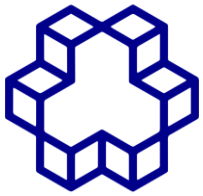
به نام خدا



دانشگاه صنعتی خواجه نصیرالدین طوسی

دانشکده برق

یادگیری ماشین

امتحان پایان ترم

Github link:

https://github.com/FatemehShokrollahiMoghadam

google drive link:

https://drive.google.com/drive/folders/1Ot1itfRgFwJMES9MD5RBcutQCEaT26v-?usp=sharing

دانشجو:

فاطمه شکراللهی مقدم

۴۰۲۰۷۳۶۴

تابستان ۱۴۰۳

Contents

پیشگفتار ۳

سوالات هماهنگ ۴

سوال اول ۴

الف ۴

ب ۷

سوالات هماهنگ نشده ۸

آو ب ۸

ج ۱۰

سوالات شبیه سازی ۱۱

سوال۴ ..... ۲۰

پیشگفتار

در صفحه اول گزارش لینگ گیت هاب و لینک گوگل درایو مربوط به نوت بوک شبیه سازی ها آورده شده است.

در اینجا نیز لینک گوگل کولب نوت بوک سوال شبیه سازی آورده می‌شود:

لینک نوت بوک سوال تحلیلی:

https://colab.research.google.com/drive/1ELS3zUECIe8UwuTORfi0wPHZcQbpJIVU?usp=sharing

لینک نوت بوک سوال ۳:

https://colab.research.google.com/drive/1Se1tiz5miT3ZeykDOebOLPrj506k7t2i?usp=sharing

سوالات هماهنگ

سوال اول

الف

مطابق مقاله (vapnik, 1995) SVM برای ساختن یک hyperplane جداکننده soft margine، باید تابع

$$\Phi = \frac{1}{2}\mathbf{w}\cdot\mathbf{w} + C\left(\sum_{i=1}^{\ell}\xi_i\right)^k, \qquad k > 1,$$

تحت شرایط زیر حداکثر شود:

$$y_i(\mathbf{x}_i\cdot\mathbf{w}+b) \geq 1-\xi_i, \qquad i=1,\ldots,\ell,$$
$$\xi_i \geq 0, \qquad i=1,\ldots,\ell.$$

برای این منظور تابع لاگرانژ زیر را در نظر می گیریم:

$$L(w.\xi.b.\Lambda.R)$$
$$= \frac{1}{2}w\cdot w + C\left(\sum_{i=1}^{l}\xi_i\right)^k - \sum_{i=1}^{l}\alpha_i[y_i(x_i\cdot w+b)-1+\xi_i]$$
$$-\sum_{i=1}^{l}r_i\xi_i$$

Λ بردار ضرایب نامنفی α_i است. و R بردار r_i است.

حال برای یافتن نقطه زینی برای کمینه کردن تابع نسبت به $\xi.w.b$ (بیشینه سازی نسبت به $\alpha_i.r_i$) داریم:

$$\frac{\partial L}{\partial w} = w_0 - \sum_{i=1}^{l}\alpha_i y_i x_i = 0 \quad (1)$$

$$\frac{\partial L}{\partial b} = \sum_{i=1}^{l}\alpha_i y_i = 0 \quad (2)$$

$$\frac{\partial L}{\partial \xi_i} = kC\left(\sum_{i=1}^{l} {\xi_i}^0\right)^{k-1} - \alpha_i - r_i \quad (3)$$

حال اگر: $\sum_{i=1}^{l} {\xi_i}^0 = (\frac{\delta}{Ck})^{\frac{1}{k-1}}$ (*)

رابطه (3) بصورت زیر در می آید:

$$\delta = \alpha_i + r_i \geq 0$$

با استفاده از نتایج حاصل از روابط (1) تا (3) و جایگذاری در تابع لاگرانژ خواهیم داشت:

$$W(\mathbf{\Lambda}, \delta) = \sum_{i=1}^{\ell} \alpha_i - \frac{1}{2}\sum_{i=1}^{\ell}\sum_{j=1}^{\ell} \alpha_i \alpha_j y_i y_j \mathbf{x}_i \cdot \mathbf{x}_j - \frac{\delta^{k/k-1}}{(kC)^{1/k-1}}\left(1 - \frac{1}{k}\right)$$

برای یافتن ابر صفحه ای که تابع بالا را تحت شرایط زیر کمینه کند،

$$\delta - \alpha_i - r_i = 0$$

$$\sum_{i=1}^{l} \alpha_i y_i = 0$$

$$w_0 = \sum_{i=1}^{l} \alpha_i y_i x_i$$

$$\alpha_i \geq 0 \text{ و } r_i \geq 0$$

نوشتار برداری زیر را برای تابع در نظر می گیریم:

$$W(\mathbf{\Lambda}, \delta) = \mathbf{\Lambda}^T \mathbf{1} - \left[\frac{1}{2}\mathbf{\Lambda}^T \mathbf{D} \mathbf{\Lambda} + \frac{\delta^{k/k-1}}{(kC)^{1/k-1}}\left(1 - \frac{1}{k}\right)\right]$$

شرایط مذکور در این فرمت نوشتاری بصورت زیر تغییر می کند:

$$\Lambda^T \mathrm{Y} = 0$$

$$\Lambda + R = \delta 1$$

$$\Lambda \geq 0. \quad R \geq 0$$

از شروط بالا به

$$0 \leq \Lambda \leq \delta 1 \quad (4)$$

می رسیم.

از رابطه (❋) داریم:

$$\delta = \mathrm{Ck}(\sum_{i=1}^{l} {\xi_i}^{0})^{k-1}$$

حال با در نظر گرفتن k=1 رابطه (4) بصورت زیر محقق می شود:

$$0 \leq \Lambda \leq C$$

ب

$$S_t = \sum_{i=1}^{N} (x_i - \mu)(x_i - \mu)^T \longrightarrow \text{Total Scatter matrix}$$

$$S_w = \sum_{c=1}^{C} \sum_{i=1}^{N} (x_i^{(c)} - \mu_c)(x_i^{(c)} - \mu_c)^T \longrightarrow \text{Within Scatter matrix}$$

$$S_b = \sum_{c=1}^{C} (\mu_c - \mu)(\mu_c - \mu)^T \times N_c \longrightarrow \text{Between Scatter matrix}$$

در روابط بالا N تعداد نمونه‌ها و C تعداد کلاس‌ها و N_c تعداد نمونه در کلاس c است.

می‌توان نوشت:

$$x_i^{(c)} - \mu = (x_i^{(c)} - \mu_c) + (\mu_c - \mu)$$

$$\Rightarrow S_t = \sum_{c=1}^{C} \sum_{i=1}^{N_c} [(x_i^{(c)} - \mu_c) + (\mu_c - \mu)][(x_i^{(c)} - \mu_c) + (\mu_c - \mu)]^T$$

$$S_t = \sum_{c=1}^{C} \sum_{i=1}^{N_c} [(x_i^{(c)} - \mu_c)(x_i^{(c)} - \mu_c)^T + (x_i^{(c)} - \mu_c)(\mu_c - \mu)^T + (\mu_c - \mu)(x_i^{(c)} - \mu_c)^T + (\mu_c - \mu)(\mu_c - \mu)^T]$$

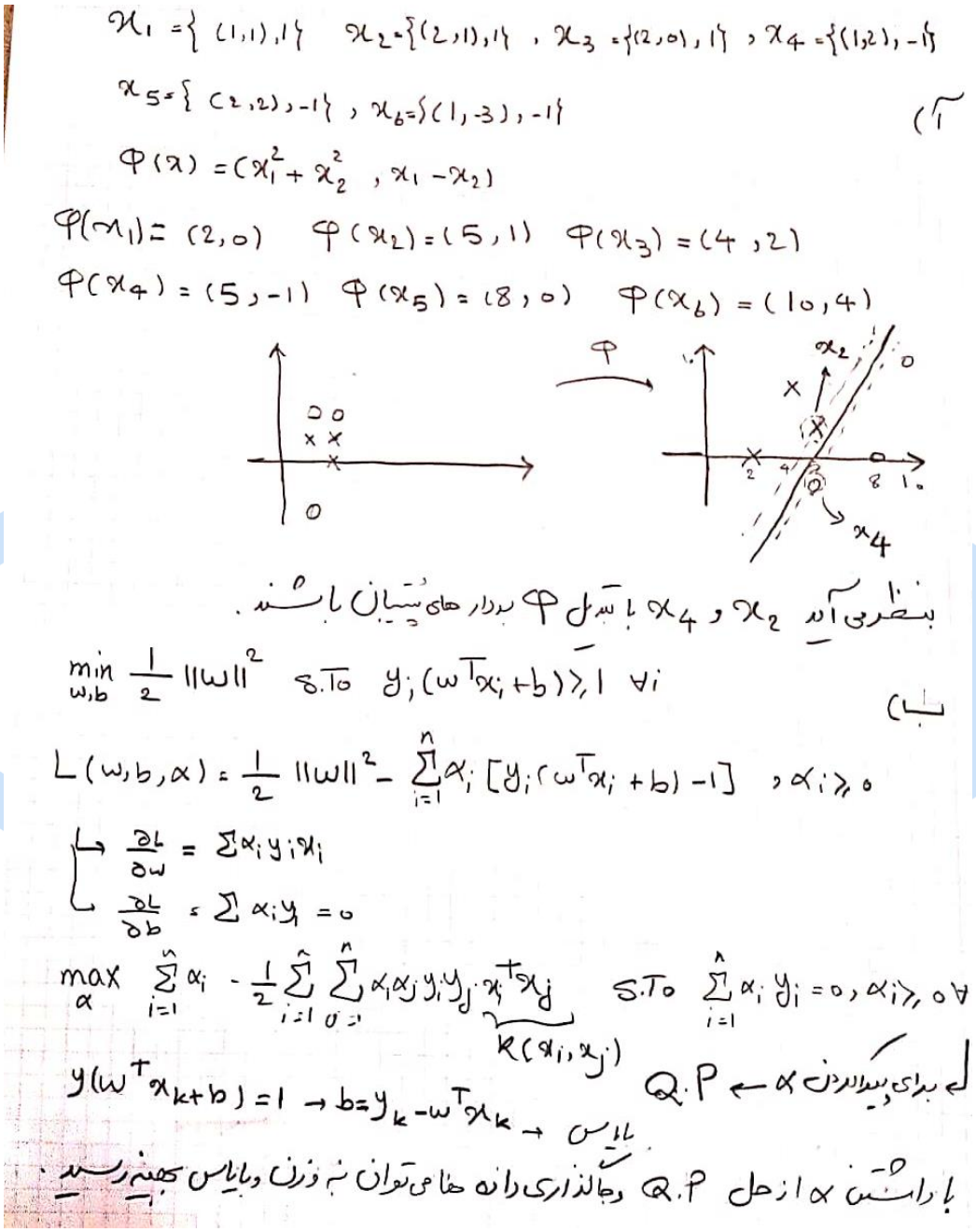
$$\sum_{i=1}^{N_c} (x_i^{(c)} - \mu_c) = 0$$

$$\Rightarrow S_t = \underbrace{\sum_{c=1}^{C} \sum_{i=1}^{N_c} (x_i^{(c)} - \mu_c)(x_i^{(c)} - \mu_c)^T}_{S_w} + \underbrace{\sum_{c=1}^{C} (\mu_c - \mu)(\mu_c - \mu)^T}_{S_b}$$

$$\Rightarrow \boxed{S_t = S_w + S_b}$$

سوالات هماهنگ نشده

آ و ب



مقادیر بهینه وزن و بایاس به صورت زیر محاسبه شد:

```python
import numpy as np
from cvxopt import matrix, solvers

# Define the kernel function
def kernel(x1, x2):
    return np.dot(x1, x2)

# Transformed points and labels
points = np.array([
    [1, 1],
    [2, 1],
    [2, 0],
    [1, 2],
    [2, 2],
    [1, -3]
])

labels = np.array([1, 1, 1, -1, -1, -1])

# Number of points
n = len(labels)

# Kernel matrix
K = np.zeros((n, n))
for i in range(n):
    for j in range(n):
        K[i, j] = kernel(points[i], points[j])
C=0.1
# Define the quadratic programming problem
P = matrix(np.outer(labels, labels) * K)
q = matrix(-np.ones(n))
G = matrix(np.vstack((-np.eye(n), np.eye(n))))
h = matrix(np.hstack((np.zeros(n), np.ones(n) * C)))
A = matrix(labels, (1, n), 'd')
b = matrix(0.0)

# Solve the quadratic programming problem
solution = solvers.qp(P, q, G, h, A, b)
alphas = np.ravel(solution['x'])

# Support vectors have non zero lagrange multipliers
sv = alphas > 1e-5
alpha_sv = alphas[sv]
points_sv = points[sv]
labels_sv = labels[sv]

# Calculate weights
w = np.sum(alpha_sv[:, None] * labels_sv[:, None] * points_sv, axis=0)

# Calculate bias
b = np.mean(labels_sv - np.dot(points_sv, w))

print("Optimal weights:", w)
print("Optimal bias:", b)
```

نتیجه به صورت زیر است:

```
     pcost       dcost       gap    pres   dres
 0: -3.0878e+00 -1.2889e+00  3e+01  5e+00  8e-16
 1: -4.8729e-01 -1.2132e+00  8e-01  1e-02  6e-16
 2: -5.4618e-01 -6.2068e-01  8e-02  1e-03  3e-16
 3: -5.8864e-01 -5.9072e-01  2e-03  3e-05  3e-16
 4: -5.8999e-01 -5.9001e-01  2e-05  3e-07  4e-16
 5: -5.9000e-01 -5.9000e-01  2e-07  3e-09  4e-16
Optimal solution found.
Optimal weights: [0.09999996 0.09999973]
Optimal bias: -0.19999981122475738
```

ج

می توانیم از تبدیل چند جمله ای استفاده کنیم. اگر مجموعه داده ای در یک فضای دو بعدی با ویژگی های (x1,x2) (x1,x2) داشته باشیم، می توانیم با استفاده از یک تابع درجه دوم مانند:

سوالات شبیه سازی

آرگومان random_state که به‌منظور ایجاد قابلیت تولید مجدد استفاده می‌شود، در طول شبیه سازی ها برابر دو رقم اخر شماره دانشجویی(۶۴) در نظر گرفته می‌شود.

ابتدا دیتاست را فراخوانی می کنیم. با دستور ()head. نیز ۵ سط ابتدایی دیتاست قابل مشاهده است:

```
data = pd.read_csv('/content/MJMusicDataset.csv')
print(data.head())
```

```
                          name dastgah instrument  zero_corssing  \
0        ney-mahoor-ebrahimi .mp3     D_2        I_4          65545
1  Mohammad_Shojaei_nei_Mahoor.mp3     D_2        I_4          59788
2     Arash_Samimi_nei_Mahoor.mp3     D_2        I_4          85072
3         19 sarebaang mahoor.mp3     D_2        I_4          89980
4                  Amjadian.mp3     D_2        I_4          58134

   spectral_centroid_mean  spectral_centroid_var  spectral_rolloff_mean  \
0             1938.040517          303472.4474            2815.166310
1             1956.981873          774951.2677            2971.032035
2             2735.525193          575671.7929            4268.615855
3             2629.389833          406198.9319            4071.233715
4             1659.262559          332341.1003            2789.041468

   spectral_rolloff_var  chroma_1_mean  chroma_2_mean  ...  mfcc_11_var  \
0          1.754412e+06       0.515210       0.158007  ...   482.619965
1          2.082504e+06       0.209722       0.335278  ...   435.338196
2          2.663909e+06       0.105867       0.108526  ...   299.526794
3          1.005496e+06       0.153988       0.098157  ...   305.978638
4          8.496746e+05       0.077795       0.036009  ...    81.764854

   mfcc_12_var  mfcc_13_var  mfcc_14_var  mfcc_15_var  mfcc_16_var  \
0   274.530334   257.260315   200.793167   177.008484    89.304535
1   289.941559   337.219269   266.507416   256.131317   153.390289
2   226.705948   163.451355   140.664673   104.658630    72.048088
3   154.510696   281.793976   348.872650   156.062454    85.470512
4   106.146141   103.081779    85.252548    90.831291   150.201111

   mfcc_17_var  mfcc_18_var  mfcc_19_var  mfcc_20_var
0    49.159683    83.412254   154.657501   417.060425
1    90.073257   120.354729   206.031006   267.015015
2    53.397228    83.085548   190.668487   331.626526
3    45.904568   120.363800   370.077820   306.180878
4   332.417633   299.858795   177.197845   186.112488

[5 rows x 72 columns]
```

پس از فراخوانی کتابخانه ها و مجموعه داده، با دستور ()info. به بررسی ابعاد و تعداد نمونه ها و همچنین با دستور ()isnull وجود یا عدم وجود داده پوچ را بررسی می کنیم.

```
print(data.isnull().sum())
data.dropna(inplace=True)
```

```
name                    0
dastgah                 0
instrument              0
zero_corssing           0
spectral_centroid_mean  0
                       ..
mfcc_16_var             0
mfcc_17_var             0
mfcc_18_var             0
mfcc_19_var             0
mfcc_20_var             0
Length: 72, dtype: int64
```

مشاهده می شود که هیچ داده پوچی نداریم.

با دستور drop ستون های name و instrument را از دیتاست حذف کرده و ویژگی ها و لیبل ها که نام دستگاه است را جدا کرده و دیتاست را باتوجه به کمترین داده های موجود در کلاس ها بالانس می کنیم.

```
# Split data into features (X) and labels (y)
import numpy as np
from imblearn.under_sampling import RandomUnderSampler
data = data.drop(['instrument','name'], axis=1)
X = data.drop('dastgah', axis=1)
y = data['dastgah']

# Count the number of samples in each class before balancing
unique, counts = np.unique(y, return_counts=True)
print("Class counts before balancing:", dict(zip(unique, counts)))

# Use RandomUnderSampler to balance the classes
rus = RandomUnderSampler(random_state=64)
X_resampled, y_resampled = rus.fit_resample(X, y)

# Count the number of samples in each class after balancing
unique_resampled, counts_resampled = np.unique(y_resampled, return_counts=True)
print("Class counts after balancing:", dict(zip(unique_resampled, counts_resampled)))

# Print the shape of the new balanced dataset
print("\nNew balanced dataset shape:", X_resampled.shape, y_resampled.shape)
```

```
Class counts before balancing: {'D_0': 122, 'D_1': 122, 'D_2': 139, 'D_3': 144, 'D_4': 116, 'D_5': 141, 'D_6': 142}
Class counts after balancing: {'D_0': 116, 'D_1': 116, 'D_2': 116, 'D_3': 116, 'D_4': 116, 'D_5': 116, 'D_6': 116}

New balanced dataset shape: (812, 69) (812,)
```

کمترین تعداد داده مربوط به کلاس D_4 با ۱۱۶ داده است. بنابراین دیتاست را به نحوی بالانس میکنیم که از هر کلاس ۱۱۶ داده وجود داشته باشد. در نهایت دیتاست جدید دارای ۸۱۲ نمونه و ۶۹ ویژگی است.

حال ماتریس همبستگی را رسم می کنیم برای این کار باید ستون مربوط به دستگاه را بصورت عددی تبدیل کنیم و از یک نگاشت استفاده می کنیم در پایان رسم ماتریس از همین نگاشت برای بازگردانی لیبل های دستگاه استفاده می کنیم:

```python
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt

# Assuming X_resampled and y_resampled are already defined
labels_df = pd.DataFrame(y_resampled, columns=['dastgah'])

# Concatenate the data matrix and the label DataFrame
data = pd.concat([X_resampled, labels_df], axis=1)

# Convert labels to integers
label_mapping = {label: idx for idx, label in enumerate(labels_df['dastgah'].unique())}
data['dastgah_int'] = data['dastgah'].map(label_mapping)

# Ensure all columns except the label column are numeric
numeric_cols = data.select_dtypes(include=[np.number]).columns.tolist()

# Calculate correlation matrix
corr_matrix = data[numeric_cols + ['dastgah_int']].corr()

# Create heatmap using seaborn
plt.figure(figsize=(25, 25))
sns.heatmap(corr_matrix, annot=True, cmap='coolwarm', linewidths=0.5, annot_kws={"size": 8}, fmt='.3f', yticklabels=corr_matrix.columns)

# Adjust font size of annotations
plt.xticks(fontsize=12)
plt.yticks(fontsize=12)

# Adjust margins of PDF file
plt.savefig('PIcS1.pdf', bbox_inches='tight')
plt.show()

# Revert labels back to original
data['dastgah'] = data['dastgah_int'].map({v: k for k, v in label_mapping.items()})
data.drop(columns=['dastgah_int'], inplace=True)
```

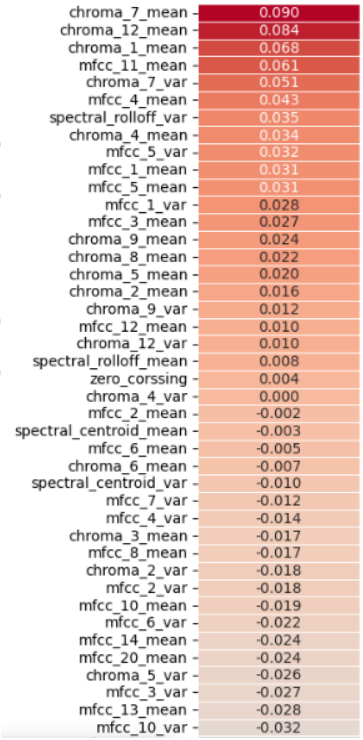
لینک مشاهده ماتریس همبستگی:

https://drive.google.com/file/d/1RQRVjw1eszMaYA1bTuusXJ50_vaY9hwT/view?usp=sharing

برای انتخاب ویژگی، به ترتیب ویژگی هایی که دارای بیشترین همبستگی با هدف هستند را sort کرده و آنهایی که بیش از ۲٪ همبستگی دارند را انتخاب و دیتاستی جدید با ویژگی های انتخاب شده ایجاد می کنیم.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np

# Assuming y_resampled and X_resampled are already defined
labels_df = pd.DataFrame(y_resampled, columns=['dastgah'])

# Concatenate the data matrix and the label DataFrame
data = pd.concat([X_resampled, labels_df], axis=1)

# Convert labels to integers
label_mapping = {label: idx for idx, label in enumerate(labels_df['dastgah'].unique())}
data['dastgah_int'] = data['dastgah'].map(label_mapping)

# Ensure all columns except the label column are numeric
numeric_cols = data.select_dtypes(include=[np.number]).columns.tolist()

# Remove 'dastgah_int' for correlation calculation
numeric_cols.remove('dastgah_int')

# Calculate correlation matrix
corr_matrix1 = data[numeric_cols].corrwith(data['dastgah_int']).sort_values(ascending=False)

# Create heatmap using seaborn
plt.figure(figsize=(2, 15))
sns.heatmap(corr_matrix1.to_frame(), annot=True, cmap='coolwarm', linewidths=0.5, annot_kws={"size": 10}, fmt='.3f', cbar=False)

# Rotate x-axis tick labels to be horizontal
plt.xticks(rotation=0)
plt.show()

# Revert labels back to original
data['dastgah'] = data['dastgah_int'].map({v: k for k, v in label_mapping.items()})
data.drop(columns=['dastgah_int'], inplace=True)
```

بخشی از نتیجه:



جداسازی ویژگی ها با بیشترین همبستگی با خروجی:

```
top_16_features = corr_matrix1.head(16).index.tolist()
data_train= data[top_16_features + ['dastgah']]
print(data_train)
```

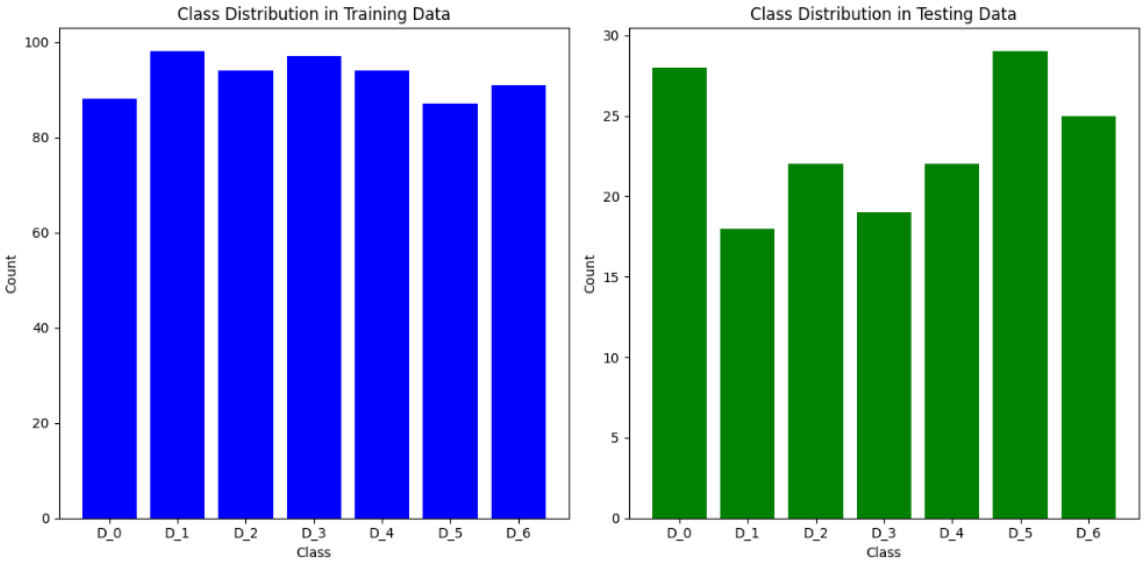
```
     chroma_7_mean  chroma_12_mean  chroma_1_mean  mfcc_11_mean  chroma_7_var  \
0         0.147926        0.224913       0.313241      4.627033      0.069000
1         0.172749        0.149601       0.273068    -21.546146      0.023120
2         0.056556        0.116372       0.390317     -1.911680      0.004121
3         0.206671        0.283500       0.132657     -4.977403      0.090737
4         0.390417        0.672468       0.247183    -17.156363      0.109467
..             ...             ...            ...           ...           ...
807       0.123933        0.427011       0.646458     -6.583026      0.026869
808       0.281922        0.193881       0.072602     -6.222386      0.086079
809       0.193709        0.299603       0.546076     -9.485244      0.050254
810       0.359442        0.119076       0.060373      2.182932      0.138261
811       0.221392        0.092544       0.060705    -15.084551      0.090236

     mfcc_4_mean  spectral_rolloff_var  chroma_4_mean  mfcc_5_var  \
0      -7.943051          2.004491e+06       0.367564  382.870178
1       5.863482          1.283898e+06       0.313244  195.298157
2       7.991030          1.397087e+05       0.289907  223.082123
3      31.408905          3.294337e+05       0.171328  613.377441
4      -1.507796          6.803355e+05       0.324291  159.495590
..           ...                   ...            ...         ...
807    15.744759          1.386179e+06       0.129396  251.794846
808     8.230885          1.561480e+06       0.076422  188.775223
809     4.333771          9.264193e+03       0.168443  227.974365
810     0.055459          4.718581e+05       0.387149  129.825195
811    12.052809          5.836073e+05       0.144373  350.432251

     mfcc_1_mean  mfcc_5_mean    mfcc_1_var  mfcc_3_mean  chroma_9_mean  \
0    -246.897537   -38.925575   8929.200195   -51.357330       0.254767
1    -207.880859   -34.239590   8404.088867   -60.536015       0.157513
2    -238.993210   -32.901897   4547.577637   -87.864044       0.173376
3    -303.290039   -37.964108  10534.710940   -74.569801       0.446477
4    -205.767929   -35.894131   6087.571777   -75.156235       0.081766
..           ...          ...           ...          ...            ...
807  -353.969391   -17.628613  10222.173830   -30.709774       0.292166
808  -266.990723    -3.321193  11687.807620   -15.514976       0.094005
809  -407.880371   -13.410451   2660.318604    72.425865       0.183543
810  -234.390335   -22.904810   3797.703125   -56.671722       0.107117
811  -220.636993   -57.624599   5063.823730   -61.585102       0.662884

     chroma_8_mean  chroma_5_mean dastgah
0         0.197679       0.188779     D_0
1         0.304773       0.305772     D_0
2         0.152776       0.079829     D_0
3         0.132157       0.202210     D_0
4         0.218806       0.291842     D_0
..             ...            ...     ...
807       0.195101       0.112167     D_6
808       0.199557       0.217698     D_6
809       0.345162       0.217669     D_6
810       0.230194       0.306345     D_6
811       0.374562       0.104248     D_6
```

حال داده ها را با نسبت ۸۰ و ۲۰ به آموزش و آزمون تقسیم می کنیم:

```
from sklearn.model_selection import train_test_split
np.random.seed(64)
array =data_train.values
np.random.shuffle(array)  # Shuffle the array
shuffled_df = pd.DataFrame(array, columns=data_train.columns)
X = shuffled_df.drop(columns='dastgah').values
y = shuffled_df['dastgah'].values
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.2, random_state=64)
print('Train:', X_train.shape, y_train.shape, '\nTest:', X_test.shape, y_test.shape)
```

```
Train: (649, 16) (649,)
Test: (163, 16) (163,)
```

نمودار پراکندگی کلاس ها در هر دیتاست بصورت زیر است:



با استفاده از StandardScaler داده ها را مقیاس کنیم در این عملیات توجه داریم که از اطلاعات داده های آزمون استفاده نکنیم؛ چراکه باعث نشت اطلاعات میگردد. نشت اطلاعات به معنای انتقال اطلاعات از داده های تستی به مدل یادگیری است که باعث میشود مدل بهترین عملکرد را بر روی داده های تستی نشان دهد؛ اما در واقع مدل تعمیم پذیری و عمل کرد خوبی ندارد. دستورات مربوط به این قسمت به شرح زیر است:

```
from sklearn.preprocessing import StandardScaler
scaler=StandardScaler()
scaler.fit(X_train)
X_train = scaler.transform(X_train)
print(X_train.shape)
X_test = scaler.transform(X_test)
print(X_test.shape)
```

```
(649, 16)
(163, 16)
```

حال با تبدیل pca داده ها را به دو بعد کاهش داده و از طبقه بند svm استفاده می کنیم:

```
from sklearn.decomposition import PCA
np.random.seed(64)
pca = PCA(n_components=2)
X_train = pca.fit_transform(X_train)
X_test = pca.transform(X_test)
```

برای یافتن هایپر پارامترهای بهینه از gridsearch استفاده می کنیم:

```
from sklearn.svm import SVC
param_grid = {'C': [0.1, 1, 10, 100, 1000], 'kernel': ['linear']}
from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=3)
grid.fit(X_train, y_train)
grid.best_estimator_
```

```
Fitting 5 folds for each of 5 candidates, totalling 25 fits
[CV 1/5] END .............C=0.1, kernel=linear;, score=0.154 total time=   0.0s
[CV 2/5] END .............C=0.1, kernel=linear;, score=0.169 total time=   0.0s
[CV 3/5] END .............C=0.1, kernel=linear;, score=0.162 total time=   0.0s
[CV 4/5] END .............C=0.1, kernel=linear;, score=0.138 total time=   0.0s
[CV 5/5] END .............C=0.1, kernel=linear;, score=0.178 total time=   0.0s
[CV 1/5] END ...............C=1, kernel=linear;, score=0.131 total time=   0.0s
[CV 2/5] END ...............C=1, kernel=linear;, score=0.169 total time=   0.0s
[CV 3/5] END ...............C=1, kernel=linear;, score=0.162 total time=   0.0s
[CV 4/5] END ...............C=1, kernel=linear;, score=0.146 total time=   0.0s
[CV 5/5] END ...............C=1, kernel=linear;, score=0.202 total time=   0.0s
[CV 1/5] END ..............C=10, kernel=linear;, score=0.131 total time=   0.1s
[CV 2/5] END ..............C=10, kernel=linear;, score=0.169 total time=   0.0s
[CV 3/5] END ..............C=10, kernel=linear;, score=0.162 total time=   0.0s
[CV 4/5] END ..............C=10, kernel=linear;, score=0.146 total time=   0.0s
[CV 5/5] END ..............C=10, kernel=linear;, score=0.202 total time=   0.0s
[CV 1/5] END .............C=100, kernel=linear;, score=0.131 total time=   0.3s
[CV 2/5] END .............C=100, kernel=linear;, score=0.169 total time=   0.4s
[CV 3/5] END .............C=100, kernel=linear;, score=0.162 total time=   0.4s
[CV 4/5] END .............C=100, kernel=linear;, score=0.146 total time=   0.3s
[CV 5/5] END .............C=100, kernel=linear;, score=0.202 total time=   0.4s
[CV 1/5] END ............C=1000, kernel=linear;, score=0.131 total time=  10.6s
[CV 2/5] END ............C=1000, kernel=linear;, score=0.169 total time=   5.9s
[CV 3/5] END ............C=1000, kernel=linear;, score=0.162 total time=   9.4s
[CV 4/5] END ............C=1000, kernel=linear;, score=0.146 total time=   4.8s
[CV 5/5] END ............C=1000, kernel=linear;, score=0.202 total time=   8.3s
SVC
SVC(C=1, kernel='linear')
```

حال با c=1 و هسته خطی کلاس بندی را انجام می دهیم. زمان اجرای کد بسیارطولانی شد و در داخل نوت بوک شبیه سازی مربوط به دیتاست میانترم برایsvmخطی با جزئیات آمده است. و ارتباطی با دیتاست پایان ترم ندارد.

در ادامه کرنل rbf گزارش می شود:

برای کرنل rbf همین مراحل را تکرار می کنیم

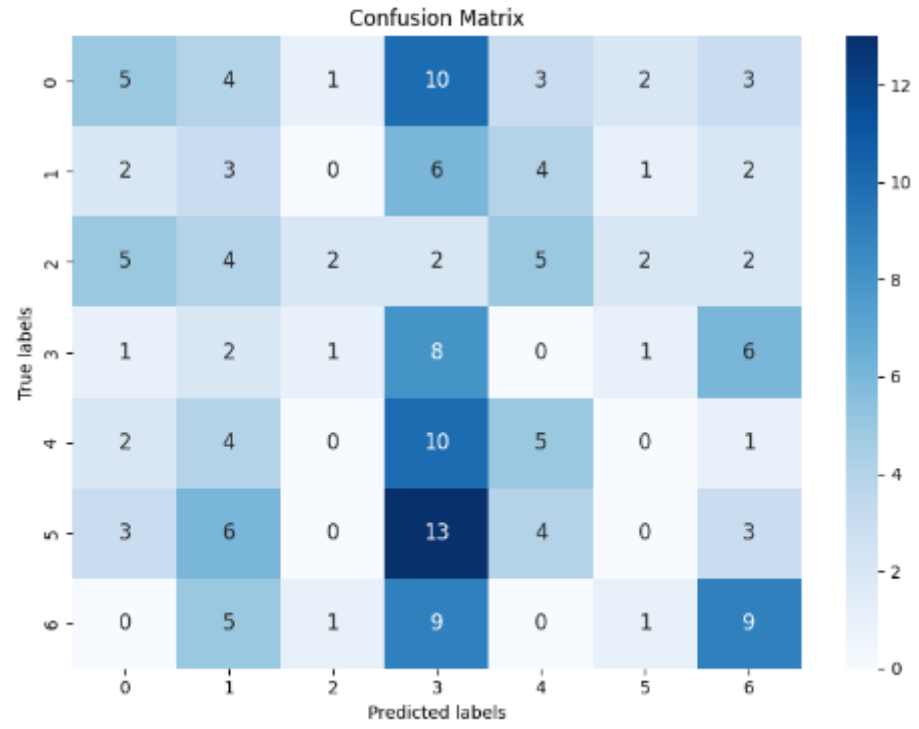
```
from sklearn.svm import SVC
param_grid = {'C': [0.1, 1, 10,], 'kernel': ['rbf']}
#param_grid = {'C': [0.1, 1, 10, 100, 1000], 'kernel': ['poly','rbf','sigmoid'],'coef0': [0, 0.1, 1, 2, 5, 1], 'degree':[2, 3, 4, 5],'gamma':['scale', 'auto']}

from sklearn.model_selection import GridSearchCV
grid = GridSearchCV(SVC(), param_grid, refit=True, verbose=4)
grid.fit(X_train, y_train)
grid.best_estimator_
```

```
Fitting 5 folds for each of 3 candidates, totalling 15 fits
[CV 1/5] END .................C=0.1, kernel=rbf;, score=0.162 total time=   0.0s
[CV 2/5] END .................C=0.1, kernel=rbf;, score=0.154 total time=   0.0s
[CV 3/5] END .................C=0.1, kernel=rbf;, score=0.154 total time=   0.0s
[CV 4/5] END .................C=0.1, kernel=rbf;, score=0.177 total time=   0.0s
[CV 5/5] END .................C=0.1, kernel=rbf;, score=0.155 total time=   0.0s
[CV 1/5] END ...................C=1, kernel=rbf;, score=0.177 total time=   0.0s
[CV 2/5] END ...................C=1, kernel=rbf;, score=0.169 total time=   0.0s
[CV 3/5] END ...................C=1, kernel=rbf;, score=0.169 total time=   0.0s
[CV 4/5] END ...................C=1, kernel=rbf;, score=0.162 total time=   0.0s
[CV 5/5] END ...................C=1, kernel=rbf;, score=0.155 total time=   0.0s
[CV 1/5] END ..................C=10, kernel=rbf;, score=0.215 total time=   0.0s
[CV 2/5] END ..................C=10, kernel=rbf;, score=0.192 total time=   0.0s
[CV 3/5] END ..................C=10, kernel=rbf;, score=0.200 total time=   0.0s
[CV 4/5] END ..................C=10, kernel=rbf;, score=0.154 total time=   0.0s
[CV 5/5] END ..................C=10, kernel=rbf;, score=0.178 total time=   0.0s
▾ SVC
SVC(C=10)
```

ماتریس در هم ریختگی بصورت زیر است:

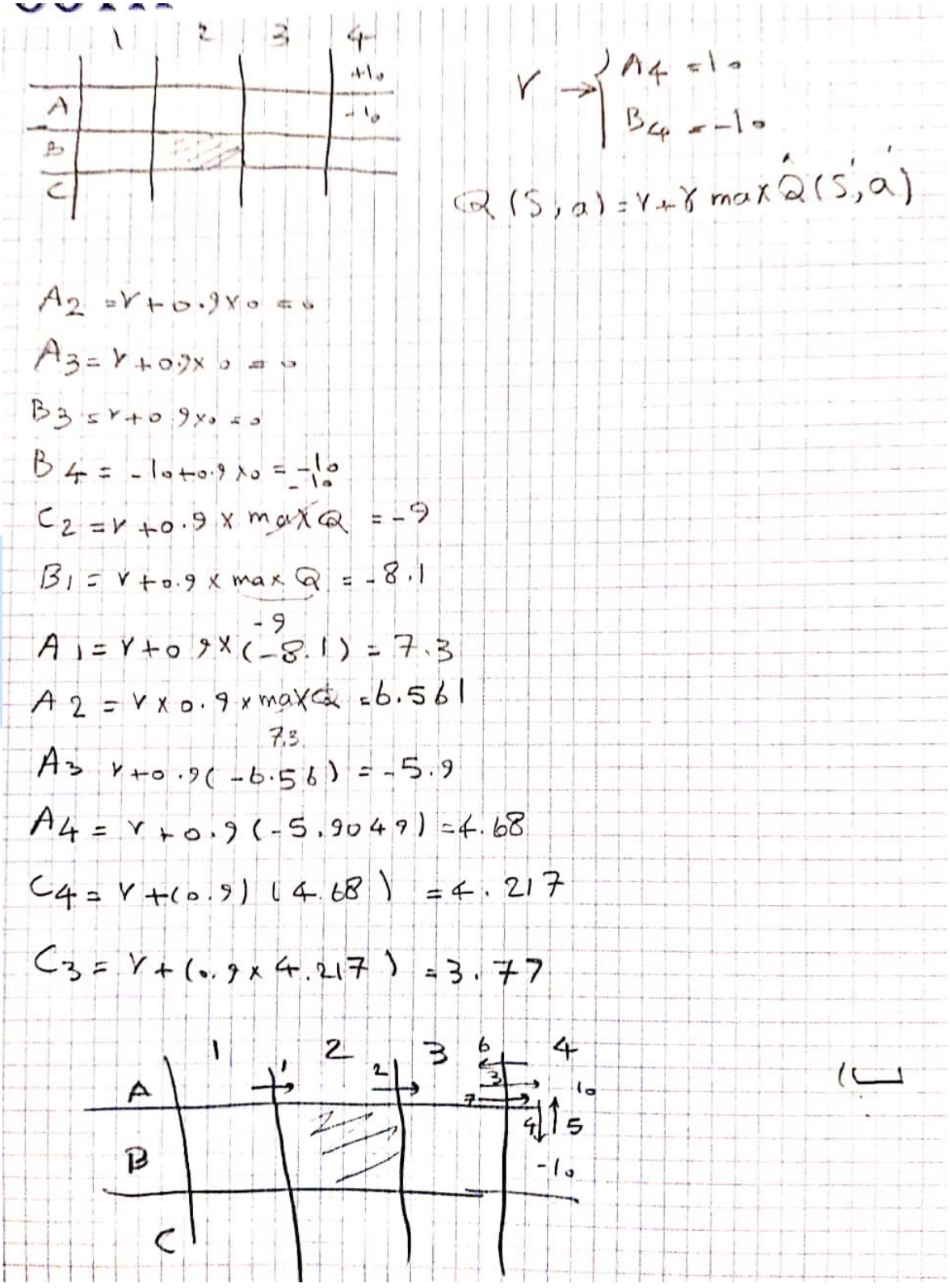
دقت کمی مشاهده می شود. همچنین نواحی تصمیم گیری به دلیل حجم بالای محاسبات قابل ترسیم نبود.

```python
from sklearn.metrics import confusion_matrix, classification_report
np.random.seed(64)
y_pred = model.predict(X_test)
cf_matrix = confusion_matrix(y_test, y_pred)
# Plotting confusion matrix as a heatmap with fitted text
plt.figure(figsize=(8, 6))
sns.heatmap(cf_matrix, annot=True, fmt='d', cmap='Blues', annot_kws={"size": 12})
# Get the axis to modify layout
plt.gca().set_ylim(len(np.unique(y_test)), 0)  # Fix for matplotlib 3.1.1 and 3.1.2
plt.title('Confusion Matrix')
plt.xlabel('Predicted labels')
plt.ylabel('True labels')

# Save the plot as PNG
plt.tight_layout()
plt.savefig('confusion_matrix.png', dpi=300)
plt.show()
from sklearn.metrics import accuracy_score
print('Accuracy :',accuracy_score(y_test,y_pred))
```




```
Accuracy : 0.1963190184049079
```

سوال ۴


1
2
3
4
+10
A
-10
B
C
r → A4 = 10
B4 = -10
Q(S,a) = r + γ max Q(S',a')
A2 = r + 0.9 x 0 = 0
A3 = r + 0.9 x 0 = 0
B3 = r + 0.9 x 0 = 0
B4 = -10 + 0.9 x 0 = -10
C2 = r + 0.9 x max Q = -9
B1 = r + 0.9 x max Q = -8.1
A1 = r + 0.9 x (-8.1) = 7.3
A2 = r x 0.9 x max Q = 6.561
A3 r + 0.9 (-6.56) = -5.9
A4 = r + 0.9 (-5.9049) = 4.68
C4 = r + (0.9) (4.68) = 4.217
C3 = r + (0.9 x 4.217) = 3.77
1
2
3
4
A
B
C
10
5
-10