_depth': 20}	0.2965	0.9905	49
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 4, 'max_features': 90, 'max_depth': 40}	0.3205	0.996	6
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 70, 'max_depth': 20}	0.3105	0.99075	29
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 70, 'max_depth': 50}	0.314	0.990625	19
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 4, 'max_features': 90, 'max_depth': 50}	0.316	0.994125	11
{'n_estimators': 300, 'min_samples_split': 8, 'min_samples_leaf': 4, 'max_features': 150, 'max_depth': 60}	0.31	0.997375	30
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 3, 'max_features': 90, 'max_depth': 40}	0.306	0.997125	39
{'n_estimators': 400, 'min_samples_split': 8, 'min_samples_leaf': 4, 'max_features': 90, 'max_depth': 40}	0.324	0.99775	3
{'n_estimators': 500, 'min_samples_split': 8, 'min_samples_leaf': 5, 'max_features': 110, 'max_depth': 30}	0.325	0.9945	1
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 150, 'max_depth': 50}	0.316	0.99575	11
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 90, 'max_depth': 20}	0.306	0.992125	37
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 130, 'max_depth': 30}	0.3055	0.99275	41
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 110, 'max_depth': 50}	0.293	0.991875	50
{'n_estimators': 500, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 130, 'max_depth': 40}	0.3055	0.9905	41
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 5, 'max_features': 150, 'max_depth': 30}	0.301	0.994125	47
{'n_estimators': 300, 'min_samples_split': 8, 'min_samples_leaf': 6, 'max_features': 150, 'max_depth': 30}	0.3045	0.99125	44
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 110, 'max_depth': 50}	0.313	0.99125	21
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 4, 'max_features': 90, 'max_depth': 60}	0.313	0.993625	21
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 130, 'max_depth': 60}	0.3165	0.99325	10
{'n_estimators': 500, 'min_samples_split': 10, 'min_samples_leaf': 4, 'max_features': 70, 'max_depth': 20}	0.3075	0.99575	34
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 3, 'max_features': 90, 'max_depth': 60}	0.316	0.99725	11
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 150, 'max_depth': 50}	0.3145	0.9955	18
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 90, 'max_depth': 60}	0.3155	0.992875	16
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 5, 'max_features': 150, 'max_depth': 40}	0.311	0.994	28
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 110, 'max_depth': 20}	0.304	0.990875	45
{'n_estimators': 500, 'min_samples_split': 10, 'min_samples_leaf': 4, 'max_features': 150, 'max_depth': 20}	0.308	0.9955	32
{'n_estimators': 400, 'min_samples_split': 8, 'min_samples_leaf': 3, 'max_features': 70, 'max_depth': 60}	0.325	0.9985	1
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 110, 'max_depth': 40}	0.307	0.99225	35
{'n_estimators': 300, 'min_samples_split': 8, 'min_samples_leaf': 4, 'max_features': 70, 'max_depth': 50}	0.3155	0.997625	16
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 150, 'max_depth': 20}	0.316	0.99525	11
{'n_estimators': 300, 'min_samples_split': 8, 'min_samples_leaf': 6, 'max_features': 70, 'max_depth': 50}	0.313	0.991	21
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 6, 'max_features': 150, 'max_depth': 50}	0.3005	0.9905	48
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 90, 'max_depth': 40}	0.309	0.99525	31
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 130, 'max_depth': 50}	0.312	0.995375	26
{'n_estimators': 500, 'min_samples_split': 8, 'min_samples_leaf': 3, 'max_features': 150, 'max_depth': 60}	0.322	0.99825	5
{'n_estimators': 500, 'min_samples_split': 8, 'min_samples_leaf': 3, 'max_features': 70, 'max_depth': 40}	0.3235	0.99825	4
{'n_estimators': 500, 'min_samples_split': 8, 'min_samples_leaf': 6, 'max_features': 70, 'max_depth': 40}	0.3195	0.99075	7
{'n_estimators': 500, 'min_samples_split': 8, 'min_samples_leaf': 3, 'max_features': 130, 'max_depth': 60}	0.314	0.998625	20
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 5, 'max_features': 110, 'max_depth': 40}	0.3115	0.994125	27
{'n_estimators': 400, 'min_samples_split': 8, 'min_samples_leaf': 6, 'max_features': 130, 'max_depth': 50}	0.306	0.991125	39
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 4, 'max_features': 90, 'max_depth': 30}	0.3025	0.993875	46
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 110, 'max_depth': 20}	0.306	0.992375	37
{'n_estimators': 500, 'min_samples_split': 10, 'min_samples_leaf': 5, 'max_features': 130, 'max_depth': 40}	0.313	0.994125	21
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 110, 'max_depth': 50}	0.307	0.995625	35
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 70, 'max_depth': 50}	0.317	0.996	9
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 70, 'max_depth': 30}	0.312	0.992875	25
{'n_estimators': 400, 'min_samples_split': 8, 'min_samples_leaf': 5, 'max_features': 90, 'max_depth': 20}	0.316	0.994625	11
{'n_estimators': 500, 'min_samples_split': 8, 'min_samples_leaf': 6, 'max_features': 90, 'max_depth': 20}	0.308	0.991	32
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 4, 'max_features': 70, 'max_depth': 60}	0.3185	0.9945	8

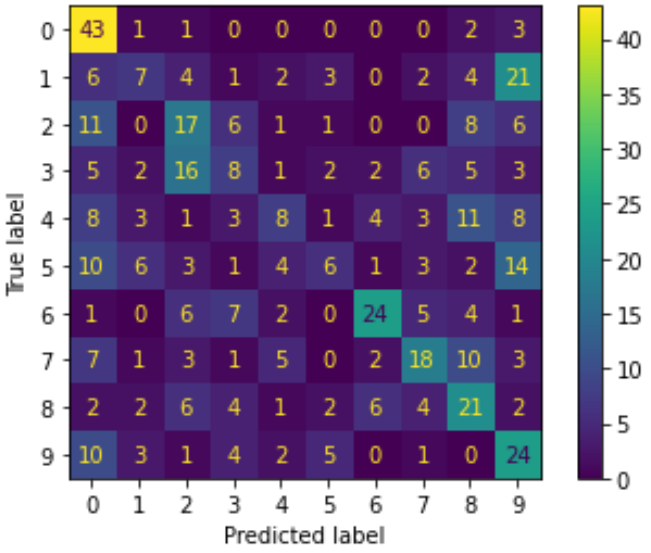
در جدول صفحه قبل ۲ حالتی که بیشترین دقت روی داده های ارزیابی را داشتند با رنگ زرد مشخص شده اند.

```
print(clf.best_params_)
y_pred = clf.predict(X_test_pca)
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test, y_pred))
```

```
{'n_estimators': 500, 'min_samples_split': 8, 'min_samples_leaf': 5, 'max_features': 110, 'max_depth': 30}

0.352

              precision    recall  f1-score   support

           0       0.42      0.86      0.56        50
           1       0.28      0.14      0.19        50
           2       0.29      0.34      0.31        50
           3       0.23      0.16      0.19        50
           4       0.31      0.16      0.21        50
           5       0.30      0.12      0.17        50
           6       0.62      0.48      0.54        50
           7       0.43      0.36      0.39        50
           8       0.31      0.42      0.36        50
           9       0.28      0.48      0.36        50

    accuracy                           0.35       500
   macro avg       0.35      0.35      0.33       500
weighted avg       0.35      0.35      0.33       500
```

Train score	Validation score	Test score
99.45%	32.5%	35.2%

در جدول بالا، دقت بهترین مدل روی هر سه نوع داده آموزش، ارزیابی و تست مشاهده میشود. همچنین confusion matrix این طبقه بند نیز به صورت زیر است. کلاس صفر بیشترین دقت را دارد و کلاس های ۱و۳و۴و۵ کمترین میزان تشخیص درست را دارند.



به منظور بررسی دقیق تر، یکبار دیگر این مراحل با ۱۰۰ ویژگی نیز تکرار شدند.(در نمودار cumulative explained variance مقدار نقطه ۱۰۰، ۰.۸ بود) اینبار min_samples_split روی ۸ ثابت شد. (در جدول اکثر دقت های بالا با این مقدار به دست آمده اند) آنچه مشاهده شد این بود که دقت به ۳۰٪ کاهش پیدا کرد و مدل همچنان overfit بود. به همین دلیل نتایج این بررسی مجدد در گزارش و فایل نوت بوک آورده نشده اند.

پ) تابع نوشته شده در این قسمت آدرس پوشه تست، آدرس پوشه مدل و آدرس پوشه فاز ۱ که فایل های پیش پردازش شده و لیبل های آموزش در آن ذخیره شده اند را میگیرد. همچنین ابعاد انتخاب شده از PCA و طول داده ها که از پیش پردازش داده های آموزش در بخش های قبل بدست آوردیم به عنوان ورودی به تابع داده میشوند.

```
    root = os.getcwd()
    phase1_path = os.path.join(root,'Phase_1')
    phase2a_path = os.path.join(root,'Phase_2a')
    test_path = os.path.join(dataset_path,'test')

    length = 4783
    n_pca = 300

    acc = predict_phase2(test_path,phase1_path, phase2a_path, length, n_pca)
def predict_phase2(test_path, phase1_path, phase2a_path, len_train, n_pca):
  X_train = np.load(os.path.join(phase1_path,'phase1_Xtrain.npy'))
  y_train = np.load(os.path.join(phase1_path,'phase1_ytrain.npy'))

  X_test,y_test,len_test = preprocess(test_path,len_train)
  print(X_test.shape)

  scalar = StandardScaler()
  X_train_scale = scalar.fit_transform(X_train)
  X_test_scale = scalar.transform(X_test)

  pca = PCA(n_components=n_pca)
  X_train_pca = pca.fit_transform(X_train_scale)
  X_test_pca = pca.transform(X_test_scale)

  model_path = os.path.join(phase2a_path,'Model_Phase2a.pkl')
  with open(model_path, 'rb') as f:
    clf = pickle.load(f)

  y_pred = clf.predict(X_test_pca)
  np.save(os.path.join(phase2a_path,'phase2a_predicted.npy'),y_pred)


  return accuracy_score(y_test,y_pred)
```

سپس در این تابع ابتدا لیبل ها و داده های آموزش از پیش پردازش شده را فرا میخوانیم. داده های تست را با دادن آدرس پوشه تست و طول مورد نظر به تابع preprocess آماده میکنیم. سپس داده ها را نرمال میکنیم و با PCA کاهش بعد میدهیم. در نهایت مدلی که در بخش قبل آموزش دادیم را فراخوانی میکنیم و دقت روی داده های تست را گزارش میکنیم.

```
print("accuracy on test data: ", acc)
```

```
4783

(500, 4783)
accuracy on test data:  0.36
```

فاز ۲: بخش دوم

الف) spectrogram نمایش طیف فرکانس های موجود در یک سیگنال در زمان های متفاوت است که معمولا به صورت یک heat map نمایش داده میشود. محور x محور زمان سیگنال است که به بازه های مساوی تقسیم شده و محور y محور فرکانس است که رنج فرکانس های موجود در تبدیل فوریه سیگنال را به صورت تقسیم شده به بازه های مساوی نشان میدهد. ایجاد یک spectrogram در حوزه دیجیتال به روش windowing انجام میشود. یعنی به زبان ساده سیگنال نمونه برداری شده دیجیتال به بازه های مساوی با هم پوشانی تقسیم میشود، در هر بازه تبدیل فوریه حساب میشود و طیف فرکانس های موجود در این بازه زمانی به دست می آید که نماینده یک ستون در heat map نهایی به مرکز بازه است. به بیان دقیق تر در هر بازه به مرکز زمان t و طول w، spectrogram از محاسبه short-time Fourier transform(STFT) به دست می آید.

$$\text{spectrogram}(t,\omega) = |\text{STFT}(t,\omega)|^2$$

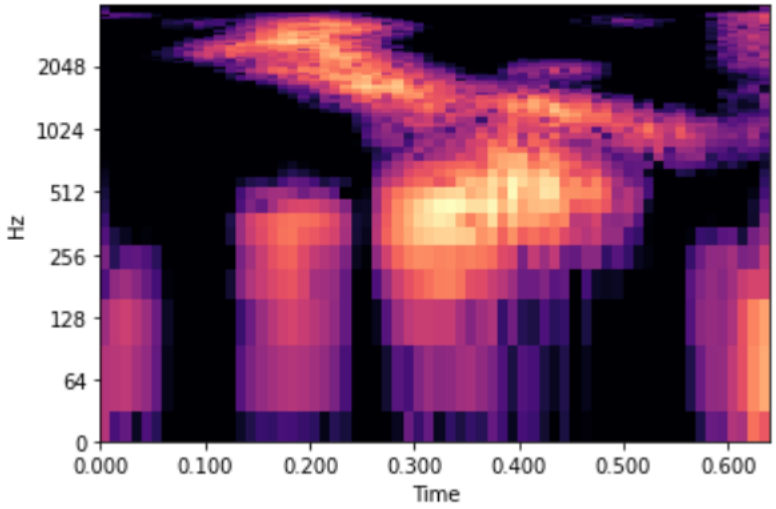
$$\mathbf{STFT}\{x[n]\}(m,\omega) \equiv X(m,\omega) = \sum_{n=-\infty}^{\infty} x[n]w[n-m]e^{-j\omega n}$$

پنجره ای که در بازه به طول w در سیگنال اصلی ضرب میشود، میتواند متفاوت باشد. یکی از پنجره های متداول استفاده از یک توزیع گاوسی شیفت یافته به اندازه m است که به تبدیل گابور نیز مشهور است.(نوع خاصی از STFT)

ب)برای محاسبه spectrogram در این پروژه از توابع STFT و magnitude_to_dB پکیج librosa استفاده میشود. تعداد سطر ها در STFT از رابطه 1+n_ftt//2 به دست می آید. پس برای اینکه به سایز ۶۴ طبق صورت پروژه برسیم n_fft برابر ۱۲۷ درنظر گرفته شده است. همچنین تعداد ستون ها به تعداد بازه های که سیگنال را در حوزه زمان به آن تقسیم میکنیم بستگی دارد پس برای رسیدن به تعداد ۶۴ بازه، میزان پرش میان هر دو بازه متوالی یعنی hop_length را برابر 1+audio.shape[0]//64 قرار میدهیم. در مورد تابع magnitude_to_db هم باید گفت این تابع بیشترین مقدار STFT را 0 dB نظر میگیرد و بقیه مقادیر نسبت به آن سنجیده میشوند. همچنین مقادیری که 80dB (قابل تنظیم) کمتر از مقدار ماکزیمم باشند، به 80 dB- ست میشوند. نمونه spectrogram یکی از فایل ها پس از فیلتر کردن به صورت زیر است:

```
S2 = np.abs(librosa.stft(wave2, n_fft=127, hop_length=(wave2.shape[0]//64) + 1))
stft2 = librosa.amplitude_to_db(S2, ref=np.max)
librosa.display.specshow(stft2, x_axis='time', y_axis='log', sr=rate,
                         hop_length=(wave2.shape[0]//64) + 1)
```

```
<matplotlib.collections.QuadMesh at 0x7fb6b7d3ad10>
```



ابتدا یک آرایه ۳ بعدی برای ذخیره سازی STFT های داده های آموزش و تست ایجاد میکنیم. سپس برای هر کدام از سیگنال ها به طریقی که بالاتر گفته شد STFT را حساب میکنیم.

```
X_spec_train = np.zeros([X_train.shape[0],64,64])
X_spec_test = np.zeros([X_test.shape[0],64,64])
print(X_spec_train.shape)
print(X_spec_test.shape)
```

```
(2000, 64, 64)
(500, 64, 64)
```

```
#calculate STFT in dBs for train and test data
for i in range(X_train.shape[0]):
  S = np.abs(librosa.stft(X_train[i], n_fft=127, hop_length=(X_train.shape[1]//64)+1))
  stft = librosa.amplitude_to_db(S, ref=np.max)
  X_spec_train[i] = stft

for i in range(X_test.shape[0]):
  S = np.abs(librosa.stft(X_test[i], n_fft=127, hop_length=(X_test.shape[1]//64)+1))
  stft = librosa.amplitude_to_db(S, ref=np.max)
  X_spec_test[i] = stft

print(X_spec_train.shape)
print(X_spec_test.shape)
```

```
(2000, 64, 64)
(500, 64, 64)
```

پ)برای تبدیل آرایه های ۲ بعدی به تک بعدی هر STFT از reshape استفاده میکنیم. برای اینکه داده ها ستون به ستون (به جای سطر به سطر) کنار هم قرار بگیرند از order='F' استفاده میکنیم.

```
X_spec_train = X_spec_train.reshape(X_train.shape[0],-1,order='F')
X_spec_test = X_spec_test.reshape(X_test.shape[0],-1,order='F')
print(X_spec_train.shape)
print(X_spec_test.shape)
```

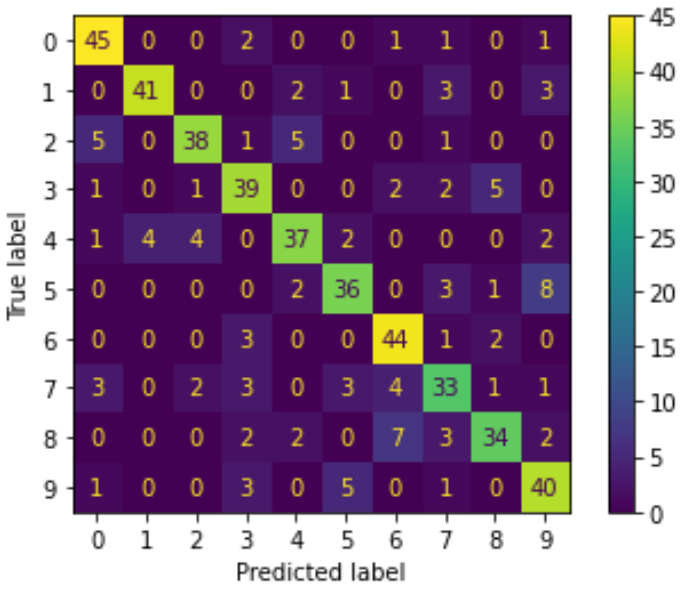
```
(2000, 4096)
(500, 4096)
```

ت)روند کار کاملا مشابه بخش ب قسمت a میباشد. یعنی داده ها را نرمال میکنیم، با PCA به ۳۰۰ بعد میرسانیم و بر روی همان گرید قبلی به دنبال پارامترهای بهینه میگردیم. نتایج این قسمت در ادامه قابل مشاهده هستند.

parameters	validation_score	train_score	rank
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 130, 'max_depth': 40}	0.771	0.98525	15
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 6, 'max_features': 130, 'max_depth': 20}	0.7675	0.979875	18
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 4, 'max_features': 90, 'max_depth': 40}	0.7825	0.994875	5
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 70, 'max_depth': 20}	0.7805	0.9835	6
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 70, 'max_depth': 50}	0.7865	0.9835	2
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 4, 'max_features': 90, 'max_depth': 50}	0.78	0.991875	7
{'n_estimators': 300, 'min_samples_split': 8, 'min_samples_leaf': 4, 'max_features': 150, 'max_depth': 60}	0.7765	0.9955	11
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 3, 'max_features': 90, 'max_depth': 40}	0.7885	0.9965	1
{'n_estimators': 400, 'min_samples_split': 8, 'min_samples_leaf': 4, 'max_features': 90, 'max_depth': 40}	0.784	0.997	4
{'n_estimators': 500, 'min_samples_split': 8, 'min_samples_leaf': 5, 'max_features': 110, 'max_depth': 30}	0.7795	0.989875	8
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 150, 'max_depth': 50}	0.7745	0.992875	12
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 90, 'max_depth': 20}	0.7845	0.986	3
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 130, 'max_depth': 30}	0.772	0.985875	13
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 110, 'max_depth': 50}	0.765	0.97975	20
{'n_estimators': 500, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 130, 'max_depth': 40}	0.7695	0.9805	16
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 5, 'max_features': 150, 'max_depth': 30}	0.7695	0.98925	16
{'n_estimators': 300, 'min_samples_split': 8, 'min_samples_leaf': 6, 'max_features': 150, 'max_depth': 30}	0.7675	0.978	18
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 110, 'max_depth': 50}	0.7775	0.980625	10
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 4, 'max_features': 90, 'max_depth': 60}	0.7795	0.99125	8
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 130, 'max_depth': 60}	0.7715	0.985625	14

```
print(clf.best_params_)
y_pred = clf.predict(X_spec_test_pca)
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test, y_pred))
```

```
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 3, 'max_features': 90, 'max_depth': 40}
0.774
              precision    recall  f1-score   support

           0       0.80      0.90      0.85        50
           1       0.91      0.82      0.86        50
           2       0.84      0.76      0.80        50
           3       0.74      0.78      0.76        50
           4       0.77      0.74      0.76        50
           5       0.77      0.72      0.74        50
           6       0.76      0.88      0.81        50
           7       0.69      0.66      0.67        50
           8       0.79      0.68      0.73        50
           9       0.70      0.80      0.75        50

    accuracy                           0.77       500
   macro avg       0.78      0.77      0.77       500
weighted avg       0.78      0.77      0.77       500
```



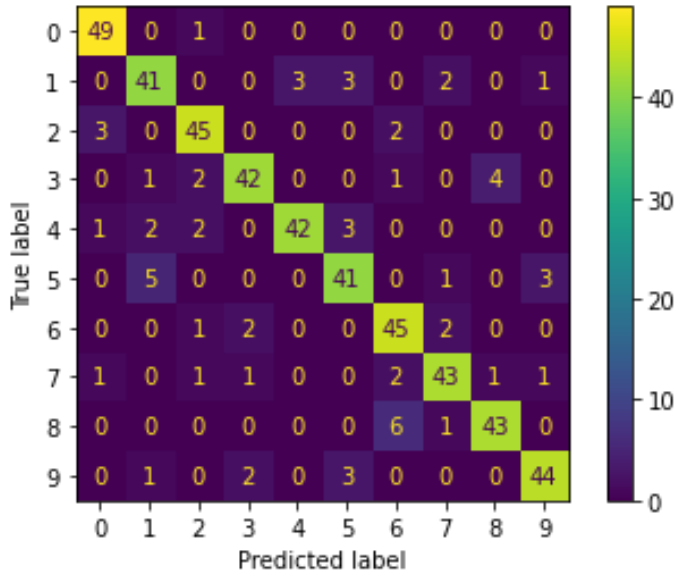
Train score	Validation score	Test score
99.65%	78.85%	77.4%

با توجه به اینکه داده های ما در این قسمت نسبت به قسمت a اطلاعات مفیدتری دارند، همین گرید بدون PCA و با تمام ۴۰۹۶ ویژگی نیز بررسی شد که نتایج آن در ادامه آمده است:

parameters	validation_score	train_score	rank
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 3, 'max_features': 200, 'max_depth': 80}	0.8805	0.9925	1
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 6, 'max_features': 200, 'max_depth': 80}	0.8705	0.979	7
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 1000, 'max_depth': 50}	0.876	0.991125	3
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 1000, 'max_depth': 40}	0.869	0.981125	11
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 800, 'max_depth': 50}	0.8635	0.9815	20
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 400, 'max_depth': 80}	0.8725	0.984375	5
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 6, 'max_features': 1000, 'max_depth': 80}	0.864	0.982125	18
{'n_estimators': 500, 'min_samples_split': 10, 'min_samples_leaf': 5, 'max_features': 400, 'max_depth': 30}	0.8695	0.987125	10
{'n_estimators': 500, 'min_samples_split': 10, 'min_samples_leaf': 5, 'max_features': 800, 'max_depth': 40}	0.873	0.986625	4
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 6, 'max_features': 600, 'max_depth': 40}	0.8645	0.98125	17
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 800, 'max_depth': 60}	0.868	0.981	12
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 400, 'max_depth': 50}	0.868	0.983375	12
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 6, 'max_features': 800, 'max_depth': 40}	0.8635	0.98175	19
{'n_estimators': 300, 'min_samples_split': 12, 'min_samples_leaf': 3, 'max_features': 800, 'max_depth': 80}	0.88	0.99	2
{'n_estimators': 500, 'min_samples_split': 8, 'min_samples_leaf': 6, 'max_features': 600, 'max_depth': 50}	0.865	0.982125	15
{'n_estimators': 400, 'min_samples_split': 8, 'min_samples_leaf': 5, 'max_features': 1000, 'max_depth': 40}	0.8705	0.986375	8
{'n_estimators': 400, 'min_samples_split': 8, 'min_samples_leaf': 6, 'max_features': 1000, 'max_depth': 30}	0.8645	0.98225	16
{'n_estimators': 400, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 800, 'max_depth': 40}	0.871	0.984875	6
{'n_estimators': 300, 'min_samples_split': 10, 'min_samples_leaf': 6, 'max_features': 600, 'max_depth': 40}	0.868	0.982	12
{'n_estimators': 500, 'min_samples_split': 12, 'min_samples_leaf': 5, 'max_features': 1000, 'max_depth': 40}	0.87	0.9855	9

```
print(clf2.best_params_)
y_pred = clf2.predict(X_spec_test_scale)
print(accuracy_score(y_test,y_pred))
print(classification_report(y_test, y_pred))
```

```
{'n_estimators': 400, 'min_samples_split': 10, 'min_samples_leaf': 3, 'max_features': 200, 'max_depth': 80}
0.87
              precision    recall  f1-score   support

           0       0.91      0.98      0.94        50
           1       0.82      0.82      0.82        50
           2       0.87      0.90      0.88        50
           3       0.89      0.84      0.87        50
           4       0.93      0.84      0.88        50
           5       0.82      0.82      0.82        50
           6       0.80      0.90      0.85        50
           7       0.88      0.86      0.87        50
           8       0.90      0.86      0.88        50
           9       0.90      0.88      0.89        50

    accuracy                           0.87       500
   macro avg       0.87      0.87      0.87       500
weighted avg       0.87      0.87      0.87       500
```

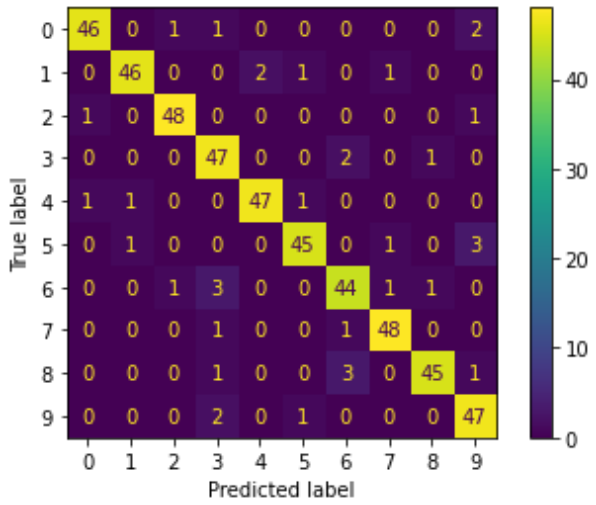


Train score	Validation score	Test score
99.25%	88.05%	87%

همانطور که مشاهده میشود با استفاده از تمام ویژگی ها به دقت ۱۰ درصد بالاتر نسبت به حالت با PCA رسیدیم.

در این قسمت طبقه بند SVM نیز بررسی شد که نسبت به Random Forest در این قسمت نتایج آن قدری بهتر شد:

parameters	validation_scor	train_scor	ran
{'C': 1, 'gamma': 0.1}	0.135	1	25
{'C': 1, 'gamma': 0.01}	0.4	1	20
{'C': 1, 'gamma': 0.001}	0.8205	0.979375	12
{'C': 1, 'gamma': 0.0001}	0.8075	0.857125	14
{'C': 1, 'gamma': 1e-05}	0.545	0.568375	15
{'C': 10, 'gamma': 0.1}	0.1455	1	21
{'C': 10, 'gamma': 0.01}	0.4285	1	16
{'C': 10, 'gamma': 0.001}	0.848	1	8
{'C': 10, 'gamma': 0.0001}	0.901	0.976375	6
{'C': 10, 'gamma': 1e-05}	0.8165	0.857625	13
{'C': 100, 'gamma': 0.1}	0.1455	1	21
{'C': 100, 'gamma': 0.01}	0.4285	1	16
{'C': 100, 'gamma': 0.001}	0.848	1	8
{'C': 100, 'gamma': 0.0001}	0.913	0.999375	1
{'C': 100, 'gamma': 1e-05}	0.895	0.968875	7
{'C': 500, 'gamma': 0.1}	0.1455	1	21
{'C': 500, 'gamma': 0.01}	0.4285	1	16
{'C': 500, 'gamma': 0.001}	0.848	1	8
{'C': 500, 'gamma': 0.0001}	0.911	1	3
{'C': 500, 'gamma': 1e-05}	0.9125	0.99775	2
{'C': 1000, 'gamma': 0.1}	0.1455	1	21
{'C': 1000, 'gamma': 0.01}	0.4285	1	16
{'C': 1000, 'gamma': 0.001}	0.848	1	8
{'C': 1000, 'gamma': 0.0001}	0.911	1	3
{'C': 1000, 'gamma': 1e-05}	0.911	0.999125	3




```
{'C': 100, 'gamma': 0.0001}
0.926
              precision    recall  f1-score   support

           0       0.96      0.92      0.94        50
           1       0.96      0.92      0.94        50
           2       0.96      0.96      0.96        50
           3       0.85      0.94      0.90        50
           4       0.96      0.94      0.95        50
           5       0.94      0.90      0.92        50
           6       0.88      0.88      0.88        50
           7       0.94      0.96      0.95        50
           8       0.96      0.90      0.93        50
           9       0.87      0.94      0.90        50

    accuracy                           0.93       500
   macro avg       0.93      0.93      0.93       500
weighted avg       0.93      0.93      0.93       500
```

Train score	Validation score	Test score
99.93%	91.3%	92.6%

در نهایت مدل SVM به عنوان مدل نهایی قسمت ت ذخیره شد.

ث)تابع این قسمت کاملا مشابه قسمت a نوشته شده است. تنها تفاوتی که وجود دارد این است که محاسبات STFT داده های تست به آن اضافه شده و قسمت کاهش بعد با PCA نیز حذف شده. خروجی این تابع:

```
4783

(500, 64, 64)
(500, 4096)
accuracy on test data:  0.926
```

با آنچه در قسمت قبل گرفتیم تطابق دارد.

ج)آنچه واضح است این است که پس از استخراج اطلاعات مفید از داده ها یعنی اطلاعات ارتباط حوزه زمان و فرکانس با یکدیگر، نتایج به شدت بهبود پیدا کرده اند. در قسمت اول فاز ۲ با وجود پیش پردازش هایی که انجام شد یعنی حذف نویز و حذف سکوت، باز هم نویز در حوزه زمان وجود داشت و اطلاعات دامنه صرفا نتایج خوبی به همراه نداشت چون آنچه در عمل دیدیم این بود که در حوزه فرکانس هم اطلاعات بسیار مفیدی وجود دارد.

نتایج قبل از استخراج ویژگی:

Train score	Validation score	Test score
99.45%	32.5%	35.2%

نتایج بعد از استخراج ویژگی:

Train score	Validation score	Test score
99.93%	91.3%	92.6%

فاز۳:

الف)در این قسمت ابتدا با تولید اعداد رندم در بازه ۰تا۲۰۰، از نمونه های STFT هر کلاس ۵ نمونه انتخاب شدند:

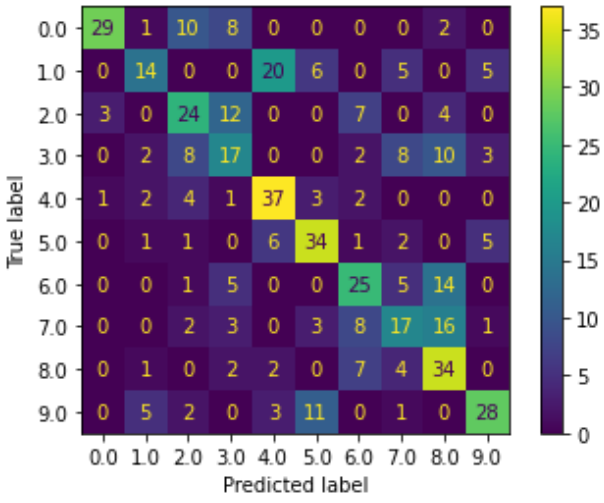
```
X_spec_train50 = np.zeros([50,4096])
y_train50 = np.zeros(50)

for i in range(10):
  rd = random.sample(range(i*200, (i+1)*200), 5)
  print(rd)
  for j in range(5):
    X_spec_train50[i*5+j] = X_spec_train[int(rd[j])]
    y_train50[i*5+j] = i
```

با توجه به تعداد کم نمونه ها و احتمال زیاد overfit شدن ابتدا بدون کاهش بعد سعی شد مدل های ساده تر با پارامترهای کمتر نیز در کنار random forest بررسی شوند:

```
pipe = Pipeline(steps=[('estimator', SVC())])

#a dict of estimator and estimator related parameters in this list
params_grid = [{
                'estimator':[SVC()],
                'estimator__kernel': ['linear', 'rbf'],
                'estimator__C': [10, 100, 1000],
                'estimator__gamma': [1e-3, 1e-4, 1e-5]
                },
                {
                'estimator': [RandomForestClassifier()],
                'estimator__n_estimators' : [50,100,150,200],
                'estimator__min_samples_leaf': [2,3,4],
                'estimator__max_features': [50,200,1000,2000],
                'estimator__min_samples_split' : [2,3,4],
                'estimator__random_state': [0]
                },
                {
                'estimator':[KNeighborsClassifier()],
                'estimator__n_neighbors':[3,4,5],
                'estimator__weights': ['uniform', 'distance']
                },
               {
                'estimator': [LogisticRegression()],
                'estimator__random_state': [0],
                'estimator__penalty': ['elasticnet'],
                'estimator__solver': ['saga'],
                'estimator__l1_ratio': [0.3,0.4,0.5],
                'estimator__C': [1,10,100]
               },
               {
                'estimator': [GaussianNB()]
               }
```

اما در نهایت مشاهده شد که در همه موارد کماکان overfit رخ میدهد و باز هم random forest نتایج بهتری دارد:(تمام نتایج در فایل cv_result_phase3_4096.xlsx قابل مشاهده هستند)

stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 150, '	0.56	1	46
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 200, '	0.56	1	46
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 50, 'e	0.58	1	7
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 100, '	0.6	1	1
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 150, '	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 200, '	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 50, 'e	0.58	1	7
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 100, '	0.6	1	1
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 150, '	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 200, '	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 50, 'e	0.58	1	7
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 100, '	0.6	1	1
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 150, '	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 200, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 200, '	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 50, '	0.48	1	142
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 100,	0.52	1	103
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 150,	0.52	1	103
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 200,	0.54	1	61
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 50, '	0.48	1	142
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 100,	0.52	1	103
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 150,	0.52	1	103
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 200,	0.54	1	61
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 50, '	0.48	1	142
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 100,	0.52	1	103
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 150,	0.52	1	103
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 2, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 200,	0.54	1	61
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 50, '	0.5	1	128
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 100,	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 150,	0.54	1	61
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 200,	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 50, '	0.5	1	128
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 100,	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 150,	0.54	1	61
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 200,	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 50, '	0.5	1	128
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 100,	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 150,	0.54	1	61
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 3, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 200,	0.56	1	28
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 50, '	0.52	1	122
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 100,	0.52	1	122
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 150,	0.6	1	1
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 2, 'estimator__n_estimators': 200,	0.58	1	7
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 50, '	0.52	1	122
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 100,	0.52	1	122
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 150,	0.6	1	1
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 3, 'estimator__n_estimators': 200,	0.58	1	7
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 50, '	0.52	1	122
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 100,	0.52	1	122
stimator': RandomForestClassifier(), 'estimator__max_features': 1000, 'estimator__min_samples_leaf': 4, 'estimator__min_samples_split': 4, 'estimator__n_estimators': 150,	0.6	1	1



0.518

	precision	recall	f1-score	support
0	0.88	0.58	0.70	50
1	0.54	0.28	0.37	50
2	0.46	0.48	0.47	50
3	0.35	0.34	0.35	50
4	0.54	0.74	0.63	50
5	0.60	0.68	0.64	50
6	0.48	0.50	0.49	50
7	0.40	0.34	0.37	50
8	0.42	0.68	0.52	50
9	0.67	0.56	0.61	50
accuracy			0.52	500
macro avg	0.54	0.52	0.51	500
weighted avg	0.54	0.52	0.51	500

Train score	Validation score	Test score
100%	60%	51.8%

با توجه به تعداد کم نمونه ها و تعداد زیاد ویژگی ها، کاهش بعد نیز بررسی شد. پس از کاهش بعد با PCA، این مراحل دوباره تکرار شدند اما دقت روی داده های ارزیابی اینبار به ۴۸٪ کاهش یافت. به همین دلیل این گزینه رد شد و نتایج آن در گزارش آورده نشده است.

ب) در این قسمت پس از خوشه بندی داده ها، لیبل ها و فاصله هر سمپل تا مرکز خوشه ها را ذخیره میکنیم. همانطور که در تصویر زیر مشاهده میشود داده ها به ۵۰ خوشه با تعداد سمپل برابر تقسیم نشده اند.

```
km = KMeans(n_clusters=50,max_iter=300,n_init=1000,random_state=0).fit(X_spec_train_scale)
```

```
#distance of each sample to cluster centers
dist = km.transform(X_spec_train_scale)
print(dist.shape)
```

```
(2000, 50)
```

```
#cluster label for each sample
labels = km.predict(X_spec_train_scale)
```

```
#count of samples in each cluster
cluster_count = np.zeros(50)
for i in range(2000):
  cluster_count[labels[i]] +=1
cluster_count
```

```
array([138.,  70.,  65.,   1., 123., 101.,  23.,   3.,   1.,  34.,   1.,
         1.,  28.,   9.,   1.,   1.,   1.,  16.,   1., 132.,  10.,  57.,
        86.,   4.,  87.,  71.,   1.,   4.,  56.,  78.,  65.,   1., 123.,
       127.,  27.,   1.,   4.,  19.,   1., 114., 106.,  40.,  18.,   5.,
         1.,  60.,  70.,   4.,   6.,   4.])
```

پ) برای محاسبه نزدیک ترین سمپل به مرکز هر خوشه، از داده های dist که در قسمت قبل ذخیره کردیم استفاده میکنیم. برای هر خوشه مینیمم فاصله تا مرکز آن را پیدا میکنیم و اندیس این داده در دیتاست را پیدامیکنیم. حالا میدانیم در دیتاستی که ذخیره کردیم اندیس های ۰ تا ۱۹۹ مربوط به کلاس ۰، اندیس های ۲۰۰ تا ۳۹۹ مربوط به کلاس ۱ و... هستند پس از این طریق میتوانیم کلاس هر نمونه را هم پیدا کنیم.

```
X_spec_tain50_KM = np.zeros([50,4096])
y_train50_KM = np.zeros(50)
label_train50 = np.zeros(50,dtype=int)
for i in range(50):
  d = dist[:, i] #distance of all samples to cluster 0 center
  index = np.argsort(d)[::][0] #least distance : nearest sample to cluster
  X_spec_tain50_KM[i] = X_spec_train[index]
  #0: 0-199, 1: 200:399, 2: 400:599 ...
  y_train50_KM[i] = index//200
  label_train50[i] = labels[index]
```

حالا تعداد نمونه های هر کلاس در دیتاست ۵۰ تایی جدیدی که ایجاد کردیم را بررسی میکنیم:

```
#count of each class samples in the 50 samples taken from clustering
count_train50 = np.zeros(10)
for i in range(50):
  count_train50[int(y_train50_KM[i])] +=1
count_train50
```

```
array([13.,  2.,  5.,  1.,  5.,  5.,  4.,  2.,  3., 10.])
```

همانطور که مشخص است توزیع کلاس ها متعادل نیست. یعنی به بیان دقیق تر در خوشه بندی، هر کلاس لزوما به ۵ خوشه تقسیم نشده و علاوه بر این، تعداد اعضای خوشه ها هم متعادل نیست. با این دیتا ست جدید با توجه به اینکه از یکی از کلاس ها فقط یک نمونه داریم، از cross validation موجود در GridSearch هم نمیتوانیم برای تعیین هایپر پارامتر بهره ببریم چون در تقسیم داده ها به آموزش و ارزیابی این ۱ نمونه نمیتواند در هر دو حضور داشته باشد!

ث) در نهایت برای لیبل زدن تمام داده ها در هر خوشه، اندیس تمام سمپل های آن خوشه را بدست می آوریم و لیبل این اندیس ها را برابر لیبل نماینده خوشه قرار میدهیم. همانطور که در قسمت پ گفته شده، به دلیل خوشه بندی نا دقیق میبینیم که مثلا کلاس ۳ در خروجی تنها ۱۹ لیبل دارد. بررسی دیگری که انجام میدهیم، دقت این نوع لیبل زدن است. لیبل زده شده با نماینده خوشه را با لیبل اصلی نمونه یعنی index//200 مقایسه میکنیم و تعداد موارد نادرست را میشماریم. همانطور که مشاهده میشود دقت 41.9% شده است یعنی به طور کلی عملکرد خوبی نداشتیم. این ممکن است به دلیل نویز موجود در داده ها باشد یا نزدیکی خوشه ها به یکدیگر.

```
#labels given according to KMeans
y_train_KM = np.zeros(2000,dtype=int)
err = 0
for i in range(50):
  #indices of all cluster i samples
  idx = np.where(km.labels_ == i)[0]
  for j in range(idx.shape[0]) :
    if idx[j]//200 != y_train50_KM[i] : #predicted label is no correct
      err +=1
  #set the labels according the label of the representative sample
  y_train_KM[idx] = y_train50_KM[i]

print('accuracy of labeling:', 1-(err/2000))
```

```
accuracy of labeling: 0.41900000000000004
```

```
#count of each class samples after labeling with KMeans
count_train = np.zeros(10)
for i in range(2000):
  count_train[int(y_train_KM[i])] +=1
count_train
```

```
array([260., 185., 136.,  19., 123., 307., 406., 270., 170., 124.])
```

مراحل ۳ بخش ب، پ و ث با استفاده از PCA نیز انجام و نتایج بررسی شدند. به ازای ۲۰۰۰ بعد نتیجه دقیقا مشابه بود، برای ۱۰۰۰ بعد:

```
accuracy of labeling: 0.378
```

```
#count of each class samples after labeling with KMeans
count_train = np.zeros(10)
for i in range(2000):
  count_train[int(y_train_KM[i])] +=1
count_train
```

```
array([213., 213.,  99.,  98., 164., 210., 192., 440., 319.,  52.])
```

برای ۵۰۰ بعد:

```
accuracy of labeling: 0.403
```

```
#count of each class samples after labeling with KMeans
count_train = np.zeros(10)
for i in range(2000):
  count_train[int(y_train_KM[i])] +=1
count_train
```

```
array([180.,  93., 171.,  68., 104., 190., 377., 196., 282., 339.])
```

برای ۲۰۰ بعد:

```
accuracy of labeling: 0.372
```

```
#count of each class samples after labeling with KMeans
count_train = np.zeros(10)
for i in range(2000):
  count_train[int(y_train_KM[i])] +=1
count_train
```

```
array([238., 334.,  40., 100.,  50., 151., 120., 540., 246., 181.])
```

همانطور که مشاهده میشود با تمام ۴۰۹۶ ویژگی بیشترین دقت را داشتیم. در نهایت دقت در هر کلاس نیز برای حالت بدون PCA بررسی شد:

```
accuracy of labeling in each class:
[0.82  0.425 0.405 0.06  0.365 0.45  0.655 0.3   0.435 0.275]
```

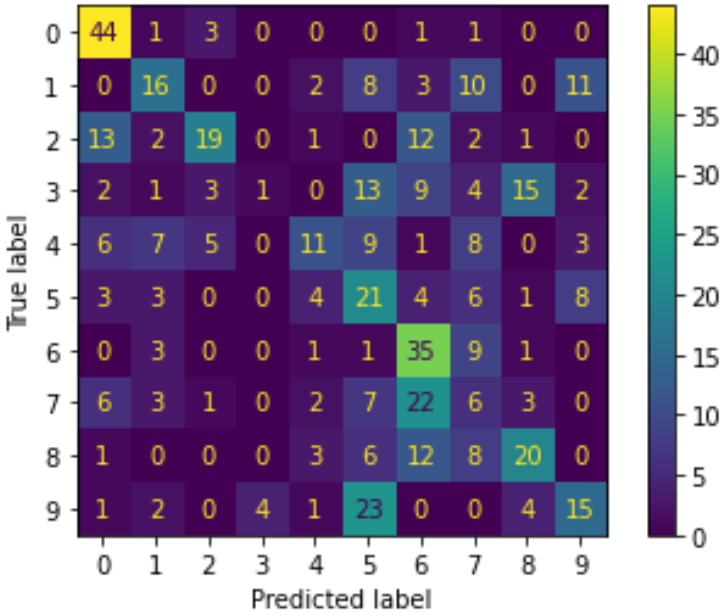
ج) با توجه به اینکه در فاز قبل با SVM به حداکثر دقت رسیدیم، در این بخش نیز سعی شد پارامتر های SVM بهینه شود:

	parameters	validation_score	train_score	rank
0	{'C': 1, 'gamma': 0.1}	0.2030	1.000000	21
1	{'C': 1, 'gamma': 0.01}	0.2410	1.000000	20
2	{'C': 1, 'gamma': 0.001}	0.8360	0.993750	11
3	{'C': 1, 'gamma': 0.0001}	0.8255	0.929375	13
4	{'C': 1, 'gamma': 1e-05}	0.6935	0.750500	15
5	{'C': 10, 'gamma': 0.1}	0.2030	1.000000	21
6	{'C': 10, 'gamma': 0.01}	0.2510	1.000000	16
7	{'C': 10, 'gamma': 0.001}	0.8490	1.000000	5
8	{'C': 10, 'gamma': 0.0001}	0.8550	0.983000	4
9	{'C': 10, 'gamma': 1e-05}	0.8235	0.917875	14
10	{'C': 100, 'gamma': 0.1}	0.2030	1.000000	21
11	{'C': 100, 'gamma': 0.01}	0.2510	1.000000	16
12	{'C': 100, 'gamma': 0.001}	0.8490	1.000000	5
13	{'C': 100, 'gamma': 0.0001}	0.8715	1.000000	3
14	{'C': 100, 'gamma': 1e-05}	0.8380	0.974375	10
15	{'C': 500, 'gamma': 0.1}	0.2030	1.000000	21
16	{'C': 500, 'gamma': 0.01}	0.2510	1.000000	16
17	{'C': 500, 'gamma': 0.001}	0.8490	1.000000	5
18	{'C': 500, 'gamma': 0.0001}	0.8730	1.000000	1
19	{'C': 500, 'gamma': 1e-05}	0.8360	0.993250	12
20	{'C': 1000, 'gamma': 0.1}	0.2030	1.000000	21
21	{'C': 1000, 'gamma': 0.01}	0.2510	1.000000	16
22	{'C': 1000, 'gamma': 0.001}	0.8490	1.000000	5
23	{'C': 1000, 'gamma': 0.0001}	0.8730	1.000000	1
24	{'C': 1000, 'gamma': 1e-05}	0.8385	0.997875	9

```
{'C': 500, 'gamma': 0.0001}
0.376
              precision    recall  f1-score   support

           0       0.58      0.88      0.70        50
           1       0.42      0.32      0.36        50
           2       0.61      0.38      0.47        50
           3       0.20      0.02      0.04        50
           4       0.44      0.22      0.29        50
           5       0.24      0.42      0.30        50
           6       0.35      0.70      0.47        50
           7       0.11      0.12      0.12        50
           8       0.44      0.40      0.42        50
           9       0.38      0.30      0.34        50

    accuracy                           0.38       500
   macro avg       0.38      0.38      0.35       500
weighted avg       0.38      0.38      0.35       500
```

همانطور که مشاهده میشود روی داده های ارزیابی به دقت حداکثر ۸۷.۳٪ رسیده ایم. اما بر روی داده های تست دقت خیلی کمتر یعنی ۳۷.۶٪ شده است. این اتفاق دو دلیل دارد: ۱. داده های ما با لیبل های جدید دیگر توزیع متعادل ندارند و مثلا کلاس ۳ تنها ۱۹ لیبل داشت پس طبقه بند ما این کلاس را خوب یاد نمیگیرد و حتی روی داده ارزیابی هم در این بخش احتمالا نتیجه خوبی نداشته باشد. ۲. دقت لیبل گذاری ما در بهترین حالت ۴۲٪ بود. یعنی حدوداً ۶۰٪ داده ها عملا لیبل اشتباه خورده اند! و نتیجتا داده های آموزش ما در این حالت نماینده خوبی از داده های تست نیستند و انتظار هم میرفت که نتیجه بر روی داده های تست بشدت افت کند، هرچند بر روی داده ارزیابی به دقت ۸۷٪ رسیدیم.(در مورد داده ارزیابی، چون با داده های آموزش به یک صورت لیبل خورده اند پس طبقه بند ما هم لیبل های غلط را یاد میگیرد و نتیجتا بر روی داده های غلط لیبل خورده دیگر نیز نتیجه خوبی میتواند بدهد!) حال اگر confusion matrix را با دقت لیبل ها در قسمت قبل مقایسه کنیم:

```
accuracy of labeling in each class:
[0.82  0.425 0.405 0.06  0.365 0.45  0.655 0.3   0.435 0.275]
```



کلاس ۰ که دقت لیبل های آن ۸۲٪ بوده بر روی داده های تست هم دقت بالایی دارد. یعنی لیبل های درست به سیگنال های کلاس ۰ داده شده و طبقه بند این کلاس را به درستی با لیبل درست یادگرفته. در ادامه کلاس ۶ با ۶۵.۶٪ دقت لیبل دومین کلاسی است که نتایج قابل قبولی دارد. در این میان کلاس ۳ که تنها ۱۹ نمونه داشت و دقت لیبل آن ۶٪ بود (یعنی از این ۱۹ نمونه تنها ۱۲ نمونه واقعا لیبل ۳ داشتند) کمترین دقت را دارد و تنها ۱ نمونه درست تشخیص داده شده است.

اگر بخواهیم نتایج این بخش را با قسمت الف مقایسه کرد، باید گفت که چون داده های ما از نوع سری زمانی هستند، تعداد بیشتر ویژگی ها به نفع ما است. نتیجتا حتی با وجود تنها ۵۰ نمونه به عنوان داده آموزش توانستیم به دقت حدود ۵۰٪ روی داده های تست برسیم. مزیت قسمت الف با ۵۰ داده نسبت به قسمت ج با ۲۰۰۰ داده این است که در قسمت الف داده ها درست لیبل زده شده اند! و به همین دلیل به دقت مشابه بر روی ارزیابی و تست رسیدیم. اما در قسمت ج داده های ارزیابی خود غلط لیبل خورده بودند و معیار مناسبی برای حداقل کردن خطا روی داده های تست نبودند، نتیجتا دقت روی ارزیابی بالا و روی تست بسیار پایین بود.