

Geographical Original of Music Data Set

[Download: Data Folder](#), [Data Set Description](#)

Abstract: Instances in this dataset contain audio features extracted from 1059 wave files. The task associated with the data is to predict the geographical origin of music.

table for the dataset is shown below:

Data Set Characteristics:	Multivariate	Number of Instances:	1059	Area:	N/A
Attribute Characteristics:	Real	Number of Attributes:	68	Date Donated	2014-10-18
Associated Tasks:	Classification, Regression	Missing Values?	N/A	Number of Web Hits:	110415

Source:

Creators:

Fang Zhou (fang.zhou '@' nottingham.edu.cn)

The University of Nottingham, Ningbo, China

Donors of the Dataset:

Fang Zhou (fang.zhou '@' nottingham.edu.cn)

Claire Q (eskoala '@' gmail.com)

Ross D. King (ross.king '@' manchester.ac.uk)

Data Set Information:

The dataset was built from a personal collection of 1059 tracks covering 33 countries/area. The music used is traditional, ethnic or 'world' only, as classified by the publishers of the product on which it appears. Any Western music is not included because its influence is global - what we seek are the aspects of music that most influence location. Thus, being able to specify a location with strong influence on the music is central.

The geographical location of origin was manually collected the information from the CD sleeve notes, and when this information was inadequate we searched other information sources. The location data is limited in precision to the country of origin.

The country of origin was determined by the artist's or artists' main country/area of residence. Any track that had ambiguous origin is not included. We have taken the position of each country's capital city (or the province of the area) by latitude and longitude as the absolute point of origin.

The program MARSYAS[1] was used to extract audio features from the wave files. We used the default MARSYAS settings in single vector format (68 features) to estimate the performance with basic timbral information covering the entire length of each track. No feature weighting or pre-filtering was applied. All features were transformed to have a mean of 0, and a standard deviation of 1. We also investigated the utility of adding chromatic attributes. These describe the notes of the scale being used. This is especially important as a distinguishing feature in geographical ethnomusicology. The chromatic features provided by MARSYAS are 12 per octave - Western tuning, but it may be possible to tell something from how similar to or different from Western tuning the music is.

[1] G. Tzanetakis and P. Cook, "MARSYAS: a framework for audio analysis," *Organised Sound*, vol. 4, pp. 169–175, 2000.

Attribute Information:

The dataset is present in two files, where each file corresponds to a different feature sets.

Both files contain the audio features of 1059 tracks.

In the `default_features_1059_tracks.txt` file, the first 68 columns are audio features of the track, and the last two columns are the origin of the music, represented by latitude and longitude.

In the `default_plus_chromatic_features_1059_tracks.txt` file, the first 116 columns are audio features of the track, and the last two columns are the origin of the music.

Relevant Papers:

The description of music collection and audio features can be found in:

Fang Zhou, Claire Q and Ross. D. King Predicting the Geographical Origin of Music, ICDM, 2014

Citation Request:

The following citation is requested if you use the dataset:

Fang Zhou, Claire Q and Ross. D. King Predicting the Geographical Origin of Music, ICDM, 2014

Ders kapsamında her öğrenciye ayrı ayrı çalışabileceği bir veriseti verilmiştir. Proje kapsamında aşağıda tanımlı işlemlerin gerçekleştirilmesi beklenmektedir. Verisetleri ile ilgili bilgiyi <http://archive.ics.uci.edu/ml/> adresinde bulabilirsiniz. Çalışmanın Jupyter Notebbok ile hazırlanması ve raporun da bu şablonun içinde yazılması gerekmektedir. Ayrıca rapor hazırlamanıza gerek yoktur.

a- Veri Analizi: Veri setinin içeriği, kullanılan özelliklerin anlamı, sınıf sayısı ve isimleri, her sınıfa düşen örnek sayısı gibi bilgilerin çıkarılması gereklidir. Veriseti içindeki özelliklerin sınıflamada ki ayırt ediciliklerine göre sıralanması ve eksik veri varsa giderilmesi yöntemlerinin araştırılması beklenmektedir. Ayrıca Aykırı Değer analizinin de yapılması gerekmektedir.

b-Sınıflama: Sınıflama yöntemlerinden en başarılı olanların araştırılması ve 3 tanesinin seçilerek detaylı incelenmesi, sınıflama başarılarının ve sınıf karışıklık matrislerinin karşılaştırılması gerekmektedir.

c-Kümeleme: Kümeleme yöntemlerinden en başarılı olanların araştırılması ve 3 tanesinin seçilerek detaylı incelenmesi beklenmektedir. Kümeleme sonunda değerlendirme aşamasında örneklerin sınıf etiketlerinin kullanılarak çıkan kümelerin etiketlenmesi ve kümeleme başarılarının karşılaştırılması gerekmektedir.

Değerlendirme:

Proje kapsamında yapılan çalışmalar Jupyter Notebook platformunda (kod ve rapor bir arada) hazırlanarak **19 Aralık 23:59'a** kadar teslim edilmelidir.

Veri Analizi

Veri Setinin İçeriği:

Veri seti, 33 ülkeyi/bölgeyi kapsayan 1059 parçadan oluşan kişisel bir müzik koleksiyonundan oluşturulmuştur. Etkisi küresel olduğu için herhangi bir batı müziği dahil edilmemiştir - aranan şey; müziğin konumu en çok etkileyen yönleridir. Müzik dosyalarından ses özelliklerini çıkarmak için MARSYAS programı kullanılarak her bir müzik parçasının tüm uzunluğunu kapsayan temel timbal (Vurmali çalgılar sınıfında yer alan bir müzik aleti) bilgileriyle performansı tahmin etmek için tek vektör formatındaki (68 özellik) varsayılan MARSYAS ayarları kullanılarak özellikler oluşturulmuştur.

Veri seti son durumda 68'i özellik ve son 2'si enlem ve boylam sınıf etiketi olarak toplam 70 sütun ve her bir müzik parçasını temsil eden 1059 satırдан oluşmaktadır.

```
In [1]:  
import pandas as pd  
import numpy as np  
import matplotlib.pyplot as plt  
import seaborn as sns  
import plotly.express as px  
import plotly.graph_objects as go  
  
from sklearn.feature_selection import mutual_info_classif  
from sklearn.model_selection import train_test_split, cross_val_score  
from sklearn.metrics import classification_report, confusion_matrix, precision_score, recall_score, f1_score, accuracy_score, silhouette_score  
from sklearn.metrics import calinski_harabasz_score, adjusted_rand_score, adjusted_mutual_info_score, homogeneity_score, completeness_score  
from scipy.cluster.hierarchy import dendrogram, linkage, fcluster  
  
from sklearn.ensemble import IsolationForest, RandomForestClassifier, AdaBoostClassifier  
from sklearn.neighbors import LocalOutlierFactor, KNeighborsClassifier  
from sklearn.covariance import EllipticEnvelope  
from sklearn.svm import OneClassSVM, SVC  
from sklearn.linear_model import LogisticRegression, SGDClassifier  
from sklearn.tree import DecisionTreeClassifier  
from sklearn.naive_bayes import GaussianNB  
from sklearn.neural_network import MLPClassifier  
from sklearn.discriminant_analysis import LinearDiscriminantAnalysis, QuadraticDiscriminantAnalysis  
from sklearn.gaussian_process import GaussianProcessClassifier  
from sklearn.cluster import KMeans, AgglomerativeClustering, DBSCAN, SpectralClustering  
  
from sklearn.mixture import GaussianMixture, BayesianGaussianMixture
```

```
In [2]:  
col_names = []  
for i in range(1, 10):  
    col_names.append('f_0' + str(i))  
for i in range(10, 69):  
    col_names.append('f_' + str(i))  
  
col_names.append('lat')  
col_names.append('long')  
  
tracks = pd.read_csv("./GeographicalOriginalOfMusic/default_features_1059_tracks.txt", header = None, names = col_names)  
tracks
```

Out[2]:	f_01	f_02	f_03	f_04	f_05	f_06	f_07	f_08	f_09	f_10	...	f_61	f_62	f_63	f_64
0	7.161286	7.835325	2.911583	0.984049	-1.499546	-2.094097	0.576000	-1.205671	1.849122	-0.425598	...	-1.504263	0.351267	-1.018726	-0.174878
1	0.225763	-0.094169	-0.603646	0.497745	0.874036	0.290280	-0.077659	-0.887385	0.432062	-0.093963	...	-0.495712	-0.465077	-0.157861	-0.157189
2	-0.692525	-0.517801	-0.788035	1.214351	-0.907214	0.880213	0.406899	-0.694895	-0.901869	-1.701574	...	-0.637167	0.147260	0.217914	2.718442
3	-0.735562	-0.684055	2.058215	0.716328	-0.011393	0.805396	1.497982	0.114752	0.692847	0.052377	...	-0.178325	-0.065059	-0.724247	-1.020687
4	0.570272	0.273157	-0.279214	0.083456	1.049331	-0.869295	-0.265858	-0.401676	-0.872639	1.147483	...	-0.919463	-0.667912	-0.820172	-0.190488
...
1054	0.399577	0.310805	-0.039326	-0.111546	0.304586	-0.943453	0.114960	-0.335898	0.826753	-0.393786	...	-0.558717	0.998897	-0.106835	1.526307

1055	1.640386	1.306224	0.192745	-1.816855	-1.311906	-2.128963	-1.875967	0.094232	-1.429742	0.873777	...	0.223143	-0.032425	0.226782	0.182107
1056	-0.772360	-0.670596	-0.840420	-0.832105	0.277346	1.152162	0.241470	0.229092	0.019036	-0.068804	...	0.449239	-0.965270	-0.590039	-0.804297
1057	-0.996965	-1.099395	3.515274	-0.508185	-1.102654	0.192081	0.069821	0.264674	-0.411533	0.501164	...	1.941398	1.769292	0.738616	1.240377
1058	-0.150911	-0.094333	-0.568885	-0.614652	0.332477	-0.954948	-1.527722	-1.591471	-3.678713	-5.930209	...	5.121875	4.103031	3.673086	0.960420

1059 rows × 70 columns

Veri Setinin Sınıf Sayısı ve Her Sınıf İçin Örek Sayısı

Normalde veri setinde 2 farklı sınıf sütunu bulunmakla birlikte aslında tek bir koordinat noktası belirttiği için bunları birleştirip tek bir konum sütunuyla değiştirmek mümkündür. Bunu yapmanın birçok yolu bulunmaktadır;

- İnternette aynı veri setiyle çalışanlardan [Johannes Harmse](#) KMeans ile kümeleme işlemi yapmıştır.
- Yine aynı veri setiyle çalışan [SangeethaThai](#) ise koordinat verilerini kullanarak ülke isim kodlarını [api.geonames.org](#) kullanarak elde etmiş ve sınıf olarak kullanmıştır.

Bu ödevde ise bunlara gerek olmadığı düşünülerek iki sütunun birleştirilmesi ile yetinilmiştir.

In [3]:

```
tracks['location'] = tracks['lat'].astype(str) + ',' + tracks['long'].astype(str)
tracks = tracks.drop(['lat', 'long'], axis = 1)

tracks
```

Out[3]:

	f_01	f_02	f_03	f_04	f_05	f_06	f_07	f_08	f_09	f_10	...	f_60	f_61	f_62	f_63
0	7.161286	7.835325	2.911583	0.984049	-1.499546	-2.094097	0.576000	-1.205671	1.849122	-0.425598	...	-0.043610	-1.504263	0.351267	-1.018726
1	0.225763	-0.094169	-0.603646	0.497745	0.874036	0.290280	-0.077659	-0.887385	0.432062	-0.093963	...	-0.947933	-0.495712	-0.465077	-0.157861
2	-0.692525	-0.517801	-0.788035	1.214351	-0.907214	0.880213	0.406899	-0.694895	-0.901869	-1.701574	...	-0.556109	-0.637167	0.147260	0.217914
3	-0.735562	-0.684055	2.058215	0.716328	-0.011393	0.805396	1.497982	0.114752	0.692847	0.052377	...	0.166616	-0.178325	-0.065059	-0.724247
4	0.570272	0.273157	-0.279214	0.083456	1.049331	-0.869295	-0.265858	-0.401676	-0.872639	1.147483	...	-0.500785	-0.919463	-0.667912	-0.820172
...
1054	0.399577	0.310805	-0.039326	-0.111546	0.304586	-0.943453	0.114960	-0.335898	0.826753	-0.393786	...	0.425577	-0.558717	0.998897	-0.106835
1055	1.640386	1.306224	0.192745	-1.816855	-1.311906	-2.128963	-1.875967	0.094232	-1.429742	0.873777	...	0.723125	0.223143	-0.032425	0.226782
1056	-0.772360	-0.670596	-0.840420	-0.832105	0.277346	1.152162	0.241470	0.229092	0.019036	-0.068804	...	-0.287753	0.449239	-0.965270	-0.590039
1057	-0.996965	-1.099395	3.515274	-0.508185	-1.102654	0.192081	0.069821	0.264674	-0.411533	0.501164	...	1.565493	1.941398	1.769292	0.738616
1058	-0.150911	-0.094333	-0.568885	-0.614652	0.332477	-0.954948	-1.527722	-1.591471	-3.678713	-5.930209	...	4.159140	5.121875	4.103031	3.673086

1059 rows × 69 columns

In [4]:

```
track_counts_location = tracks['location'].value_counts().sort_values(ascending = False)
track_counts_location.head(10)
```

Out[4]:

28.61,77.2	69
12.65,-8.0	66
39.91,32.83	64
41.9,12.48	51
33.66,73.16	47
38.0,23.71	46
39.91,116.38	40
41.26,69.21	36
-15.75,-47.95	36
36.7,3.21	35

Name: location, dtype: int64

Veriseti İçindeki Özelliklerin Sınıflamadaki Ayırt Ediciliklerine Göre Sıralanması

Sklearn kütüphanesi kullanılarak [Mutual Information](#) metoduyla özelliklerin ayırt edicilikleri sıralanmıştır. Bulunan sonuç [matplotlib](#) kullanılarak grafikleştirilmiştir.

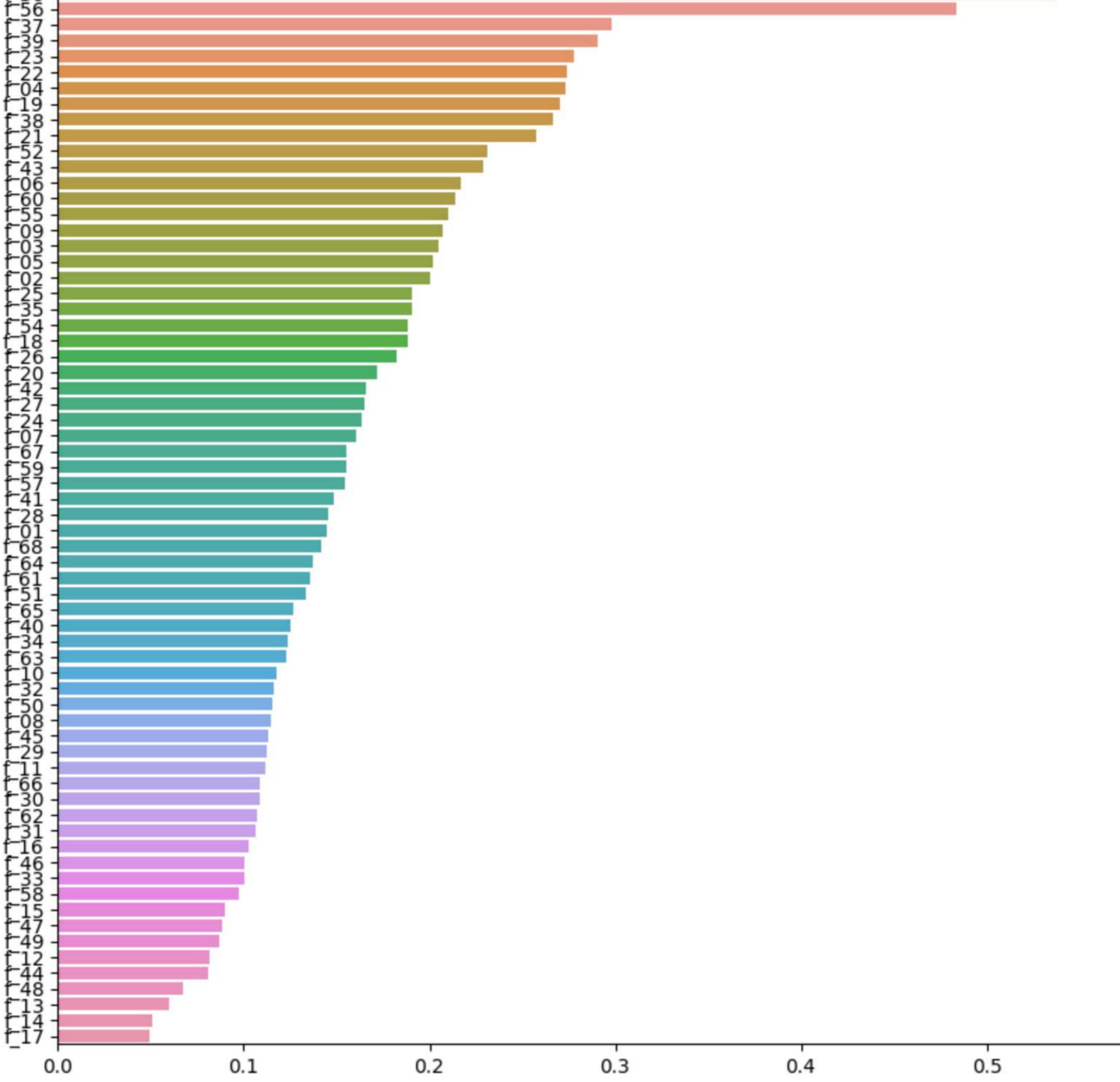
In [5]:

```
x = tracks.drop(['location'], axis = 1)
y = tracks['location']

mi = mutual_info_classif(x, y)
mi = pd.Series(mi)
mi.index = X.columns
mi.sort_values(ascending = False, inplace = True)

plt.figure(figsize = (10, 10))
sns.barplot(x = mi, y = mi.index)
plt.title('Mutual Information')
plt.show()
```

Mutual Information



Aykırı Değer Analizi

Aykırı değer analizi için sklearn kütüphanesinden [Isolation Forest](#), [Local Outlier Factor](#), [EllipticEnvelope](#) ve [OneClassSVM](#) metotları kullanılarak aykırı değerler analiz edilerek metotların bulduğu değerler tablo olarak birbirleri ile karşılaştırılmak üzere tablolaştırılmıştır.

In [6]:

```

clf = IsolationForest(max_samples = 100, random_state = 42)
clf.fit(X)
y_noano = clf.predict(X)
y_noano = pd.DataFrame(y_noano, columns = ['Top'])
y_noano[y_noano['Top'] == 1].index.values

x_outliers_isf = X.iloc[y_noano[y_noano['Top'] == -1].index.values]

clf = LocalOutlierFactor(n_neighbors = 20, contamination = 0.1)
y_noano = clf.fit_predict(X)
y_noano = pd.DataFrame(y_noano, columns = ['Top'])
y_noano[y_noano['Top'] == 1].index.values

x_outliers_lof = X.iloc[y_noano[y_noano['Top'] == -1].index.values]

clf = EllipticEnvelope(contamination = 0.1)
y_noano = clf.fit_predict(X)
y_noano = pd.DataFrame(y_noano, columns = ['Top'])
y_noano[y_noano['Top'] == 1].index.values

x_outliers_ee = X.iloc[y_noano[y_noano['Top'] == -1].index.values]

clf = OneClassSVM(nu = 0.95 * 0.05)
y_noano = clf.fit_predict(X)
y_noano = pd.DataFrame(y_noano, columns = ['Top'])
y_noano[y_noano['Top'] == 1].index.values

x_outliers_ocsvm = X.iloc[y_noano[y_noano['Top'] == -1].index.values]
x_outliers = X.iloc[y_noano[y_noano['Top'] == 1].index.values]

# to drop outliers uncomment the following 2 lines
# X = X.iloc[y_noano[y_noano['Top'] == 1].index.values]
# y = y.iloc[y_noano[y_noano['Top'] == 1].index.values]

x_outliers = pd.concat([x_outliers_isf, x_outliers_lof, x_outliers_ee, x_outliers_ocsvm], axis = 0)
x_outliers = x_outliers.drop_duplicates()

```

```

print('Number of outliers predicted by isf: {}'.format(X_outliers_isf.shape[0]))
print('Number of outliers predicted by lof: {}'.format(X_outliers_lof.shape[0]))
print('Number of outliers predicted by ee: {}'.format(X_outliers_ee.shape[0]))
print('Number of outliers predicted by ocsvm: {}'.format(X_outliers_ocsvm.shape[0]))
print('Number of outliers predicted by all algorithms: {}'.format(X_outliers.shape[0]))
print('Number of rows without outliers: {}'.format(X.shape[0]))

```

```

Number of outliers predicted by isf: 90
Number of outliers predicted by lof: 106
Number of outliers predicted by ee: 106
Number of outliers predicted by ocsvm: 73
Number of outliers predicted by all algorithms: 197
Number of rows without outliers: 1059

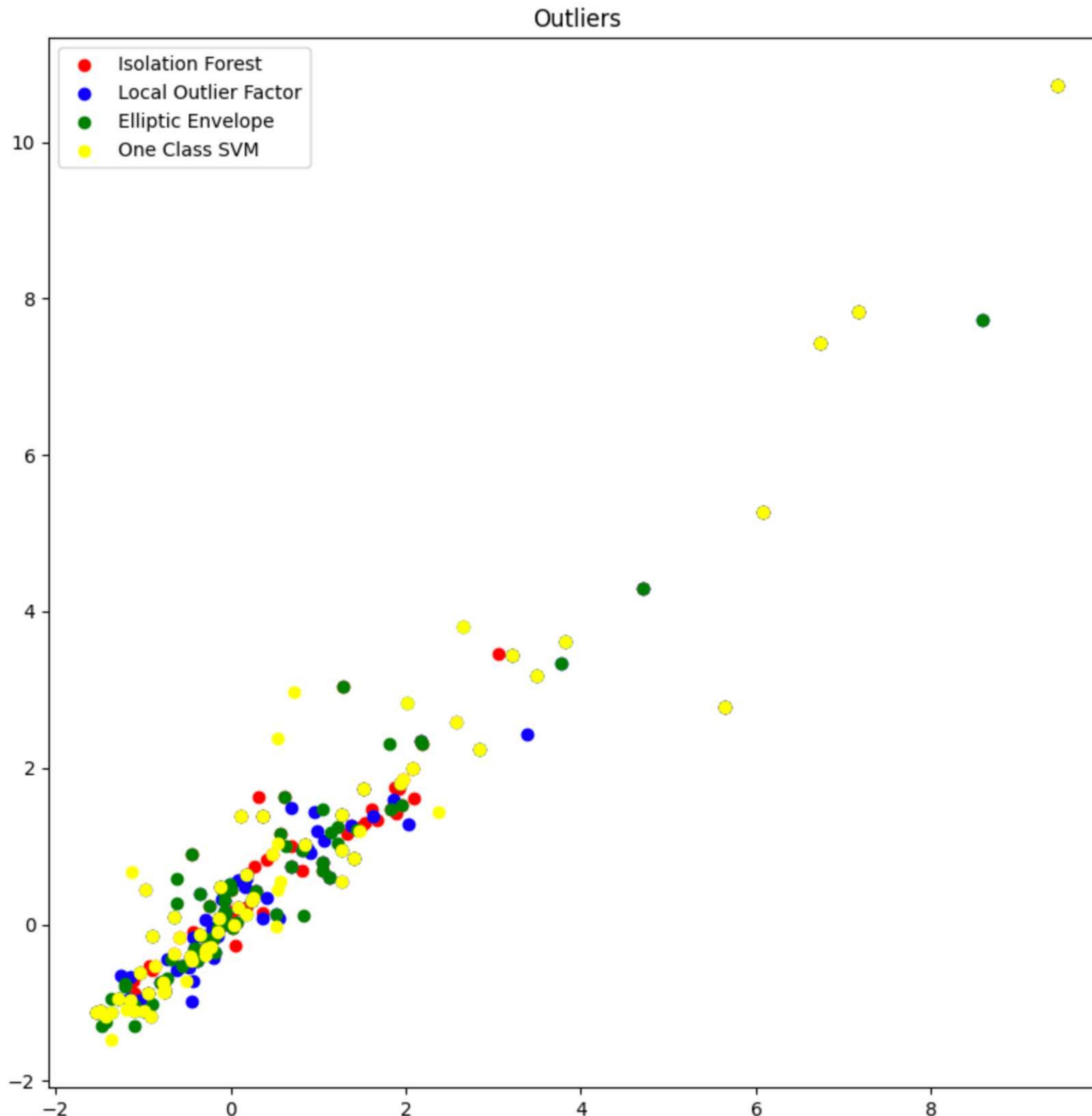
```

In [7]:

```

plt.figure(figsize = (10, 10))
plt.scatter(X_outliers_isf['f_01'], X_outliers_isf['f_02'], color = 'red', label = 'Isolation Forest')
plt.scatter(X_outliers_lof['f_01'], X_outliers_lof['f_02'], color = 'blue', label = 'Local Outlier Factor')
plt.scatter(X_outliers_ee['f_01'], X_outliers_ee['f_02'], color = 'green', label = 'Elliptic Envelope')
plt.scatter(X_outliers_ocsvm['f_01'], X_outliers_ocsvm['f_02'], color = 'yellow', label = 'One Class SVM')
plt.title('Outliers')
plt.legend()
plt.show()

```



Sınıflama

En yaygın sınıflandırma algoritmalarından bazıları şunlardır:

Lojistik Regresyon: Bu, ikili sınıflandırma problemleri (iki olası sonucu olan problemler) için yaygın olarak kullanılan doğrusal bir modeldir.

Karar Ağaçları: Bu, verilerin özelliklerine dayalı olarak ağaç benzeri bir karar yapısı oluşturarak çalışan basit, yorumlanabilir bir modeldir.

Random Forests: Bu, daha doğru ve kararlı bir model oluşturmak için birden fazla karar ağacının tahminlerini birleştiren bir topluluk yöntemidir.

Destek Vektör Makineleri (SVM'ler): Bu, farklı sınıfları maksimum düzeyde ayıran yüksek boyutlu bir uzayda hiperdüzlemi bularak çalışan güçlü ve esnek bir modeldir.

K-En Yakın Komşular (KNN): Bu, eğitim verilerinde test veri noktasına en çok benzeyen K gözlemlerini bularak ve bu K gözlemlerinin çoğuluk sınıfını kullanarak çalışan basit, parametrik olmayan bir modeldir.

Naive Bayes: Veriler verildiğinde belirli olayların meydana gelme olasılığına dayalı olarak tahminler yapan olasılıksal bir modeldir.

Gradyan Yükseltme, AdaBoost, Stokastik Gradyan İnişi (SGD), Lineer Diskriminant Analizi (LDA), İkinci Dereceden Ayrım Analizi (QDA), Doğrusal Regresyon, Sırt Regresyonu,, Kement Regresyonu gibi daha bir çok algoritma veri biliminde yaygın olarak kullanılmaktadır. Bu algoritmaların her birinin kendi güçlü ve zayıf yönleri vardır ve belirli bir problem için en iyi olan özel algoritma, verilerin özelliklerine ve analizin hedeflerine bağlı olacaktır.

In [8]:

```
def visualize_cm(cm, title: str):
    """Visualize confusion matrix."""
    plt.figure(figsize = (10, 10))
    sns.heatmap(cm, annot = True, fmt = 'd', cmap = 'Blues')
    plt.title('Confusion matrix for {}'.format(title))
    plt.ylabel('True label')
    plt.xlabel('Predicted label')
    plt.show()

def visualize_cr(cr: dict, title: str):
    """Visualize classification report."""
    plt.figure(figsize = (10, 10))
    sns.heatmap(pd.DataFrame(cr).T, annot = True, cmap = 'Blues')
    plt.title('Classification report for {}'.format(title))
    plt.show()
```

In [9]:

```
lr = LogisticRegression(max_iter = 1000)
dt = DecisionTreeClassifier()
rf = RandomForestClassifier()
svm = SVC()
knn = KNeighborsClassifier()
nb = GaussianNB()
nn = MLPClassifier(max_iter = 1000)
ab = AdaBoostClassifier()
sgd = SGDClassifier()
lda = LinearDiscriminantAnalysis()
qda = QuadraticDiscriminantAnalysis()
gpc = GaussianProcessClassifier()

models = {'Logistic Regression': {'model': lr},
          'Decision Tree Classifier': {'model': dt},
          'Random Forest Classifier': {'model': rf},
          'Support Vector Machines': {'model': svm},
          'K-Nearest Neighbors': {'model': knn},
          'Naive Bayes': {'model': nb},
          'Neural Networks': {'model': nn},
          'AdaBoost': {'model': ab},
          'Stochastic Gradient Descent': {'model': sgd},
          'Linear Discriminant Analysis': {'model': lda},
          'Quadratic Discriminant Analysis': {'model': qda},
          'Gaussian Process Classification': {'model': gpc'}
         }
```

In []:

In [10]:

```
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, random_state = 42)

# train the models
for name, model_dict in models.items():
    model_dict['model'].fit(x_train, y_train)
    y_pred = model_dict['model'].predict(x_test)

    accuracy_score_of = accuracy_score(y_test, y_pred)
    print('Accuracy of {} on test set: {:.2f}'.format(name, accuracy_score_of))

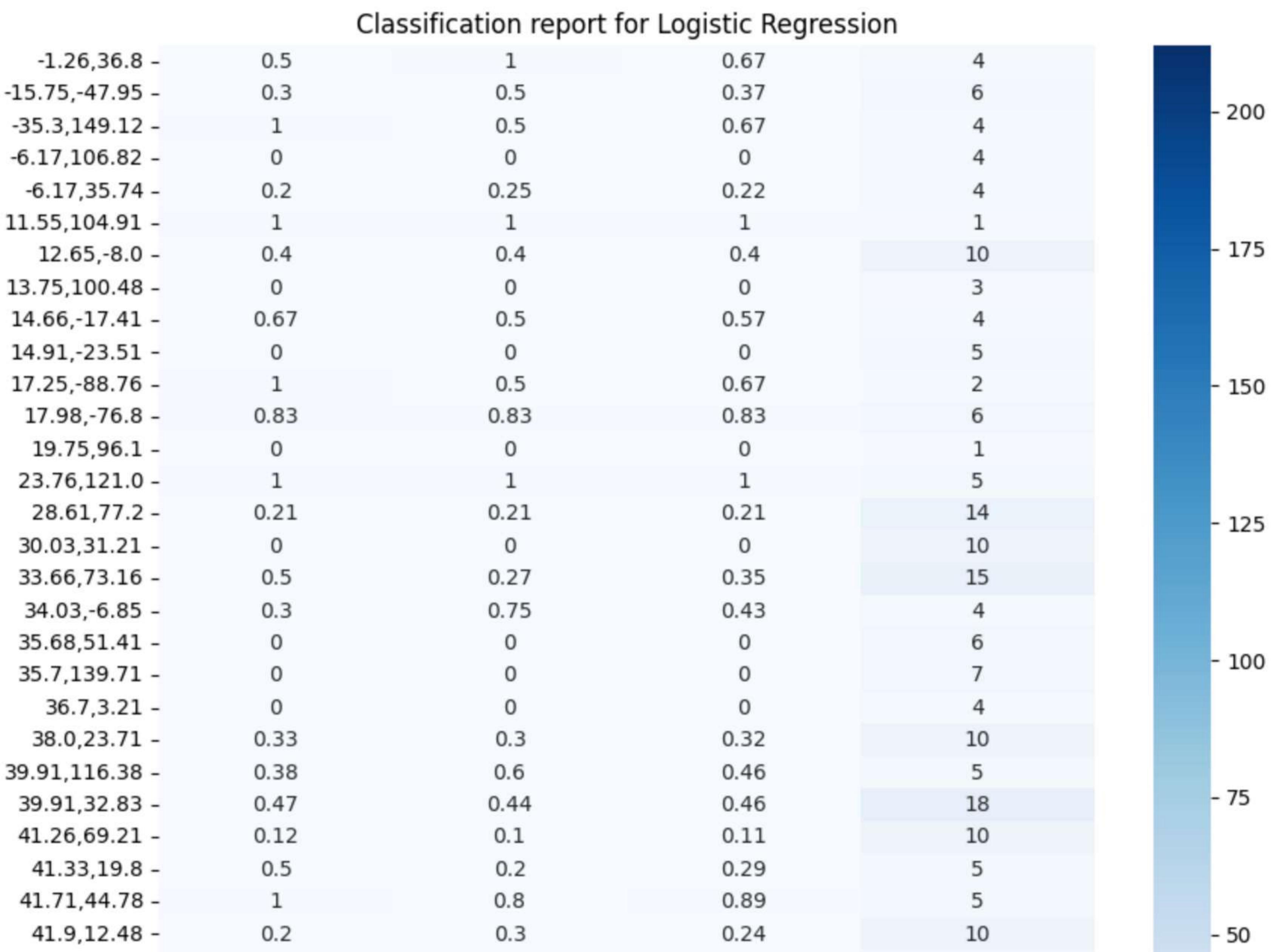
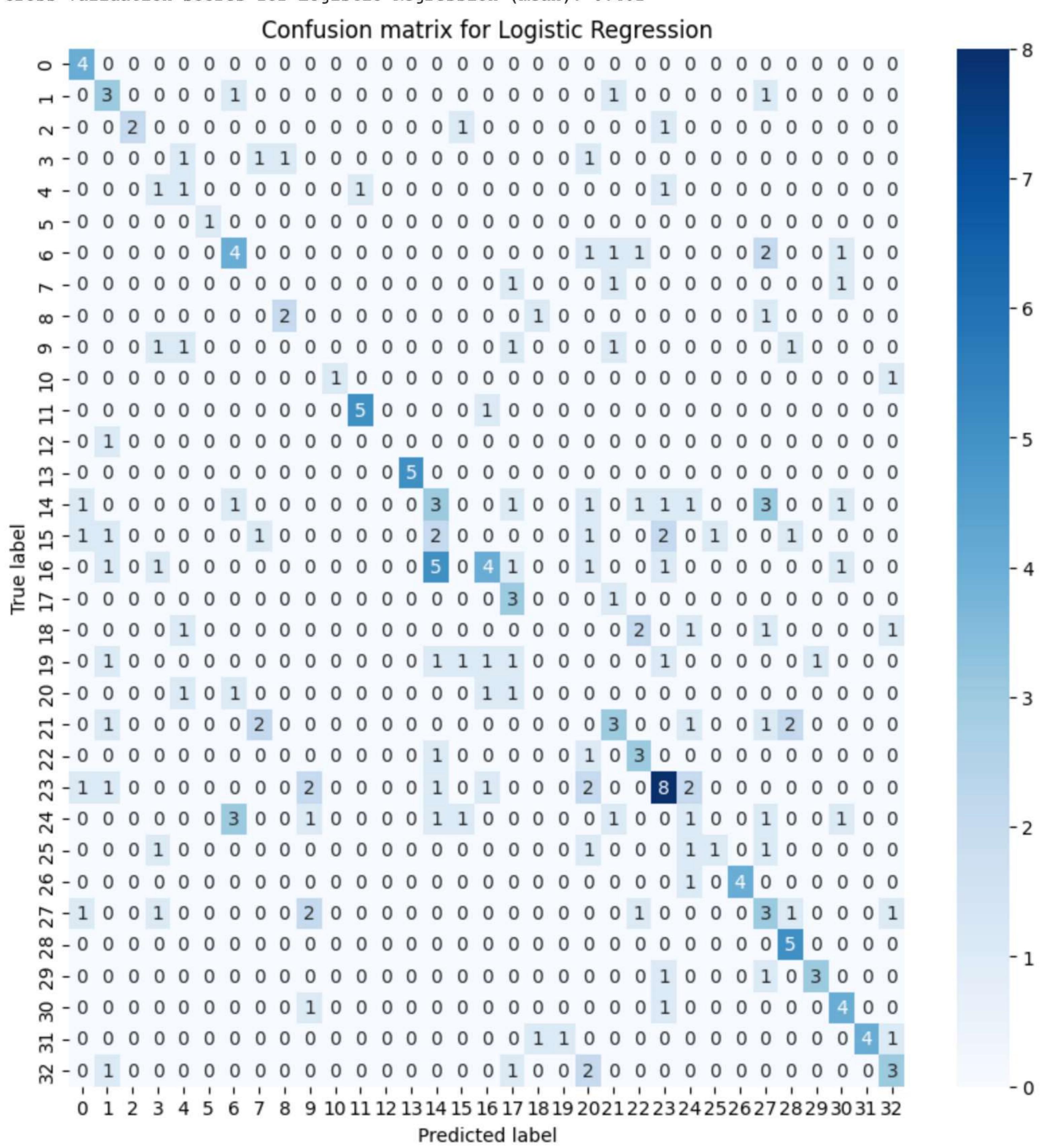
    cv_scores = cross_val_score(model_dict['model'], X, y, cv=10)
    print('Cross Validation Scores for {}: {}'.format(name, cv_scores))
    print('Cross Validation Scores for {} (mean): {:.3f}'.format(name, np.mean(cv_scores)))

    cm = confusion_matrix(y_test, y_pred)
    visualize_cm(cm, name)

    cr = classification_report(y_test, y_pred, output_dict = True, zero_division = 0)
    visualize_cr(cr, name)

    model_dict['cr'] = cr
    model_dict['accuracy_score'] = accuracy_score_of
    model_dict['cv_scores'] = cv_scores
    model_dict['cm'] = cm
```

Accuracy of Logistic Regression on test set: 0.38
Cross Validation Scores for Logistic Regression: [0.38679245 0.38679245 0.40566038 0.39622642 0.4245283 0.4245283
0.38679245 0.40566038 0.37735849 0.42857143]
Cross Validation Scores for Logistic Regression (mean): 0.402

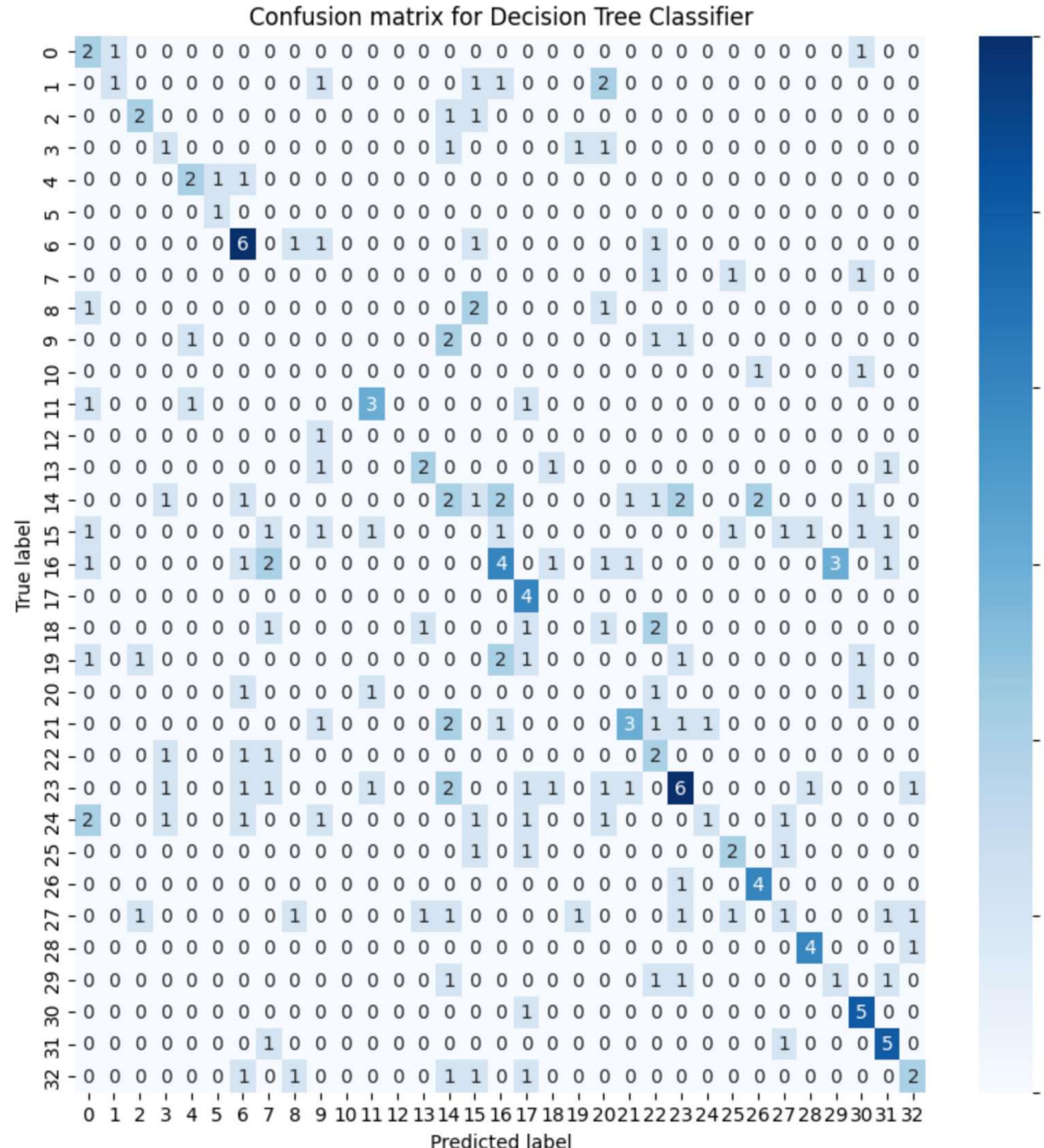


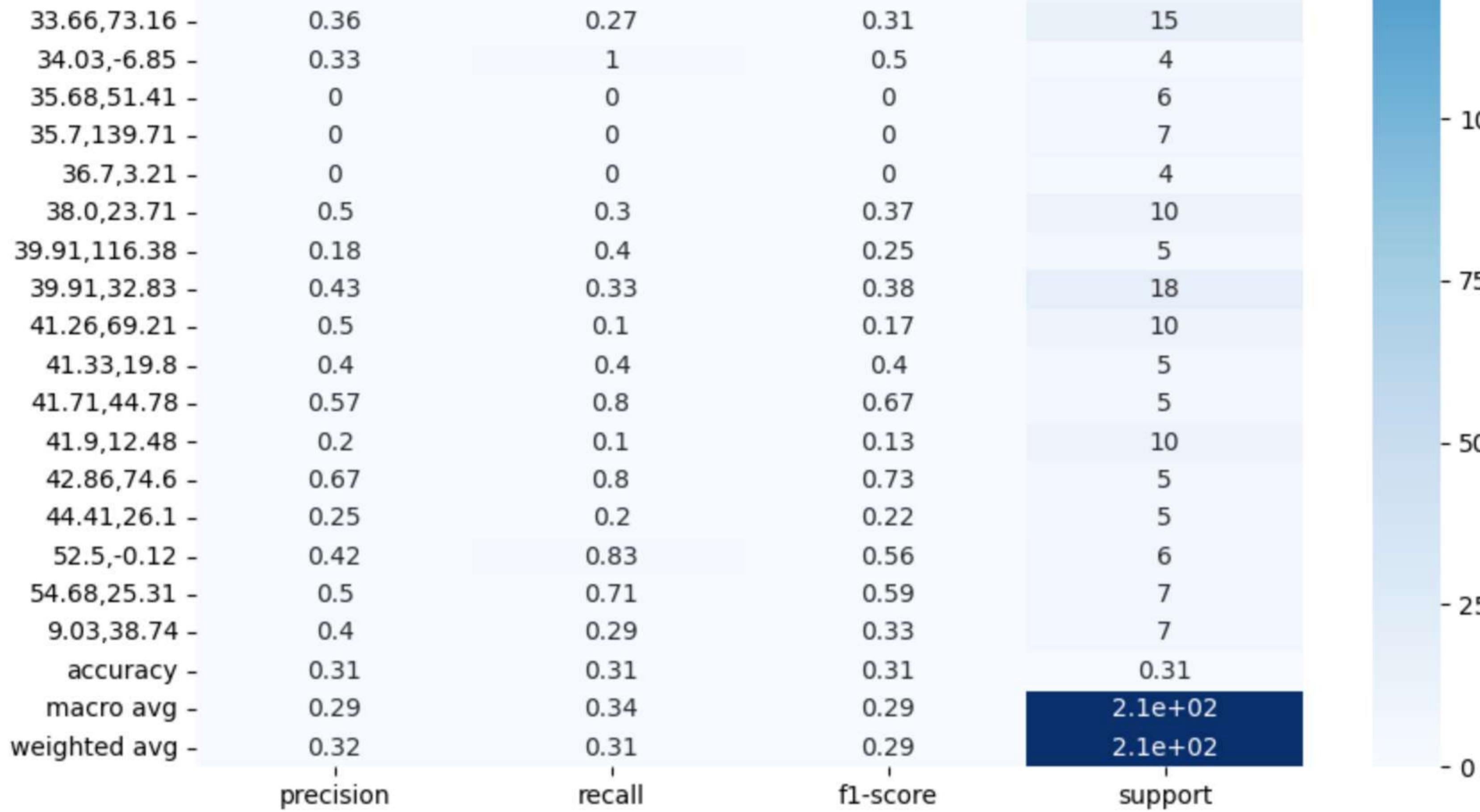
42.86,74.6 -	0.5	1	0.67	5
44.41,26.1 -	0.75	0.6	0.67	5
52.5,-0.12 -	0.44	0.67	0.53	6
54.68,25.31 -	1	0.57	0.73	7
9.03,38.74 -	0.43	0.43	0.43	7
accuracy -	0.38	0.38	0.38	0.38
macro avg -	0.43	0.42	0.4	2.1e+02
weighted avg -	0.39	0.38	0.37	2.1e+02
	precision	recall	f1-score	support

Accuracy of Decision Tree Classifier on test set: 0.31

Cross Validation Scores for Decision Tree Classifier: [0.21698113 0.21698113 0.22641509 0.27358491 0.32075472 0.28301887 0.19811321 0.21698113 0.22641509 0.23809524]

Cross Validation Scores for Decision Tree Classifier (mean): 0.242



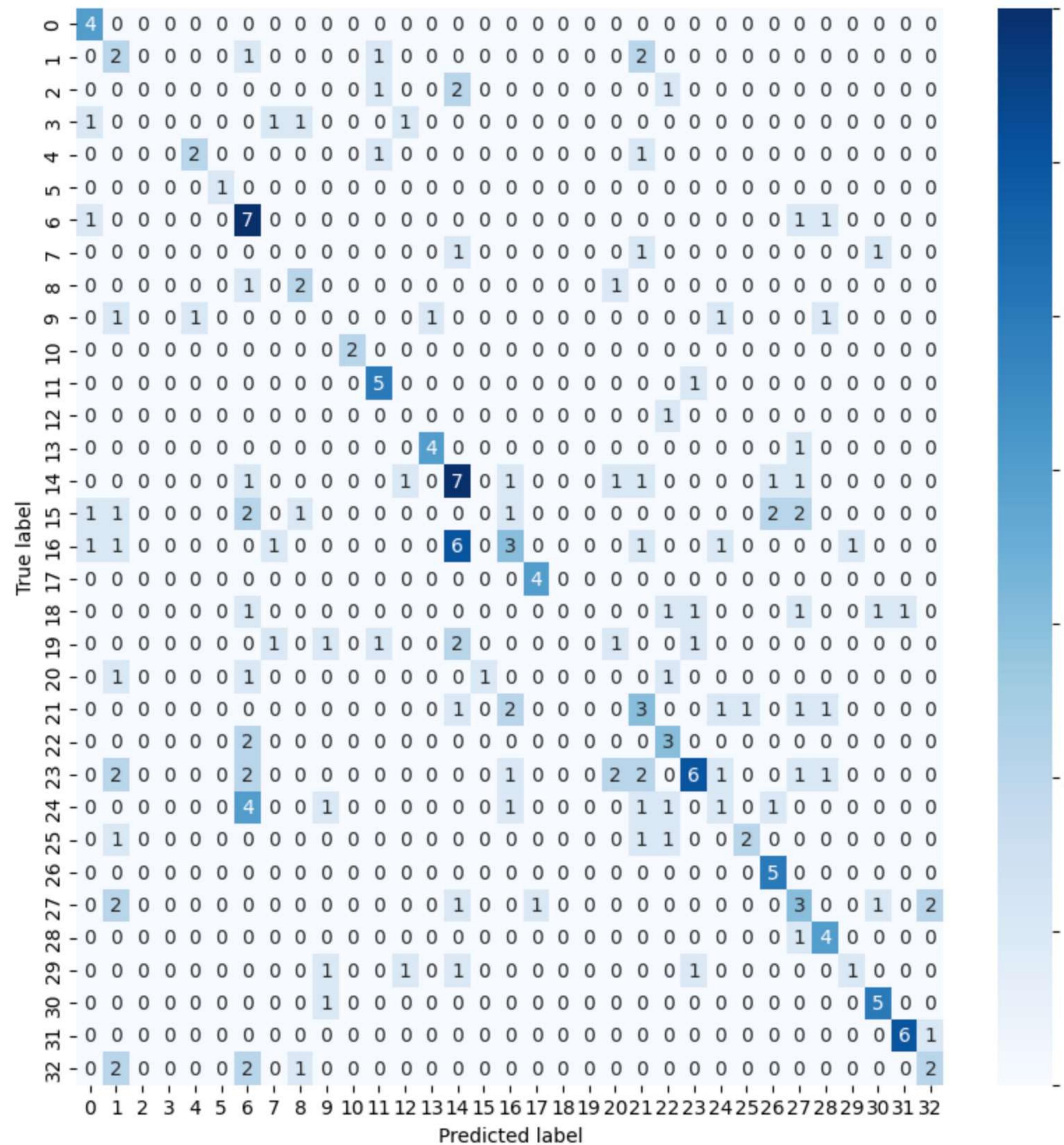


Accuracy of Random Forest Classifier on test set: 0.40

Cross Validation Scores for Random Forest Classifier: [0.4245283 0.39622642 0.49056604 0.47169811 0.54716981 0.52830189
0.38679245 0.46226415 0.4245283 0.4952381]

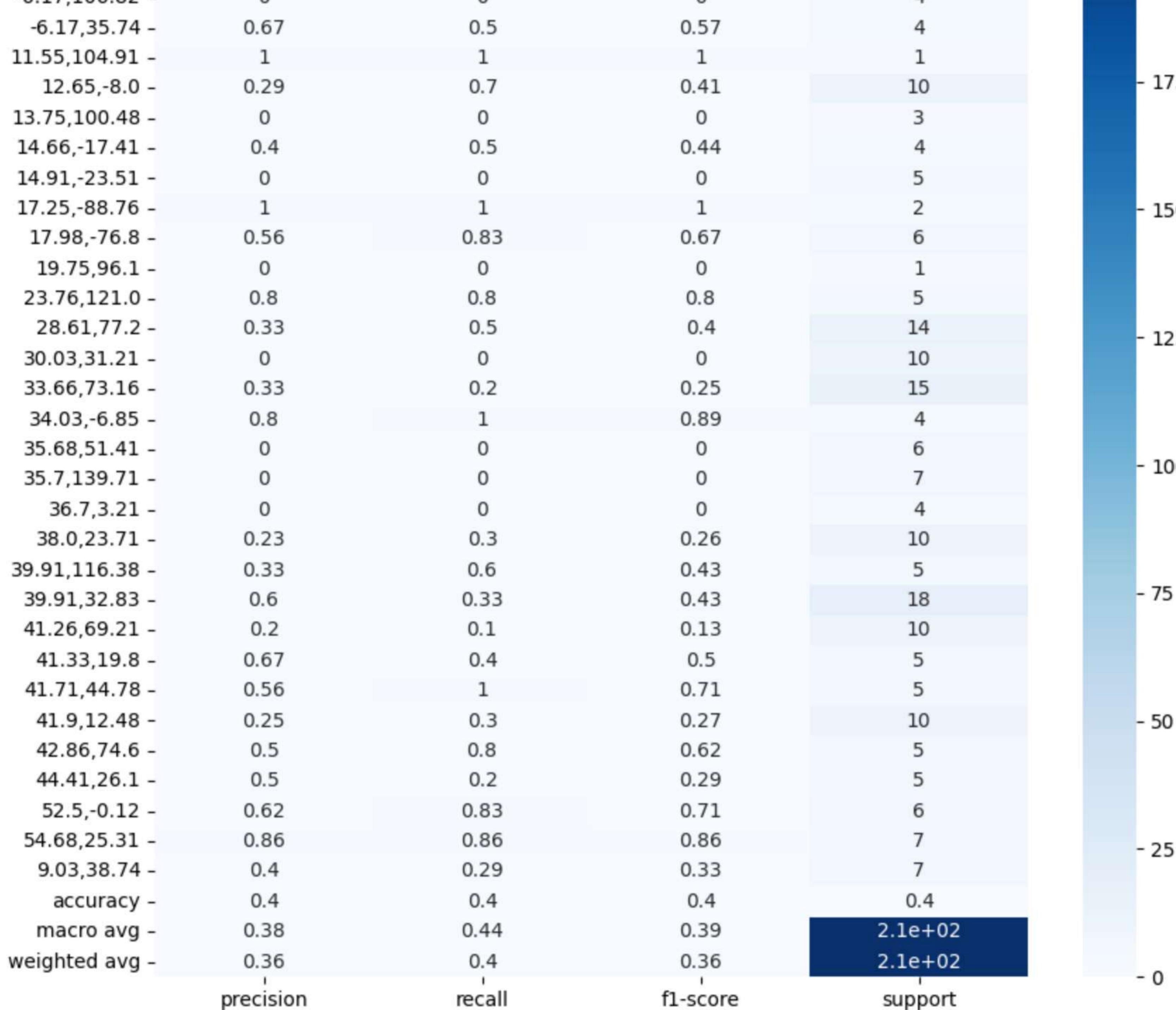
Cross Validation Scores for Random Forest Classifier (mean): 0.463

Confusion matrix for Random Forest Classifier



Classification report for Random Forest Classifier

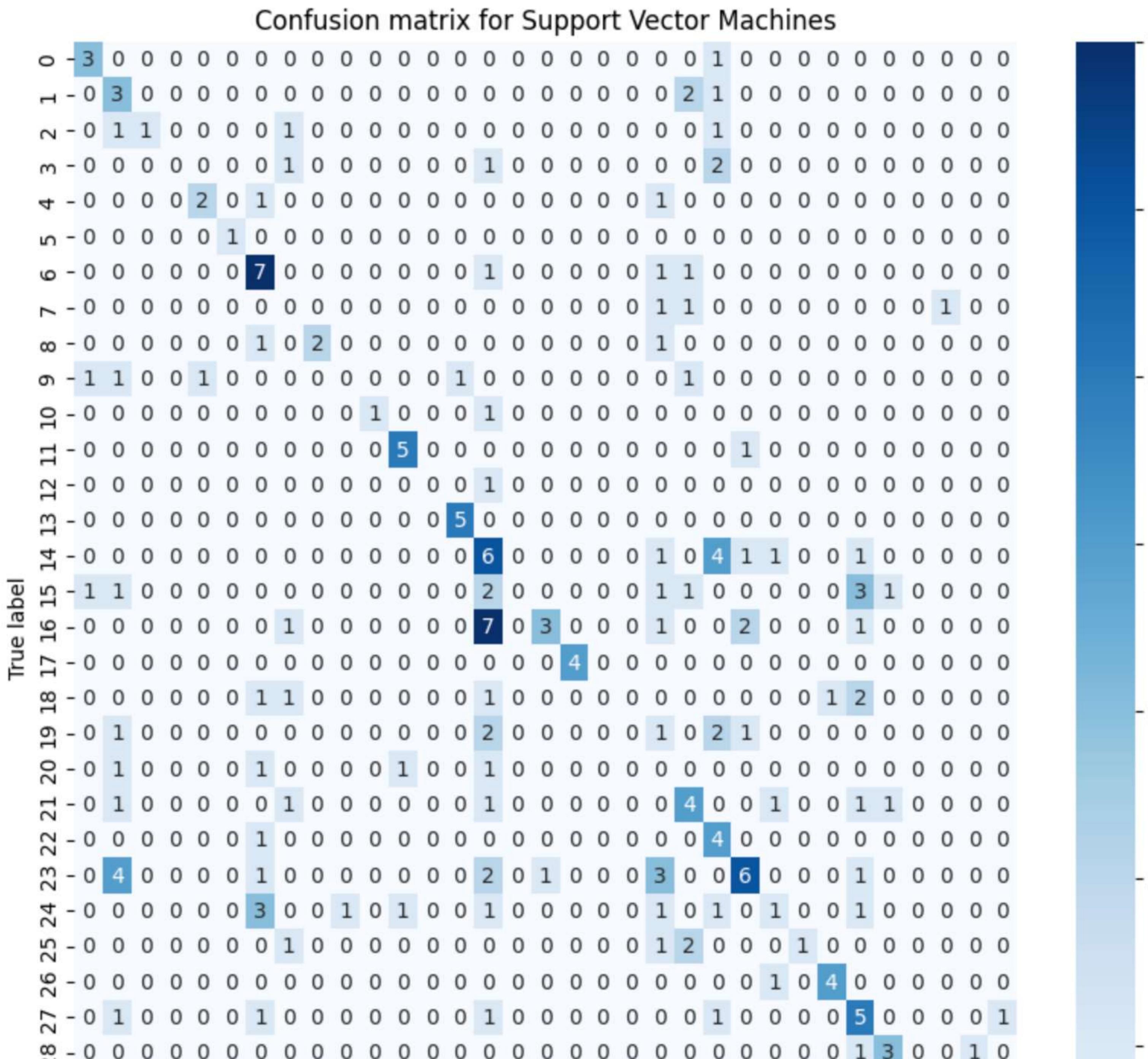


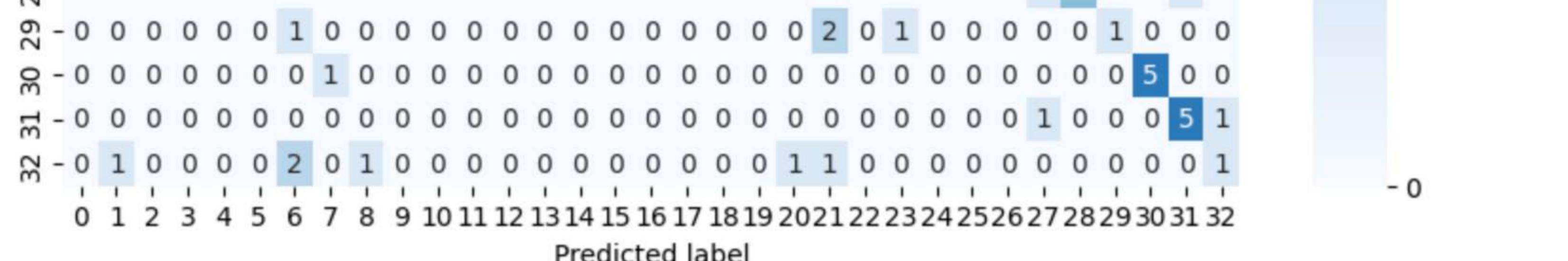


Accuracy of Support Vector Machines on test set: 0.39

Cross Validation Scores for Support Vector Machines: [0.3490566 0.40566038 0.44339623 0.44339623 0.49056604 0.46226415 0.35849057 0.4245283 0.33962264 0.4]

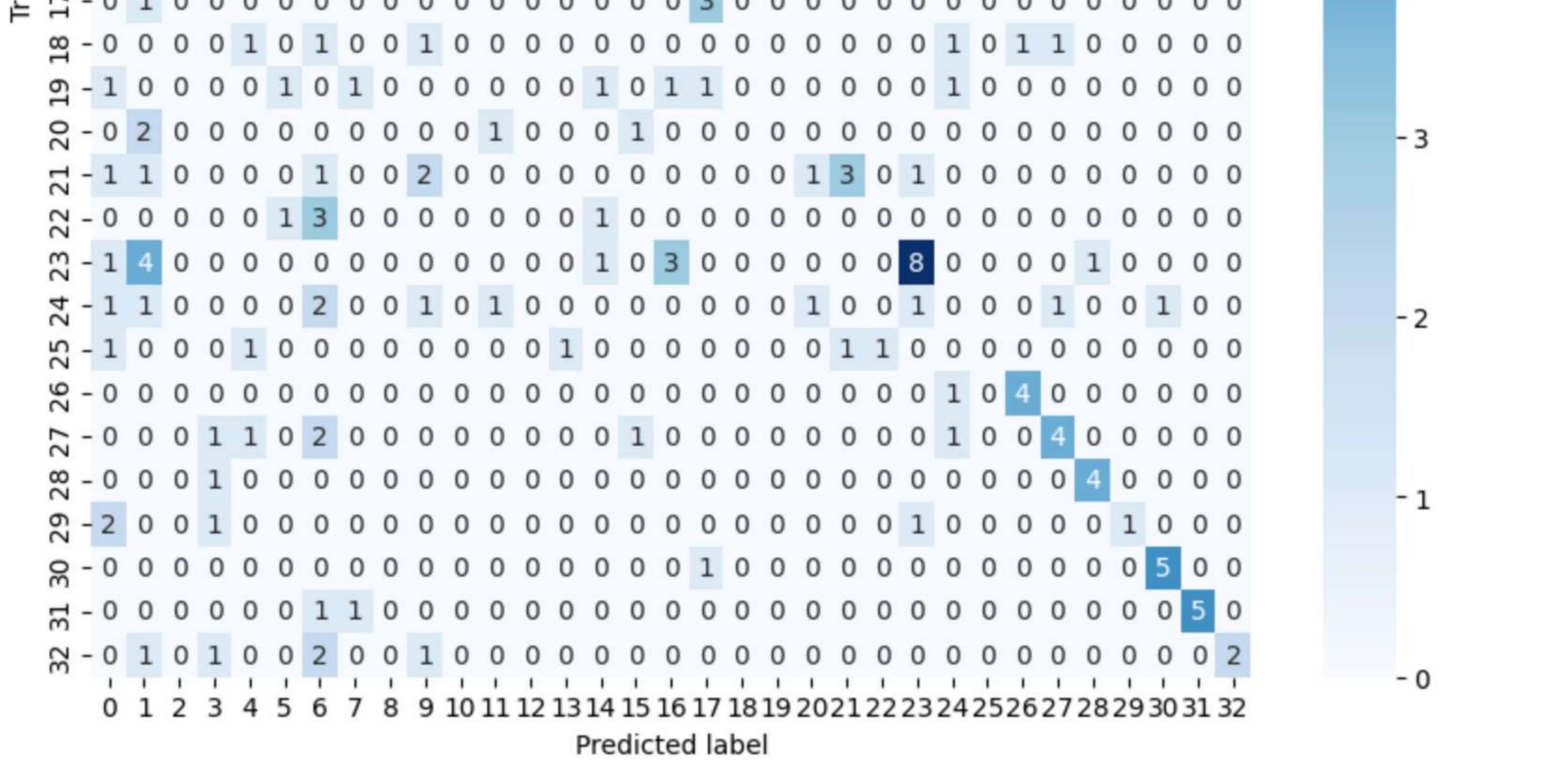
Cross Validation Scores for Support Vector Machines (mean): 0.412





Classification report for Support Vector Machines

	precision	recall	f1-score	support
-1.26,36.8 -	0.6	0.75	0.67	4
-15.75,-47.95 -	0.2	0.5	0.29	6
-35.3,149.12 -	1	0.25	0.4	4
-6.17,106.82 -	0	0	0	4
-6.17,35.74 -	0.67	0.5	0.57	4
11.55,104.91 -	1	1	1	1
12.65,-8.0 -	0.35	0.7	0.47	10
13.75,100.48 -</td				



Classification report for K-Nearest Neighbors

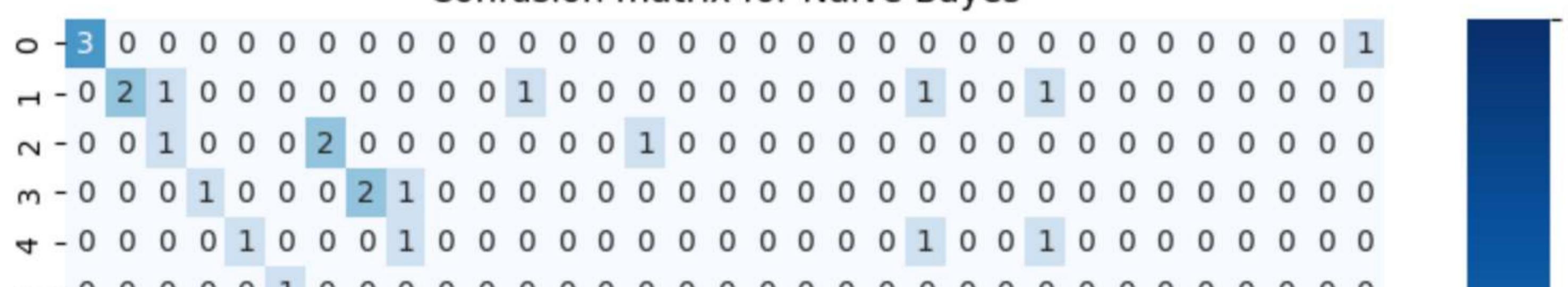
	precision	recall	f1-score	support	
-1.26,36.8 -	0.25	1	0.4	4	- 200
-15.75,-47.95 -	0.067	0.17	0.095	6	
-35.3,149.12 -	0.67	0.5	0.57	4	
-6.17,106.82 -	0	0	0	4	
-6.17,35.74 -	0.2	0.5	0.29	4	
11.55,104.91 -	0.2	1	0.33	1	- 175
12.65,-8.0 -	0.32	0.7	0.44	10	
13.75,100.48 -	0	0	0	3	
14.66,-17.41 -	1	0.5	0.67	4	
14.91,-23.51 -	0	0	0	5	
17.25,-88.76 -	1	0.5	0.67	2	- 150
17.98,-76.8 -	0.44	0.67	0.53	6	
19.75,96.1 -	0	0	0	1	
23.76,121.0 -	0.62	1	0.77	5	
28.61,77.2 -	0.31	0.29	0.3	14	- 125
30.03,31.21 -	0	0	0	10	
33.66,73.16 -	0.29	0.13	0.18	15	
34.03,-6.85 -	0.43	0.75	0.55	4	
35.68,51.41 -	0	0	0	6	- 100
35.7,139.71 -	0	0	0	7	
36.7,3.21 -	0	0	0	4	
38.0,23.71 -	0.33	0.3	0.32	10	
39.91,116.38 -	0	0	0	5	- 75
39.91,32.83 -	0.38	0.44	0.41	18	
41.26,69.21 -	0	0	0	10	
41.33,19.8 -	0	0	0	5	
41.71,44.78 -	0.8	0.8	0.8	5	
41.9,12.48 -	0.57	0.4	0.47	10	- 50
42.86,74.6 -	0.67	0.8	0.73	5	
44.41,26.1 -	1	0.2	0.33	5	
52.5,-0.12 -	0.71	0.83	0.77	6	
54.68,25.31 -	1	0.71	0.83	7	- 25
9.03,38.74 -	1	0.29	0.44	7	
accuracy -	0.35	0.35	0.35	0.35	
macro avg -	0.37	0.38	0.33	2.1e+02	
weighted avg -	0.36	0.35	0.32	2.1e+02	- 0

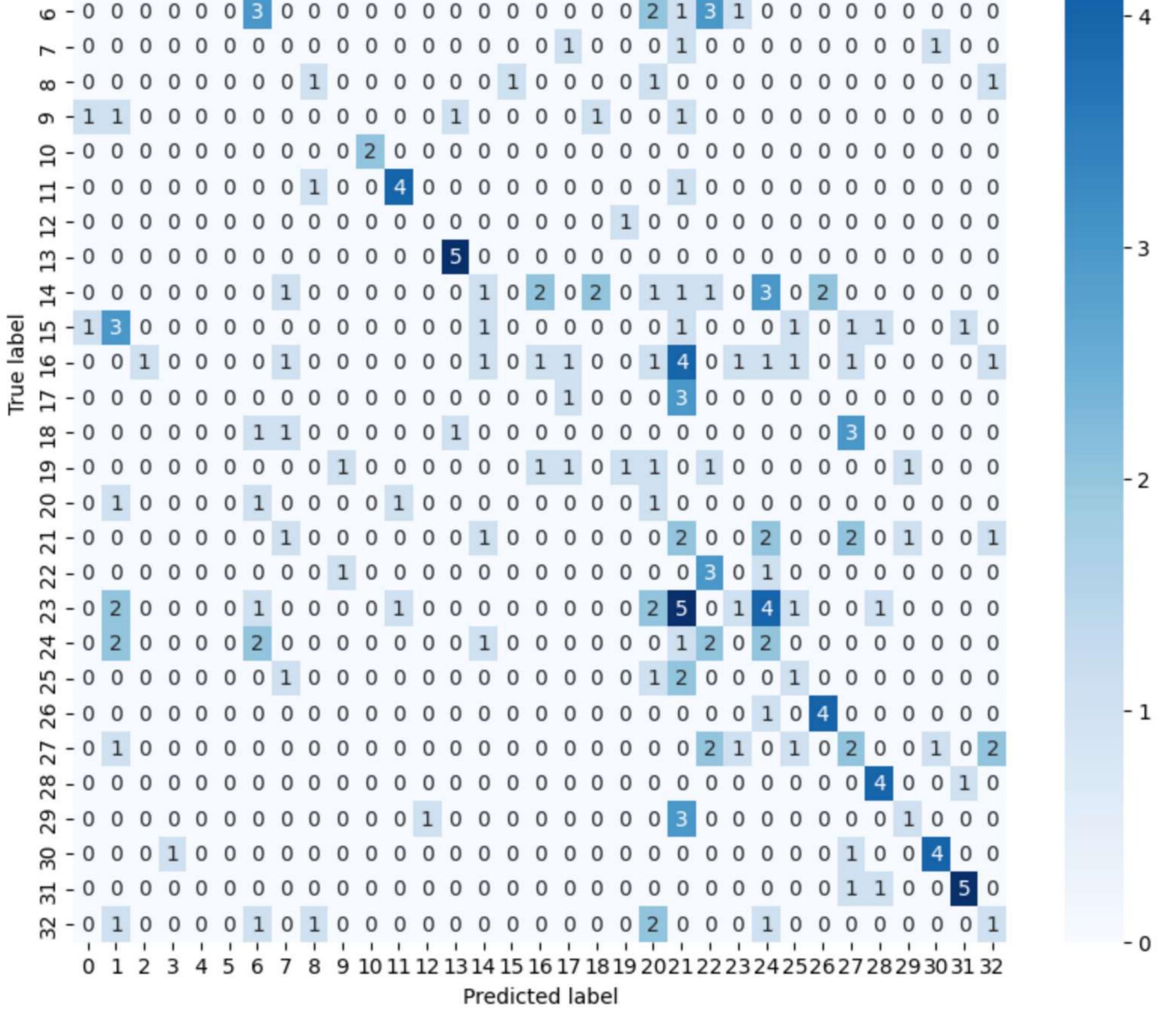
Accuracy of Naive Bayes on test set: 0.28

```
Cross Validation Scores for Naive Bayes: [0.31132075 0.32075472 0.33018868 0.33962264 0.36792453 0.4150943  
0.27358491 0.32075472 0.22641509 0.3047619]
```

Cross Validation Scores for Naive Bayes (mean): 0.321

Confusion matrix for Naive Bayes

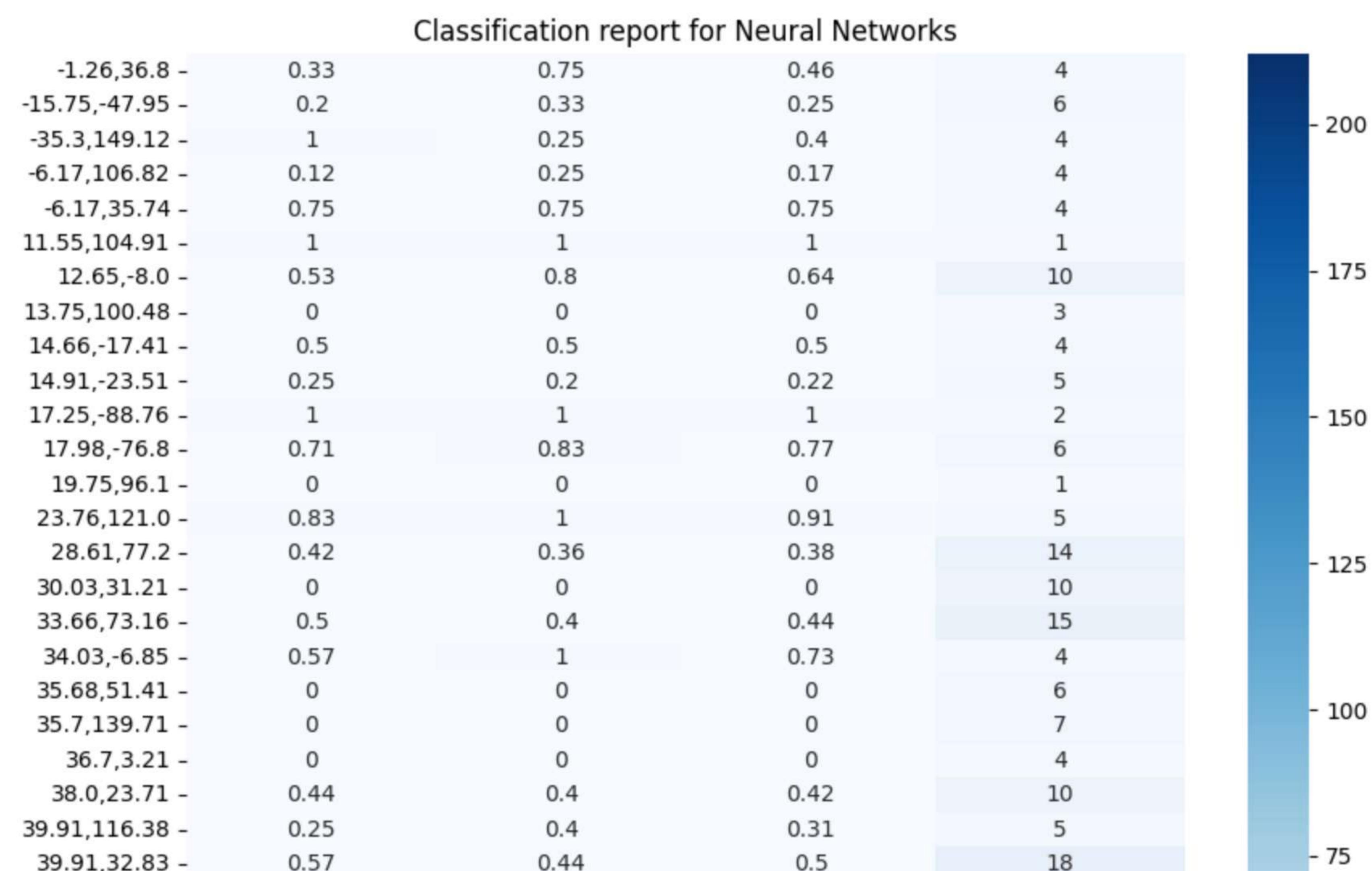
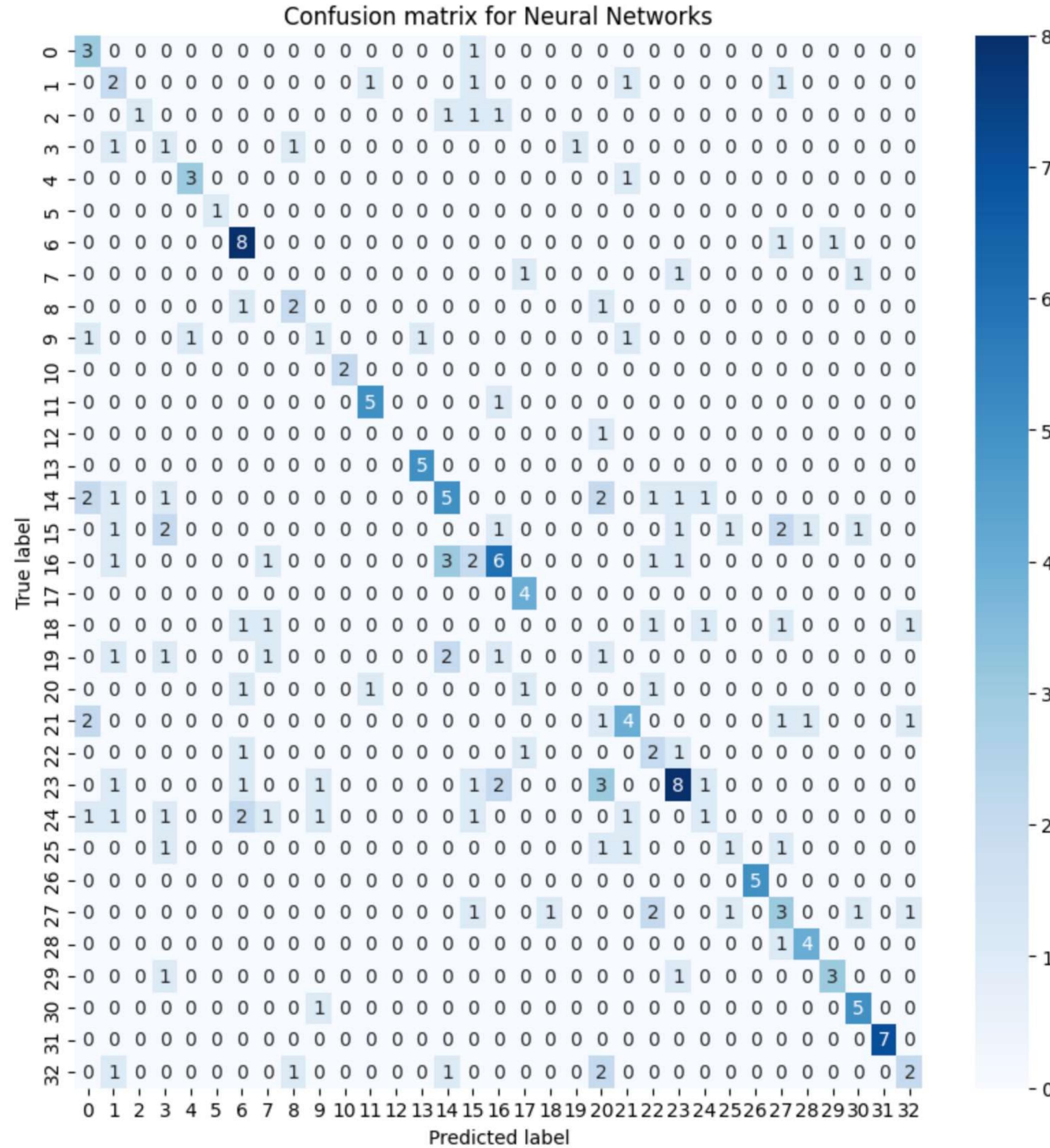




Classification report for Naive Bayes

	precision	recall	f1-score	support
-1.26,36.8 -	0.6	0.75	0.67	4
-15.75,-47.95 -	0.15	0.33	0.21	6
-35.3,149.12 -	0.33	0.25	0.29	4
-6.17,106.82 -	0.5	0.25	0.33	4
-6.17,35.74 -	1	0.25	0.4	4
11.55,104.91 -	1	1	1	1
12.65,-8.0 -	0.27	0.3	0.29	10
13.75,100.48 -	0	0	0	3
14.66,-17.41 -	0.2	0.25	0.22	4
14.91,-23.51 -	0	0	0	5
17.25,-88.76 -	1	1	1	2
17.98,-76.8 -	0.57	0.67	0.62	6
19.75,96.1 -	0	0	0	1
23.76,121.0 -	0.71	1	0.83	5
28.61,77.2 -	0.17	0.071	0.1	14
30.03,31.21 -	0	0	0	10
33.66,73.16 -	0.25	0.067	0.11	15
34.03,-6.85 -	0.25	0.25	0.25	4
35.68,51.41 -	0	0	0	6
35.7,139.71 -	0.5	0.14	0.22	7
36.7,3.21 -	0.083	0.25	0.12	4
38.0,23.71 -	0.071	0.2	0.11	10
39.91,116.38 -	0.25	0.6	0.35	5
39.91,32.83 -	0.25	0.056	0.091	18
41.26,69.21 -	0.12	0.2	0.15	10
41.33,19.8 -	0.2	0.2	0.2	5
41.71,44.78 -	0.67	0.8	0.73	5
41.9,12.48 -	0.18	0.2	0.19	10
42.86,74.6 -	0.57	0.8	0.67	5
44.41,26.1 -	0.33	0.2	0.25	5
52.5,-0.12 -	0.67	0.67	0.67	6
54.68,25.31 -	0.71	0.71	0.71	7
9.03,38.74 -	0.14	0.14	0.14	7
accuracy -	0.28	0.28	0.28	0.28
macro avg -	0.36	0.35	0.33	2.1e+02
weighted avg -	0.3	0.28	0.26	2.1e+02

precision recall f1-score support
 Accuracy of Neural Networks on test set: 0.44
 Cross Validation Scores for Neural Networks: [0.39622642 0.35849057 0.40566038 0.41509434 0.47169811 0.48113208
 0.36792453 0.38679245 0.38679245 0.42857143]
 Cross Validation Scores for Neural Networks (mean): 0.410

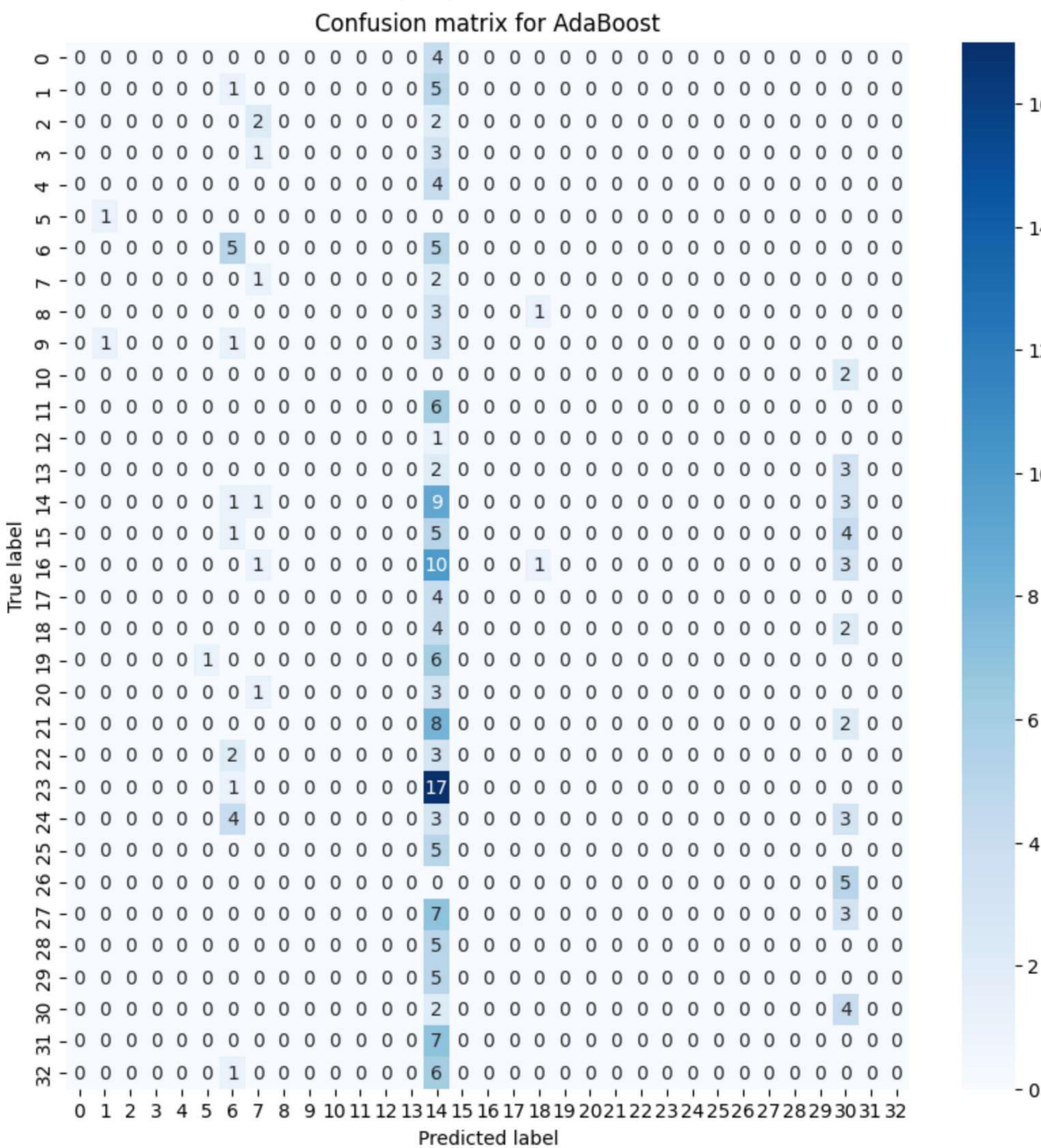


	precision	recall	f1-score	support	
41.26,69.21 -	0.25	0.1	0.14	10	- 50
41.33,19.8 -	0.33	0.2	0.25	5	
41.71,44.78 -	1	1	1	5	
41.9,12.48 -	0.27	0.3	0.29	10	- 25
42.86,74.6 -	0.67	0.8	0.73	5	
44.41,26.1 -	0.75	0.6	0.67	5	
52.5,-0.12 -	0.62	0.83	0.71	6	
54.68,25.31 -	1	1	1	7	
9.03,38.74 -	0.4	0.29	0.33	7	
accuracy -	0.44	0.44	0.44	0.44	
macro avg -	0.46	0.48	0.45	2.1e+02	
weighted avg -	0.44	0.44	0.43	2.1e+02	- 0

Accuracy of AdaBoost on test set: 0.09

Cross Validation Scores for AdaBoost: [0.10377358 0.10377358 0.1509434 0.14150943 0.10377358 0.10377358 0.10377358 0.11320755 0.12380952]

Cross Validation Scores for AdaBoost (mean): 0.115



Classification report for AdaBoost

	precision	recall	f1-score	support	
-1.26,36.8 -	0	0	0	4	- 200
-15.75,-47.95 -	0	0	0	6	
-35.3,149.12 -	0	0	0	4	
-6.17,106.82 -	0	0	0	4	
-6.17,35.74 -	0	0	0	4	
11.55,104.91 -	0	0	0	1	
12.65,-8.0 -	0.29	0.5	0.37	10	- 175
13.75,100.48 -	0.14	0.33	0.2	3	
14.66,-17.41 -	0	0	0	4	
14.91,-23.51 -	0	0	0	5	
17.25,-88.76 -	0	0	0	2	
17.98,-76.8 -	0	0	0	6	- 150

	precision	recall	f1-score	support	
19.75,96.1 -	0	0	0	1	- 125
23.76,121.0 -	0	0	0	5	
28.61,77.2 -	0.06	0.64	0.11	14	
30.03,31.21 -	0	0	0	10	
33.66,73.16 -	0	0	0	15	
34.03,-6.85 -	0	0	0	4	
35.68,51.41 -	0	0	0	6	
35.7,139.71 -	0	0	0	7	
36.7,3.21 -	0	0	0	4	
38.0,23.71 -	0	0	0	10	
39.91,116.38 -	0	0	0	5	
39.91,32.83 -	0	0	0	18	- 75
41.26,69.21 -	0	0	0	10	
41.33,19.8 -	0	0	0	5	
41.71,44.78 -	0	0	0	5	
41.9,12.48 -	0	0	0	10	- 50
42.86,74.6 -	0	0	0	5	
44.41,26.1 -	0	0	0	5	
52.5,-0.12 -	0.12	0.67	0.2	6	
54.68,25.31 -	0	0	0	7	- 25
9.03,38.74 -	0	0	0	7	
accuracy -	0.09	0.09	0.09	0.09	
macro avg -	0.019	0.065	0.027	2.1e+02	
weighted avg -	0.023	0.09	0.033	2.1e+02	

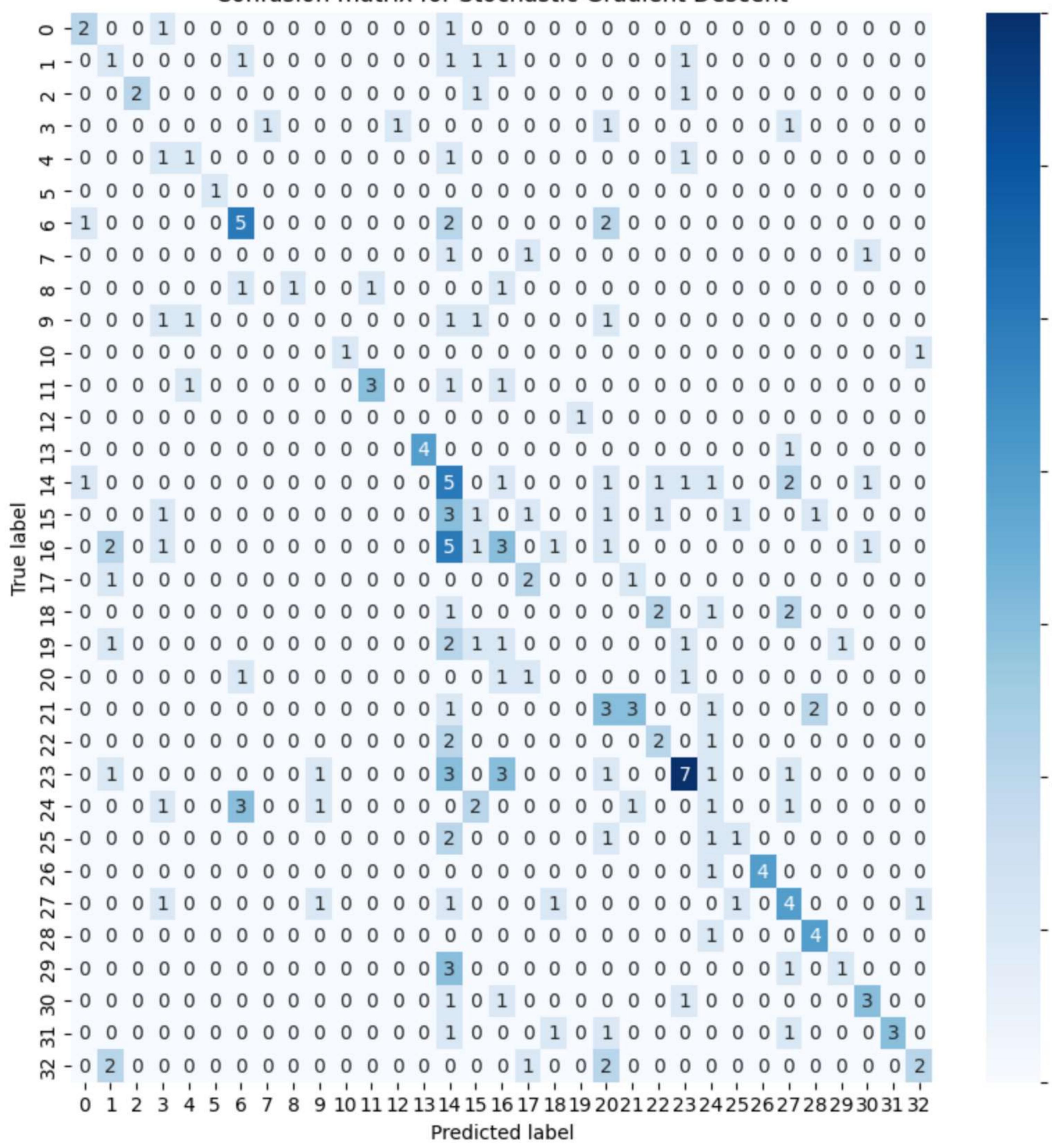
Accuracy of Stochastic Gradient Descent on test set: 0.32

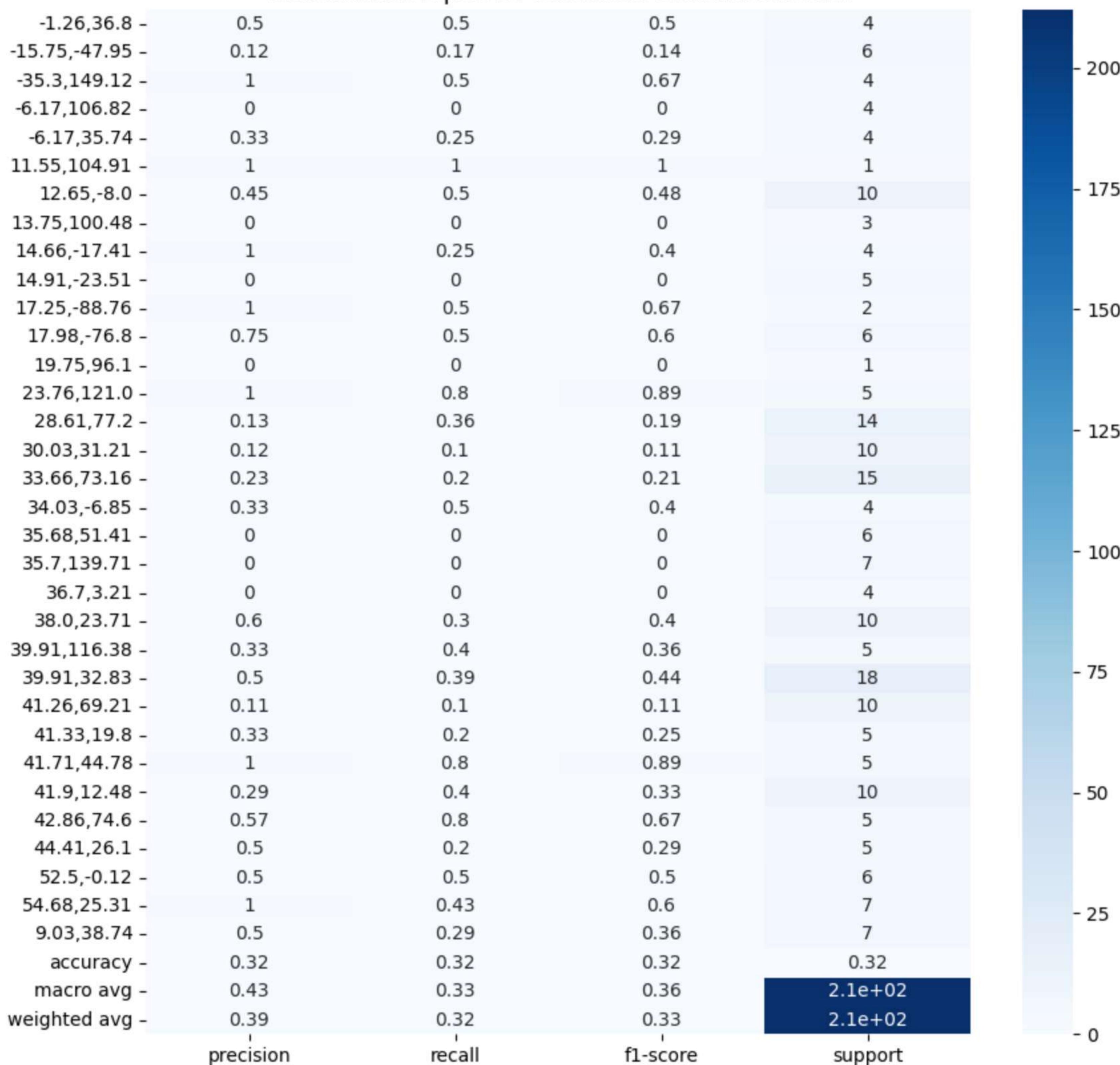
Cross Validation Scores for Stochastic Gradient Descent: [0.29245283 0.3490566 0.31132075 0.36792453 0.38679245 0.37735849

0.33018868 0.36792453 0.33962264 0.38095238]

Cross Validation Scores for Stochastic Gradient Descent (mean): 0.350

Confusion matrix for Stochastic Gradient Descent

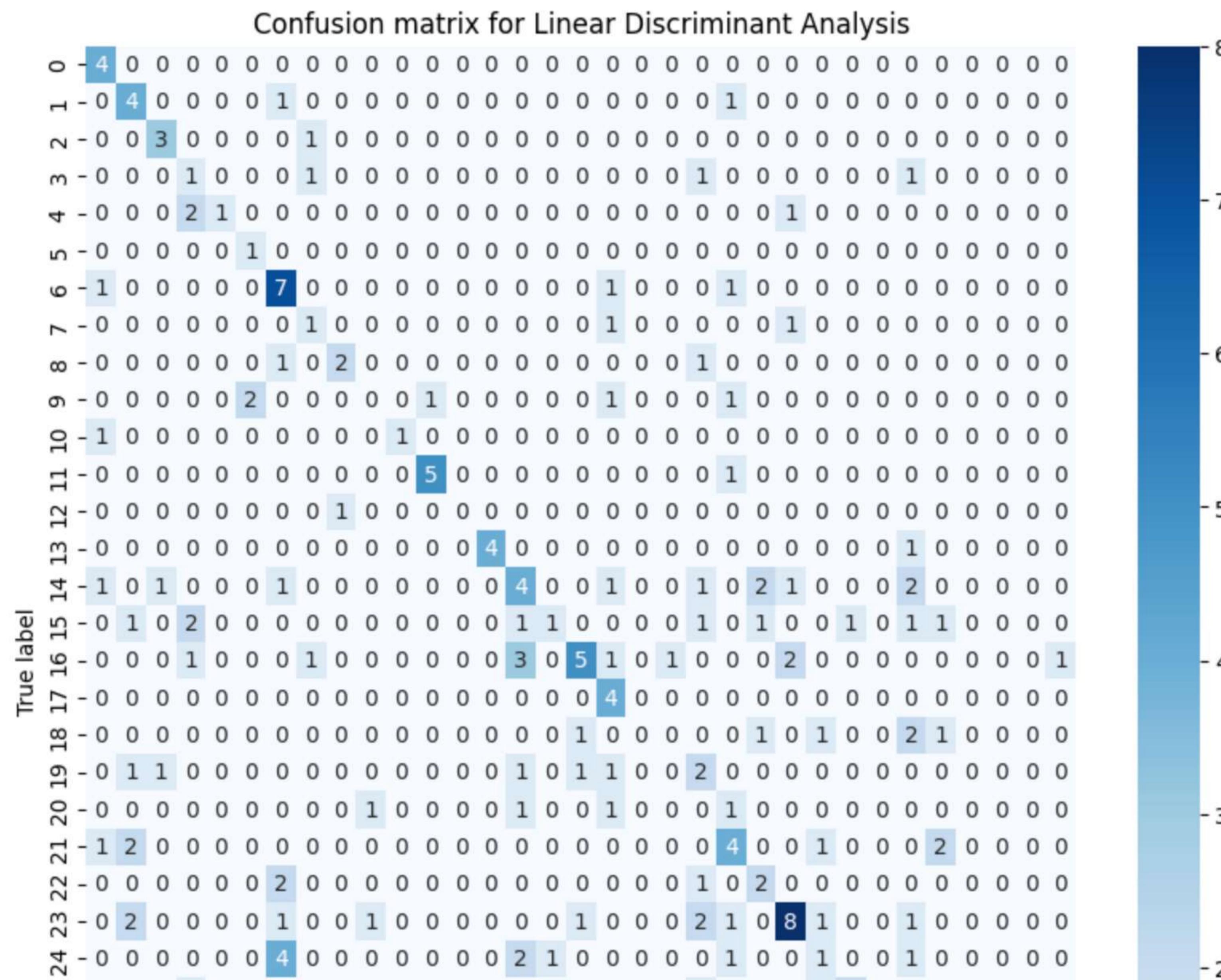


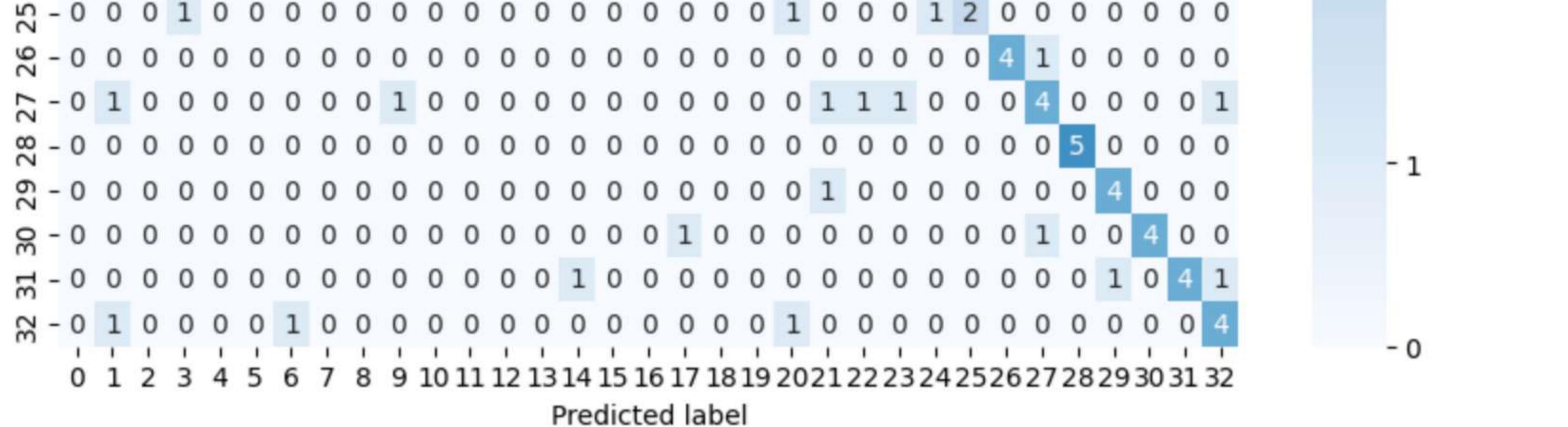


Accuracy of Linear Discriminant Analysis on test set: 0.44

Cross Validation Scores for Linear Discriminant Analysis: [0.38679245 0.4245283 0.43396226 0.39622642 0.47169811 0.47169811 0.39622642 0.41509434 0.41509434 0.4]

Cross Validation Scores for Linear Discriminant Analysis (mean): 0.421



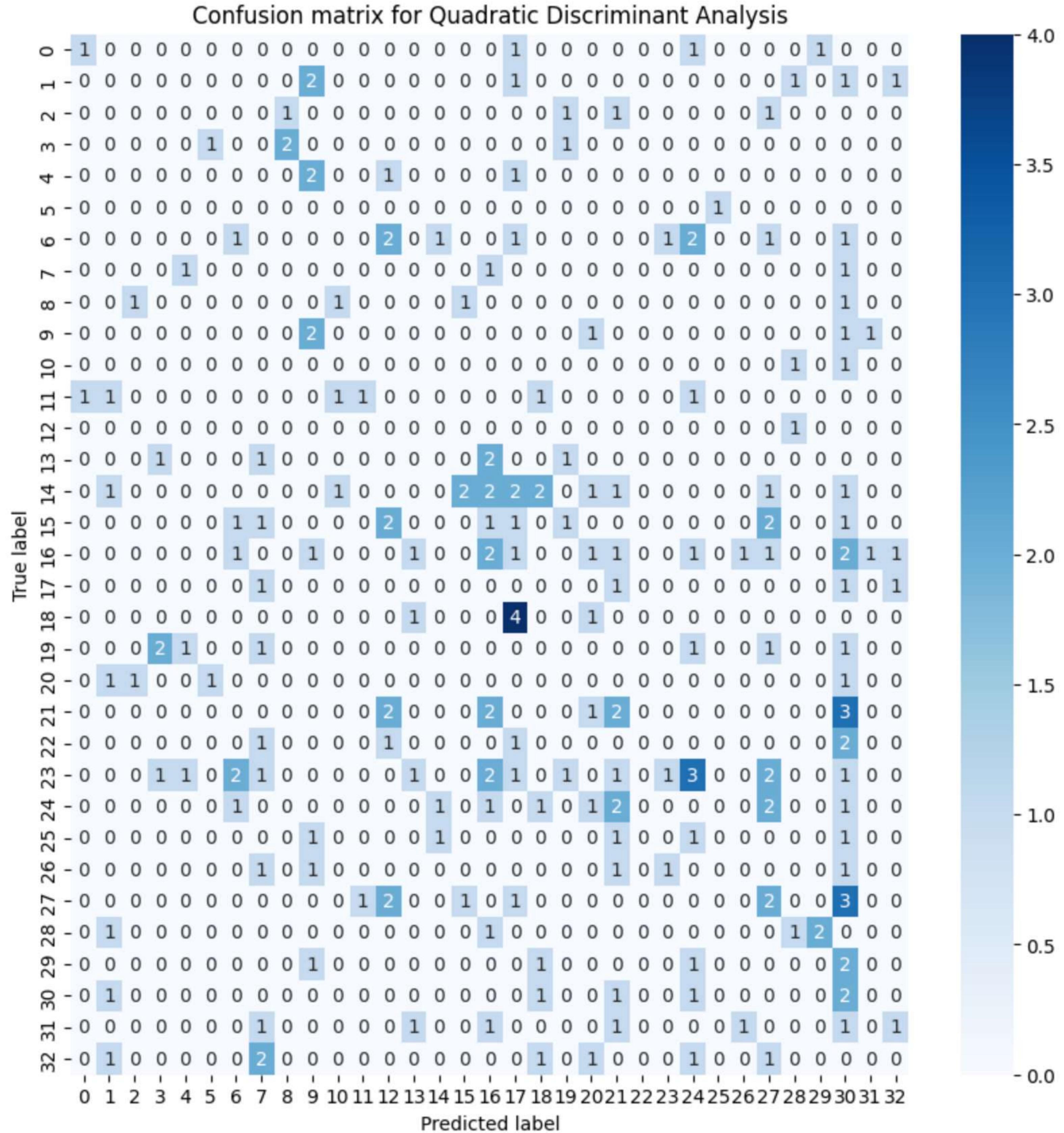


Classification report for Linear Discriminant Analysis

	precision	recall	f1-score	support
-1.26,36.8 -	0.5	1	0.67	4
-15.75,-47.95 -	0.33	0.67	0.44	6
-35.3,149.12 -	0.6	0.75	0.67	4
-6.17,106.82 -	0.14	0.25	0.18	4
-6.17,35.74 -	1	0.25	0.4	4
11.55,104.91 -	0.33	1	0.5	1
12.65,-8.0 -	0.39	0.7	0.5	10
13.75,100.48 -	0.25	0.33	0.29	3
14.66,-17.41 -	0.67	0.5	0.57	4
14.91,-23.51 -	0	0	0	5
17.25,-88.76 -	1	0.5	0.67	2
17.98,-76.8 -	0.83	0.83	0.83	6
19.75,96.1 -	0	0	0	1
23.76,121.0 -	1	0.8	0.89	5
28.61,77.2 -	0.31	0.29	0.3	14
30.03,31.21 -	0.5	0.1	0.17	10
33.66,73.16 -	0.62	0.33	0.43	15
34.03,-6.85 -	0.33	1	0.5	4
35.68,51.41 -	0	0	0	6
35.7,139.71 -	0	0	0	7
36.7,3.21 -	0	0	0	4
38.0,23.71 -	0.31	0.4	0.35	10
39.91,116.38 -	0.29	0.4	0.33	5
39.91,32.83 -	0.57	0.44	0.5	18
41.26,69.21 -	0.2	0.1	0.13	10
41.33,19.8 -	0.67	0.4	0.5	5
41.71,44.78 -	1	0.8	0.89	5
41.9,12.48 -	0.27	0.4	0.32	10
42.86,74.6 -	0.56	1	0.71	5
44.41,26.1 -	0.8	0.8	0.8	5
52.5,-0.12 -	1	0.67	0.8	6
54.68,25.31 -	1	0.57	0.73	7
9.03,38.74 -	0.57	0.57	0.57	7
accuracy -	0.44	0.44	0.44	0.44
macro avg -	0.49	0.48	0.44	2.1e+02
weighted avg -	0.48	0.44	0.43	2.1e+02

```
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/discriminant_analysis.py:887: UserWarning: Variables are collinear
  warnings.warn("Variables are collinear")
Accuracy of Quadratic Discriminant Analysis on test set: 0.07
Cross Validation Scores for Quadratic Discriminant Analysis: [0.00943396 0.03773585 0.04716981 0.06603774 0.05660377
 0.04716981 0.02830189 0.04716981 0.02857143]
Cross Validation Scores for Quadratic Discriminant Analysis (mean): 0.037
```

Cross-validation scores for Quadratic Discriminant Analysis (mean): 0.057



Classification report for Quadratic Discriminant Analysis

-1.26,36.8 -	0.5	0.25	0.33	4
-15.75,-47.95 -	0	0	0	6
-35.3,149.12 -	0	0	0	4
-6.17,106.82 -	0	0	0	4
-6.17,35.74 -	0	0	0	4
11.55,104.91 -	0	0	0	1
12.65,-8.0 -	0.17	0.1	0.12	10
13.75,100.48 -	0	0	0	3
14.66,-17.41 -	0	0	0	4
14.91,-23.51 -	0.2	0.4	0.27	5
17.25,-88.76 -	0	0	0	2
17.98,-76.8 -	0.5	0.17	0.25	6
19.75,96.1 -	0	0	0	1
23.76,121.0 -	0	0	0	5
28.61,77.2 -	0	0	0	14
30.03,31.21 -	0	0	0	10
33.66,73.16 -	0.13	0.13	0.13	15
34.03,-6.85 -	0	0	0	4
35.68,51.41 -	0	0	0	6
35.7,139.71 -	0	0	0	7
36.7,3.21 -	0	0	0	4
38.0,23.71 -	0.15	0.2	0.17	10
39.91,116.38 -	0	0	0	5
39.91,32.83 -	0.33	0.056	0.095	18
41.26,69.21 -	0	0	0	10
41.33,19.8 -	0	0	0	5
41.71,44.78 -	0	0	0	5
41.9,12.48 -	0.14	0.2	0.17	10

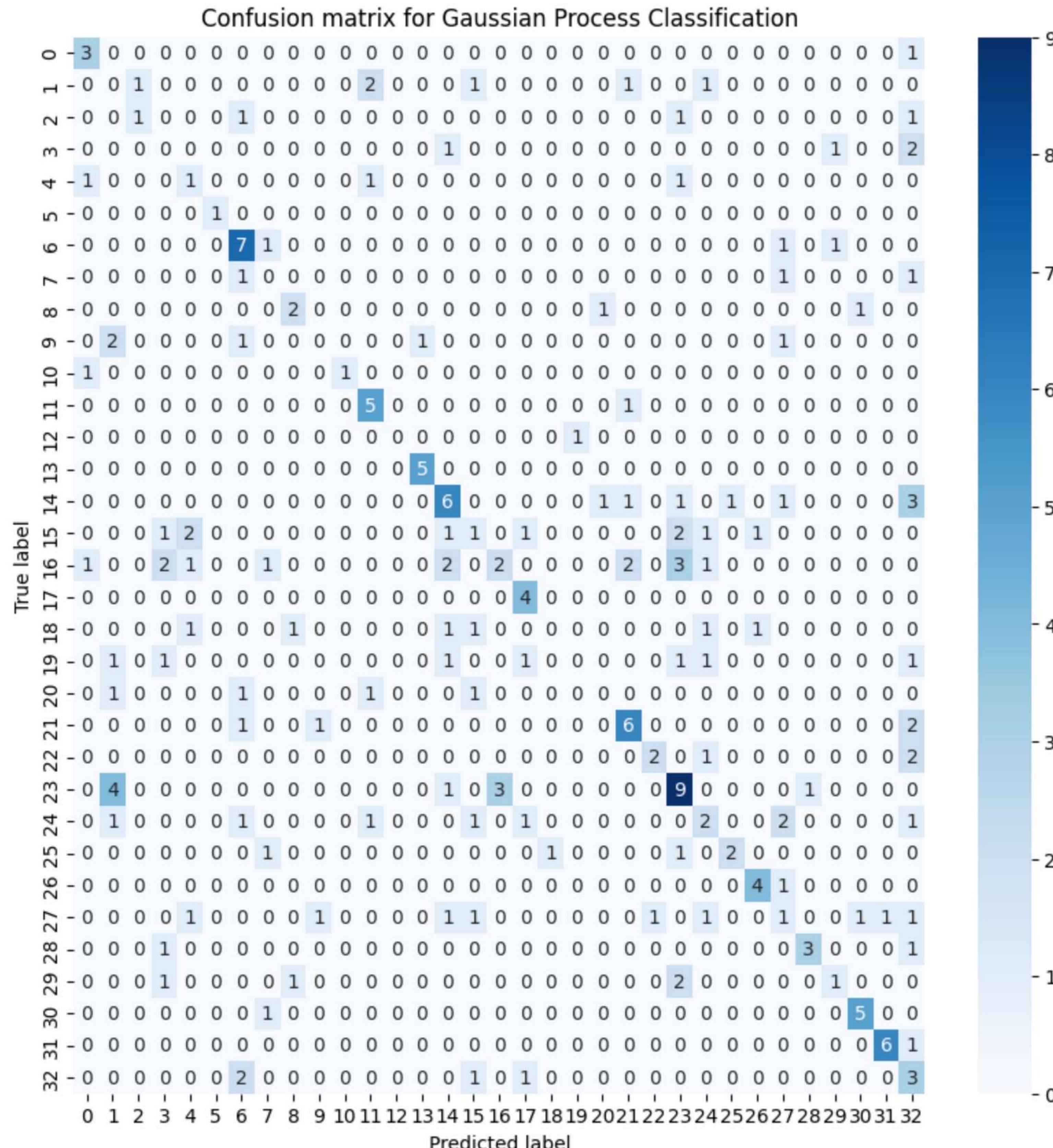
	precision	recall	f1-score	support
42.86,74.6 -	0.25	0.2	0.22	5
44.41,26.1 -	0	0	0	5
52.5,-0.12 -	0.067	0.33	0.11	6
54.68,25.31 -	0	0	0	7
9.03,38.74 -	0	0	0	7
accuracy -	0.071	0.071	0.071	0.071
macro avg -	0.074	0.062	0.057	2.1e+02
weighted avg -	0.096	0.071	0.068	2.1e+02

Accuracy of Gaussian Process Classification on test set: 0.39

Cross Validation Scores for Gaussian Process Classification: [0.30188679 0.35849057 0.39622642 0.44339623 0.46226415 0.44339623]

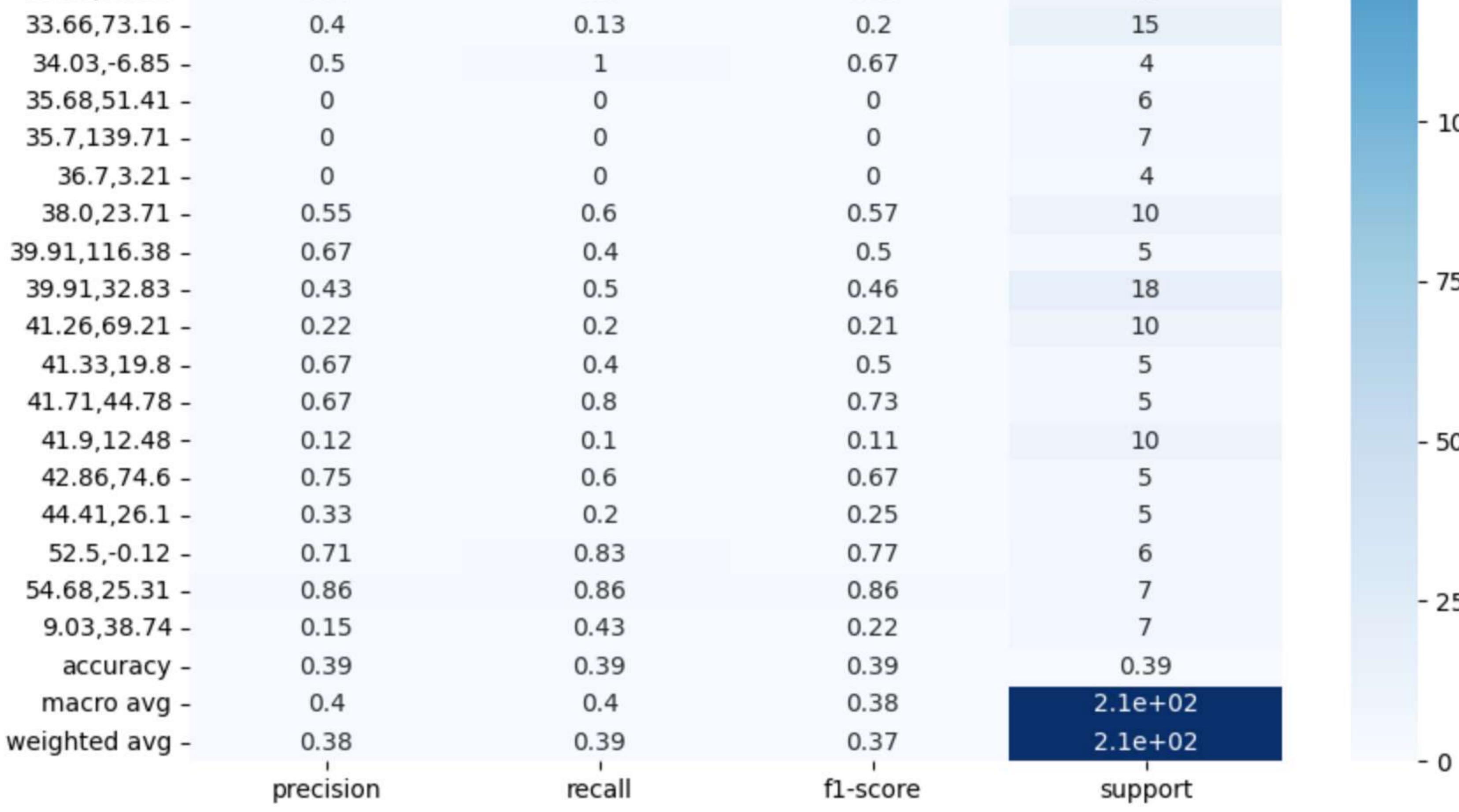
```
0.31132075 0.40566038 0.37735849 0.39047619]
```

Cross Validation Scores for Gaussian Process Classification (mean): 0.389



Classification report for Gaussian Process Classification

Classification Report for Gaussian Process Classification					
	Prediction	True Class	TP	FP	FN
-1.26,36.8 -	0.5	0.75	0.6	4	
-15.75,-47.95 -	0	0	0	6	
-35.3,149.12 -	0.5	0.25	0.33	4	
-6.17,106.82 -	0	0	0	4	
-6.17,35.74 -	0.17	0.25	0.2	4	
11.55,104.91 -	1	1	1	1	
12.65,-8.0 -	0.47	0.7	0.56	10	
13.75,100.48 -	0	0	0	3	
14.66,-17.41 -	0.5	0.5	0.5	4	
14.91,-23.51 -	0	0	0	5	
17.25,-88.76 -	1	0.5	0.67	2	
17.98,-76.8 -	0.5	0.83	0.62	6	
19.75,96.1 -	0	0	0	1	
23.76,121.0 -	0.83	1	0.91	5	
28.61,77.2 -	0.43	0.43	0.43	14	
30.03,31.21 -	0.14	0.1	0.12	10	



In [11]:

```
def evaluate(models, X_train, X_test, y_train, y_test):
    # make a dictionary to keep model scores
    model_scores = {}
    # loop through models
    for name, model_dict in models.items():
        # predict the test set results
        y_pred = model_dict['model'].predict(X_test)
        # evaluate the model and append its score to model_scores
        model_scores[name] = {'Accuracy': accuracy_score(y_test, y_pred),
                             'Precision': precision_score(y_test, y_pred, average = 'weighted'),
                             'Recall': recall_score(y_test, y_pred, average = 'weighted'),
                             'F1': f1_score(y_test, y_pred, average = 'weighted')}
    return model_scores

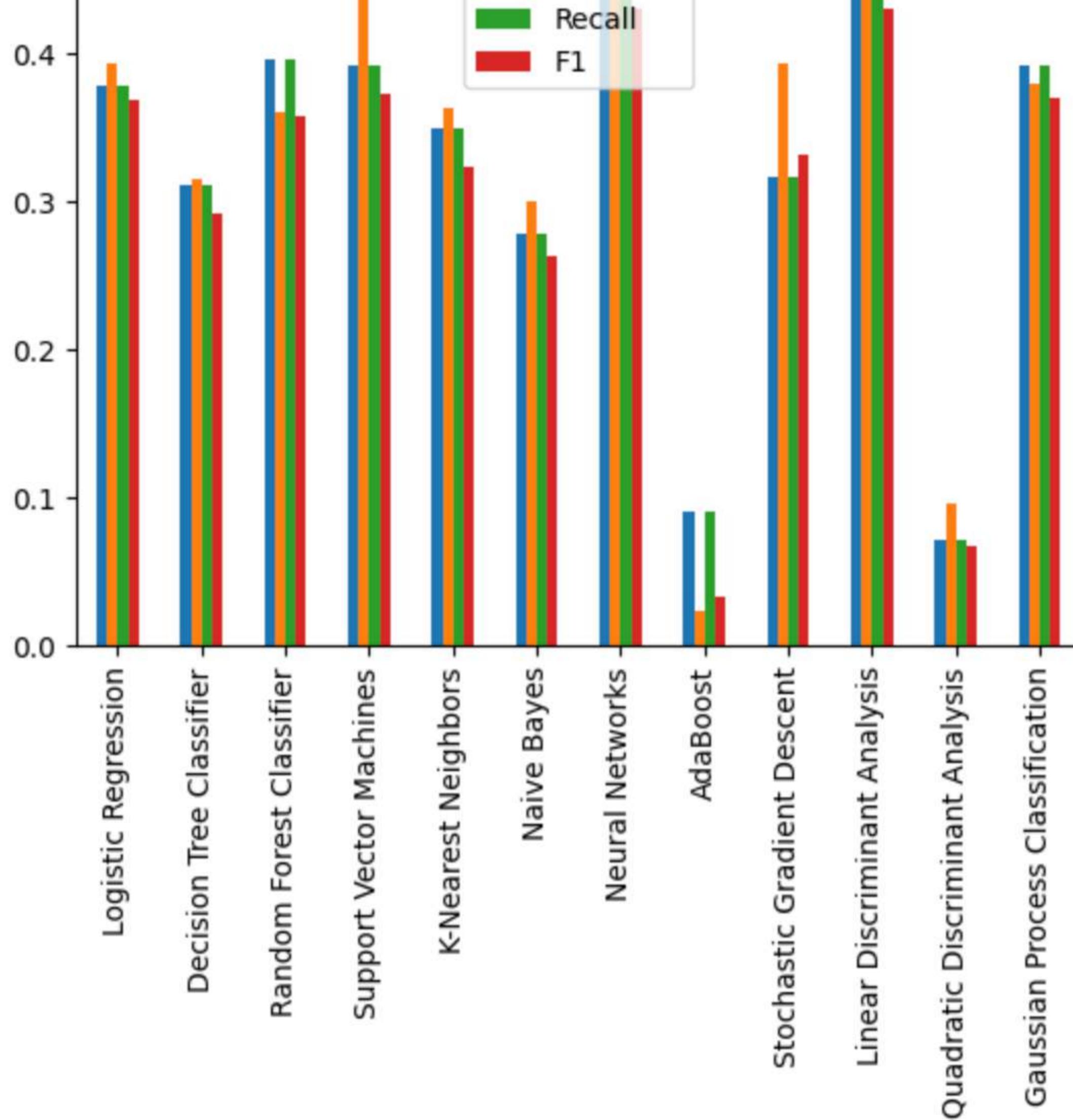
model_evaluation_scores = evaluate(models = models,
                                    X_train = X_train,
                                    X_test = X_test,
                                    y_train = y_train,
                                    y_test = y_test)
```

```
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
/Users/ma/Library/Python/3.9/lib/python/site-packages/sklearn/metrics/_classification.py:1334: UndefinedMetricWarning: Precision is ill-defined and being set to 0.0 in labels with no predicted samples. Use `zero_division` parameter to control this behavior.
    _warn_prf(average, modifier, msg_start, len(result))
```

In [12]:

```
# visualize the model scores
model_compare = pd.DataFrame(model_evaluation_scores)
model_compare.T.plot.bar()
plt.xticks(rotation = 90)
plt.show()
```





Linear Discriminant Analysis

Linear Discriminant Analysis (LDA), makine öğrenimi ve istatistikte sıkılıkla kullanılan bir öğrenme algoritmasıdır. Bu algoritma, sınıflar arasında en iyi şekilde ayırtırı̄an doğrusal bir özellik kombinasyonunu bulma fikrine dayanır.

LDA şu şekilde çalışır:

1. Algoritma ilk olarak, her sınıfın verilerinin Gauss dağılımından çekildiğini varsayar.
2. Daha sonra, her sınıf için Gauss dağılımının parametrelerini (ortalama ve varyans) tahmin eder.
3. Sonra, sınıflar arasında en iyi şekilde ayırtırı̄an doğrusal özellik kombinasyonunu bulur. Bu, sınıflar arası yayılım ile sınıflar içi yayılımı arasındaki oranı en yüksek olan kombinasyonu bulmak suretiyle yapılır.
4. Son olarak, verileri doğrusal özellik kombinasyonuna yansıtır ve yeni veri noktalarını sınıflandırmak için basit bir eşik kullanır.

LDA, sınıflandırma görevleri için kullanışlı bir algoritmadır çünkü kolayca uygulanır, hızlı bir şekilde eğitilebilir ve çeşitli veri kümelerinde iyi bir performans gösterir. Ayrıca, bazı diğer sınıflandırma algoritmalarına göre daha az overfittinge duyarlıdır. Ancak, bazı kısıtlamaları da vardır. Örneğin, verilerin Gauss dağılımı izlediğini varsayar, bu her zaman olmamayırlar. Ayrıca, sınıfların eşit varyanslara sahip olduğunu varsayar, bu da her zaman doğru olmamayırlar.

Genel olarak, LDA sınıflandırma görevleri için basit ve etkili bir algoritmadır.

Support Vector Machines

Support Vector Machines (SVM) bir sınıflandırma algoritmasıdır ve çoklu sınıflandırma görevleri için de kullanılabilir. Bu algoritma, veri noktalarını iki sınıfa bölmek için en iyi şekilde çalışan doğrusal bir ayırcı çizgiyi bulmayı hedefler. SVM algoritması, veri noktalarının sınıflandırılması için doğrusal olmayan çizgiler de kullanabilir, ancak bunlar için daha fazla özellik dönüşümü gereklidir.

SVM algoritması çalışma şéklini şu şekilde özetleyebiliriz:

1. Öncelikle, veri noktalarının sınıflandırılması için doğrusal bir ayırcı çizgi bulunur. Bu çizgi, veri noktalarının en büyük mümkün mesafeyi sağladığı noktalardan geçer. Bu noktalara "destek vektörleri" denir.
2. Daha sonra, veri noktaları destek vektörlerine göre sınıflandırılır.
3. Sınıflandırma işlemi tamamlandıktan sonra, yeni gelen veri noktalarının sınıflandırılması için aynı doğrusal çizgi kullanılır.

SVM algoritması, veri noktalarını doğrusal olmayan çizgilerle de sınıflandırabilir. Bunun için, veri noktalarını daha yüksek boyutlu bir uzayda dönüştürür ve orada doğrusal bir ayırcı çizgi bulunur. Bu işlem "özellik dönüşümü" olarak adlandırılır.

SVM algoritması, veri noktalarını en iyi şekilde ayırtırı̄an doğrusal çizgiyi bulma kabiliyetine sahip olması nedeniyle çok popülerdir ve çeşitli veri kümelerinde iyi bir performans gösterir. Ancak, algoritmanın performansı büyük veri kümelerinde yavaş olabilir ve hiperparametrelerin doğru ayarlanması gereklidir.

Neural Networks

Neural Networks (NN), sınıflandırma ve regresyon görevleri için kullanılan bir makine öğrenimi algoritmasıdır. NN, birçok gizli katmanı olan bir sinir ağı yapısına sahiptir ve bu katmanlar arasında veri taşınır. Gizli katmanlar arasındaki veri taşınışı, ağırlık değerleri ile kontrol edilir. Ağırlık değerleri, öğrenme sırasında optimize edilir ve ağıın performansını etkiler.

resmin neye ait olduğunu sınıflandırır. NN'ler, öğrenme sırasında ağırlık değerlerini optimize ederek daha iyi performans gösterirler.

NN'ler, çok sayıda girdi verisi işleyebildiğinden, çok sayıda özellik içeren veri kümelerinde iyi bir performans gösterirler. Ayrıca, NN'ler, veri örüntülerini öğrenme kabiliyetine sahiptirler ve bu öğrendiklerini yeni veri noktaları için kullanabilirler. Ancak, NN'lerin performansı hiperparametrelerin doğru ayarlanmasıyla bağlıdır ve bazen overfitting problemine neden olabilirler.

Kümeleme:

Kümeleme, benzer veri noktalarını kümelere grupperleyen bir süperfisyonal olmaksızın makine öğrenimi teknigidir. Kümeleme amacı, verideki desenleri tespit etmek ve veri noktalarını benzerliklerine göre grupperleyemektir. Kümeleme algoritmaları, görüntü bölütleme, anomalî tespiti ve müsteri bölütleme gibi çeşitli uygulamalarda kullanılır.

Popüler kümeleme algoritmaları arasında şunlar bulunur:

K-Means Kümeleme: Bu, en popüler ve basit kümeleme algoritmalarından biridir. Rastgele bir sayıda (k) merkez noktasını seçerek başlar ve sonra her veri noktasını en yakın merkez noktasına atar. Merkez noktaları, onlara atan veri noktalarının ortalamasına göre güncellenir ve işlem dönmeye devam eder.

Hiyerarşik Kümeleme: Bu algoritma, veri noktalarının tek tek başladığı ve ilerleyen algoritma sırasında daha büyük ve daha büyük kümelere bireleştirildiği bir kümeler hiyerarşisi oluşturur. Hiyerarşik kümeleme iki ana tiptedir: agglomerative (aşağıdan yukarı) ve divisive (yukarıdan aşağı).

DBSCAN (Gürültülü Uygulamalar İçin Yoğunluğa Dayalı Uzaysal Kümeleme): Bu algoritma, merkez noktalarından değil, yoğunluktan veri noktalarını kümelere ayırır. DBSCAN, belirli bir mesafedeki (Eps parametresi olarak adlandırılan) komşu noktalarının yüksek sayısına sahip "çekirdek noktaları" belirleyerek başlar ve sonra kümeyi, çekirdek noktalarından erişilebilen tüm noktaları da içerecek şekilde genişletir.

Gaussian Karması Modeli (GMM): Bu, verinin birkaç farklı Gaussian dağılımından oluşan bir karışımından üretildiği varsayımlarına dayalı bir olasılık modelidir. GMM, küresel olmayan veya açıkça ayrılmış olan veriyi modellemek için esnek bir algoritmadır.

Spektral Kümeleme: Bu algoritma, veri noktaları arasındaki benzerliği kodlayan bir benzerlik matrisinin eigenvectors'ını kullanarak veriyi düşük boyutlu bir uzayda projeler ve sonra projekte edilmiş veri üzerinde başka bir kümeleme algoritmasını (örneğin k-means) uygular. Spektral kümeleme, doğrusal olarak ayırt edilemeyecek veri için özellikle yararlıdır.

Bunların dışında Fuzzy C-Means (FCM), Self-Organizing Map (SOM), Affinity Propagation, Mean-Shift, Expectation-Maximization (EM), Agglomerative Information Bottleneck (AIB), Single Linkage Clustering, Ward's Method, Complete Linkage Clustering, Birch Clustering gibi algoritmalar da bulunmaktadır. Hangi algoritmanın en uygun olduğu, verinizin özelliklerine ve kümeleme amacınıza göre değişebilir. Bu yüzden, veri kümelerini inceleyerek hangi algoritmanın en uygun olduğunu belirlemeye çalışmak gereklidir.

In [13]:

```
kmeans = KMeans(n_clusters = 33, random_state = 42)
agg = AgglomerativeClustering(n_clusters = 33, affinity = 'euclidean', linkage = 'ward')
dbSCAN = DBSCAN(eps = 0.3, min_samples = 10)
spectral = SpectralClustering(n_clusters = 33, affinity = 'nearest_neighbors')
gmm = GaussianMixture(n_components = 33, covariance_type = 'full')
bgmm = BayesianGaussianMixture(n_components = 33, covariance_type = 'full')

clusters = {
    'K-Means': {'model': kmeans},
    'Agglomerative Clustering': {'model': agg},
    'DBSCAN': {'model': dbSCAN},
    'Spectral Clustering': {'model': spectral},
    'Gaussian Mixture': {'model': gmm},
    'Bayesian Gaussian Mixture': {'model': bgmm}
}
```

In [14]:

```
def visualize_cluster(X, y, title: str):
    """Visualize clusters."""
    plt.figure(figsize = (10, 10))
    plt.scatter(X.iloc[:, 0], X.iloc[:, 1], c = y, s = 50, cmap = 'viridis')
    plt.title(title)
    plt.show()

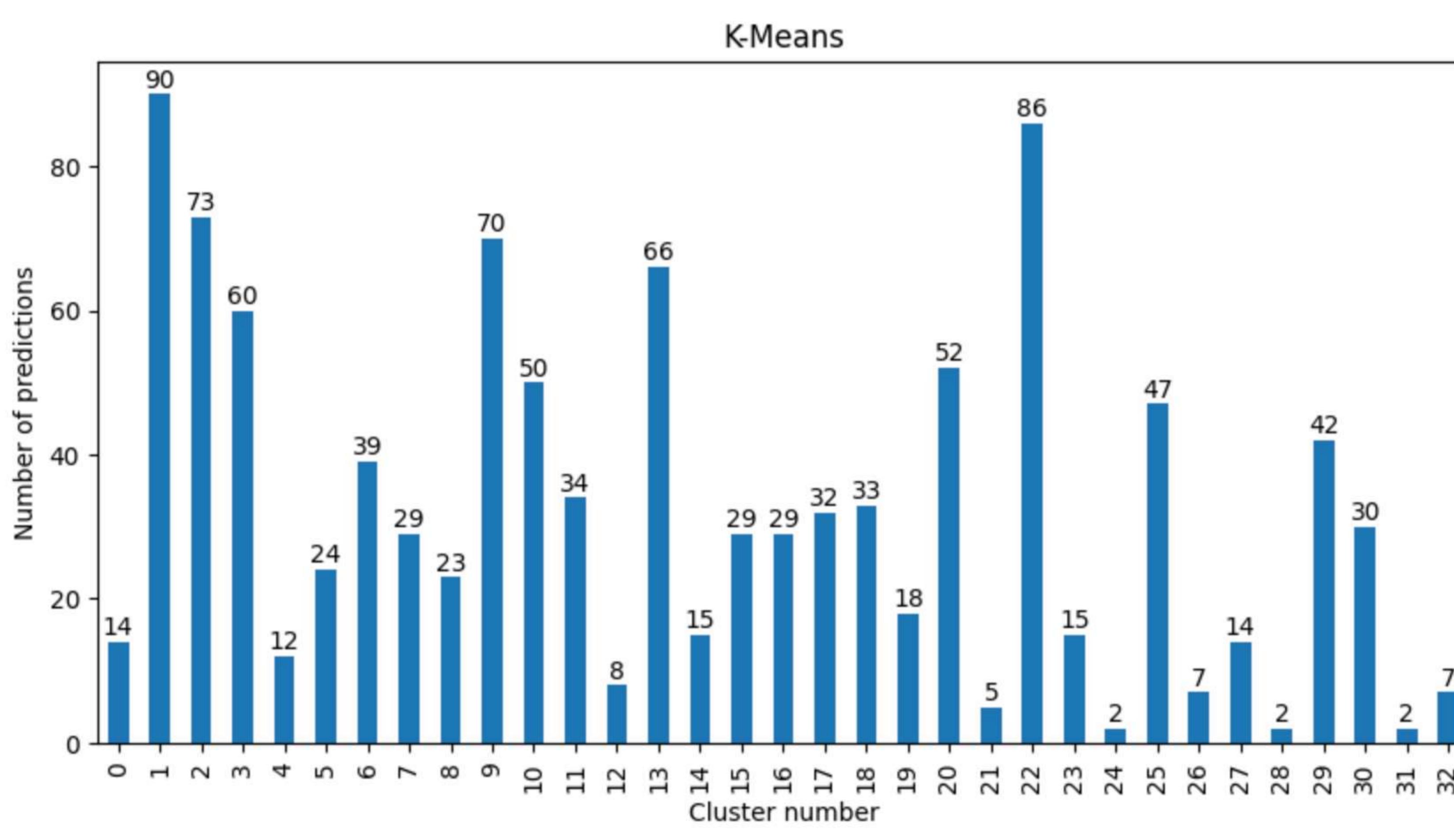
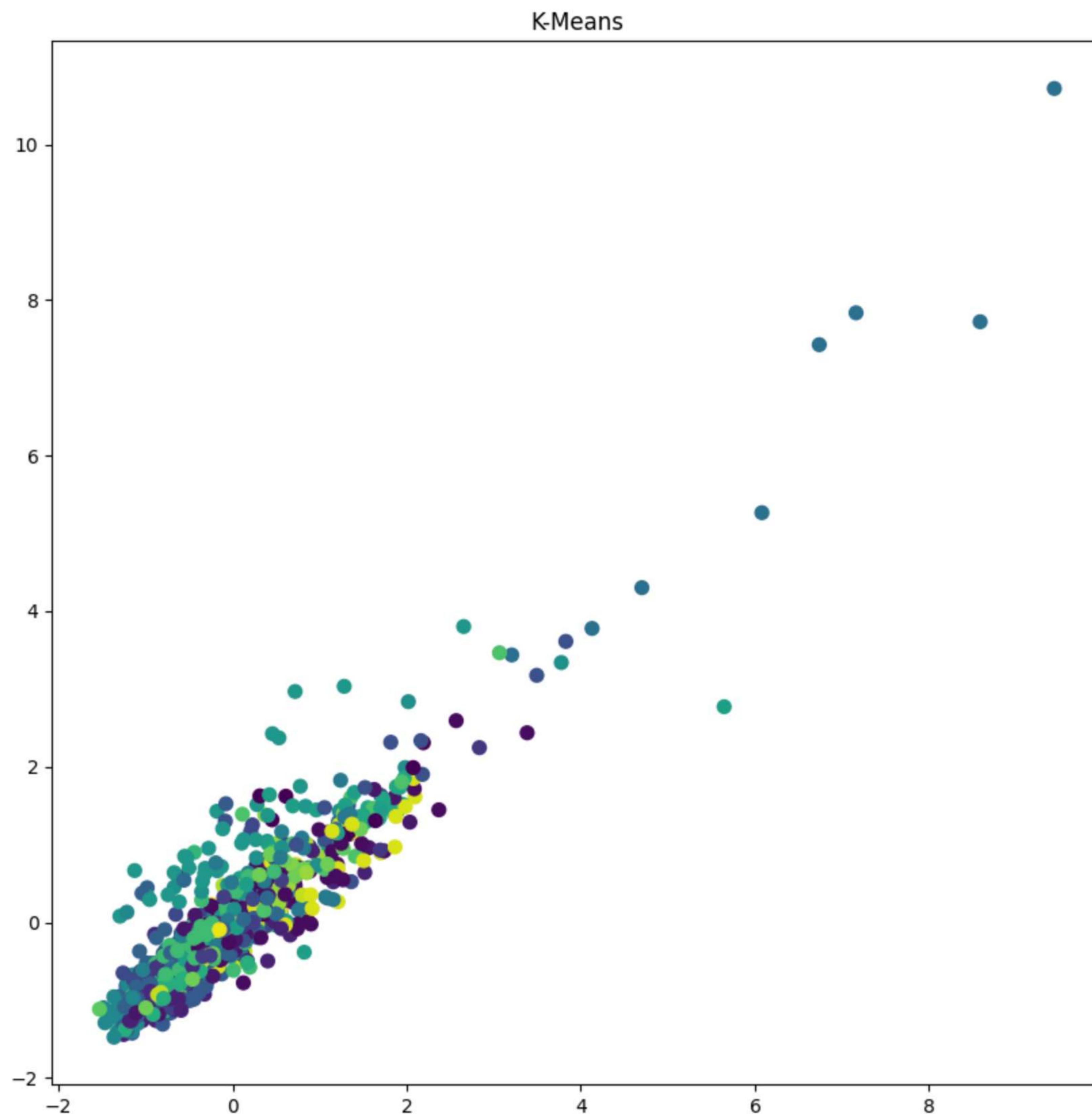
    prediction = pd.Series(y).value_counts().sort_index()
    prediction.plot(kind = 'bar') # bar plot
    # increase width of bars
    plt.gcf().set_size_inches(10, 5)
    plt.xlabel('Cluster number')
    plt.ylabel('Number of predictions')
    plt.title(title)
    for i, v in enumerate(prediction):
        plt.text(i, v + 1, str(v), horizontalalignment = 'center')
    plt.show()
```

In [15]:

```
for name, model_dict in clusters.items():
    model_dict['model'].fit(X)
    try:
        y_pred = model_dict['model'].predict(X)
    except:
        y_pred = model_dict['model'].fit_predict(X)
    model_dict['predictions'] = y_pred
```

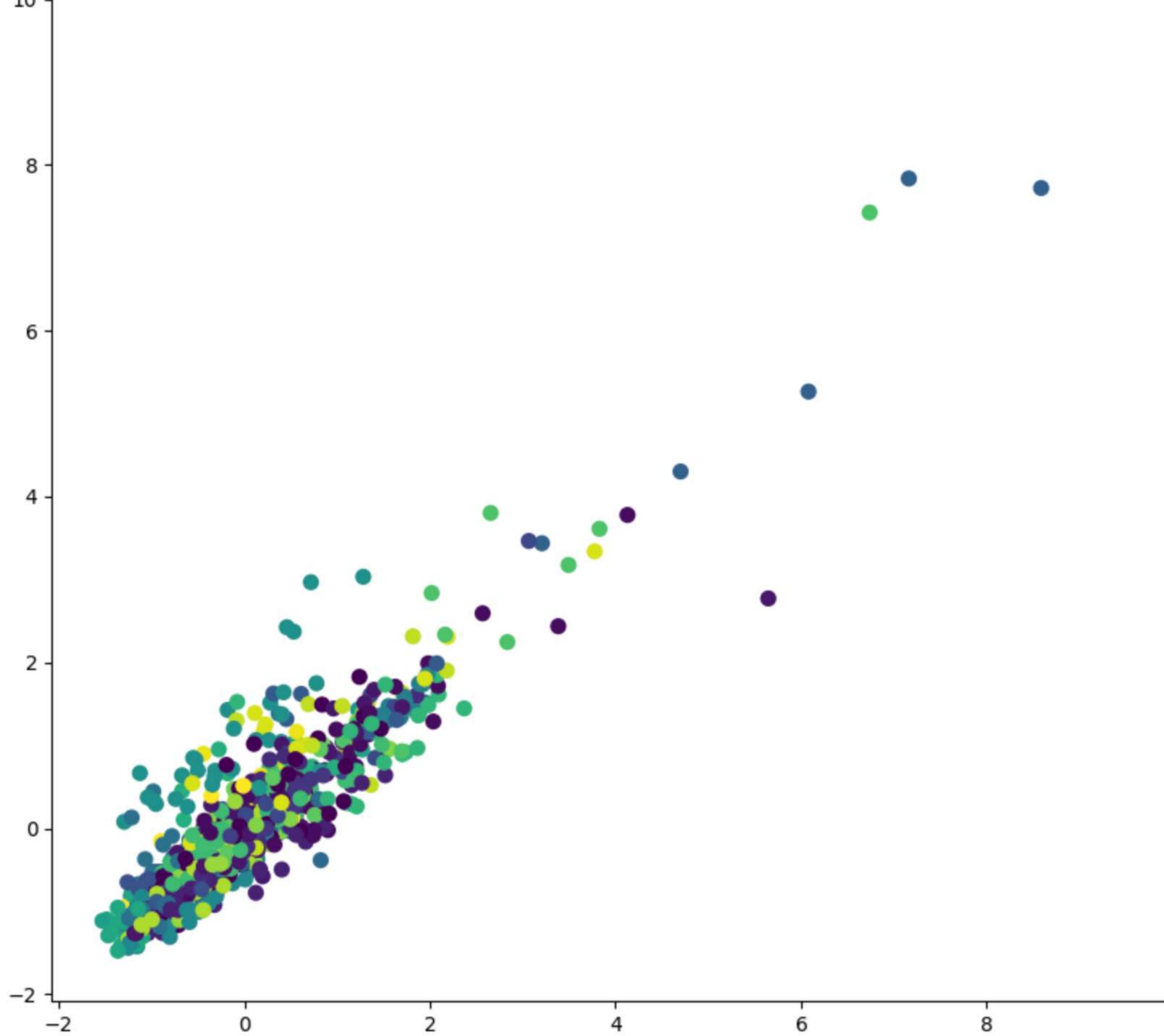
In [16]:

```
for name, model_dict in clusters.items():
    visualize_cluster(X, model_dict['predictions'], name)
```

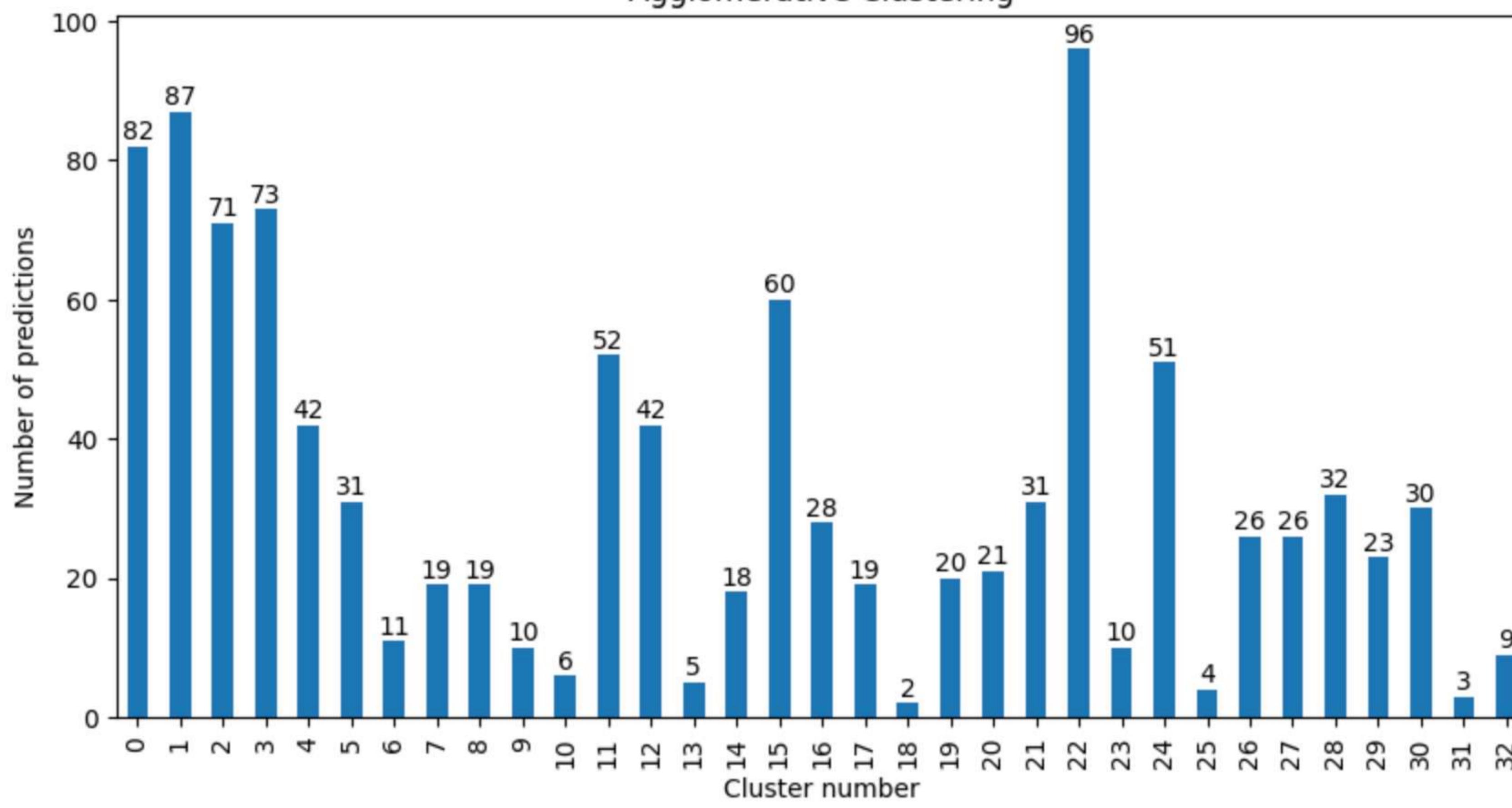


Agglomerative Clustering

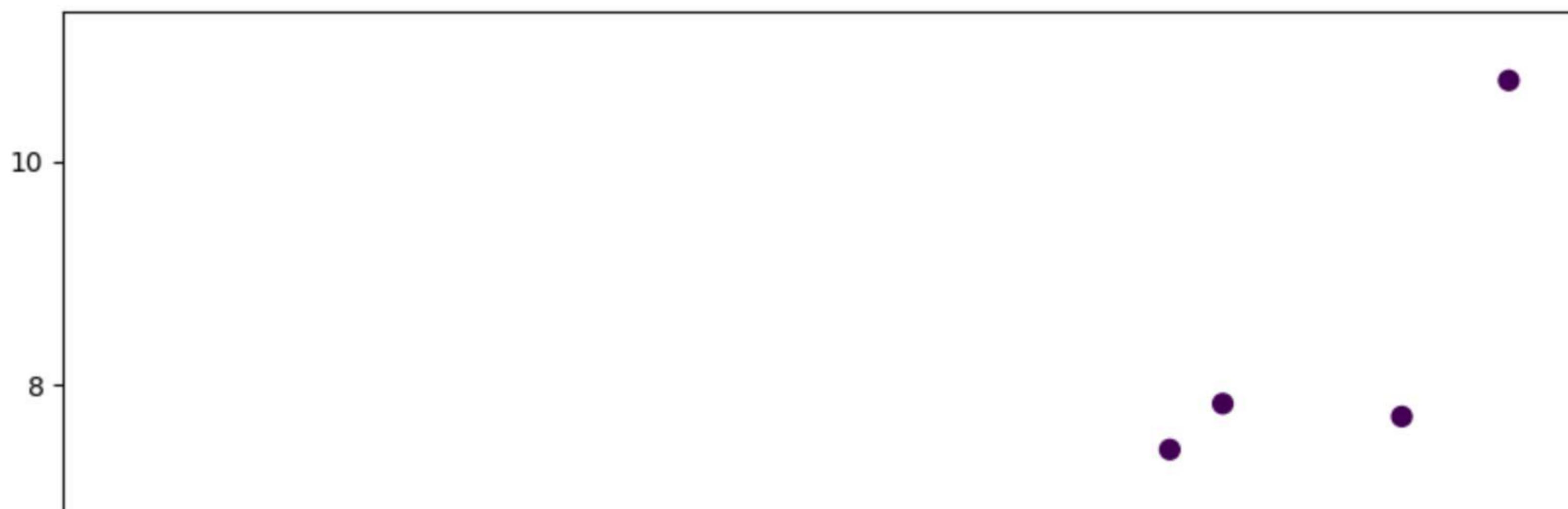


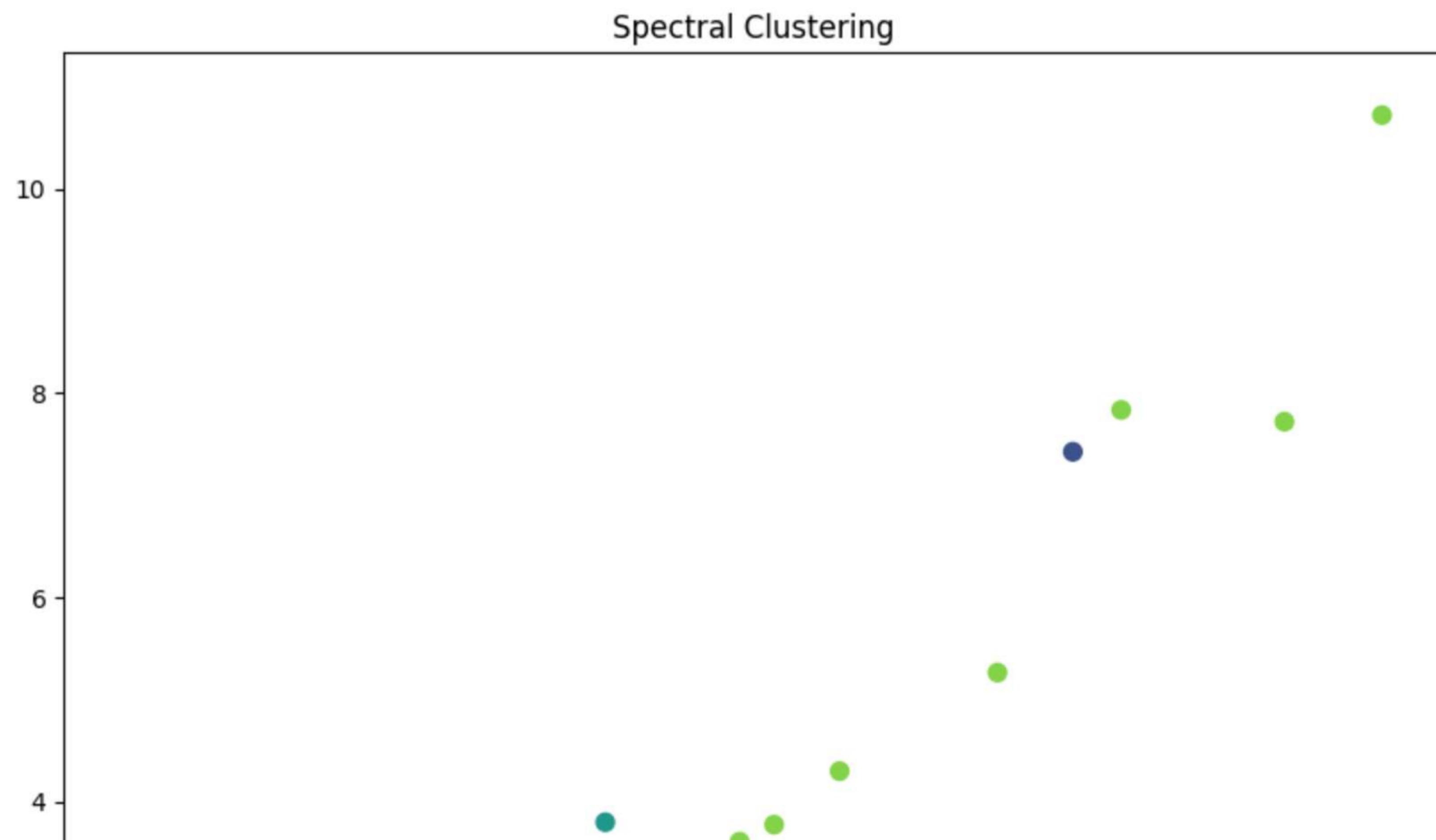
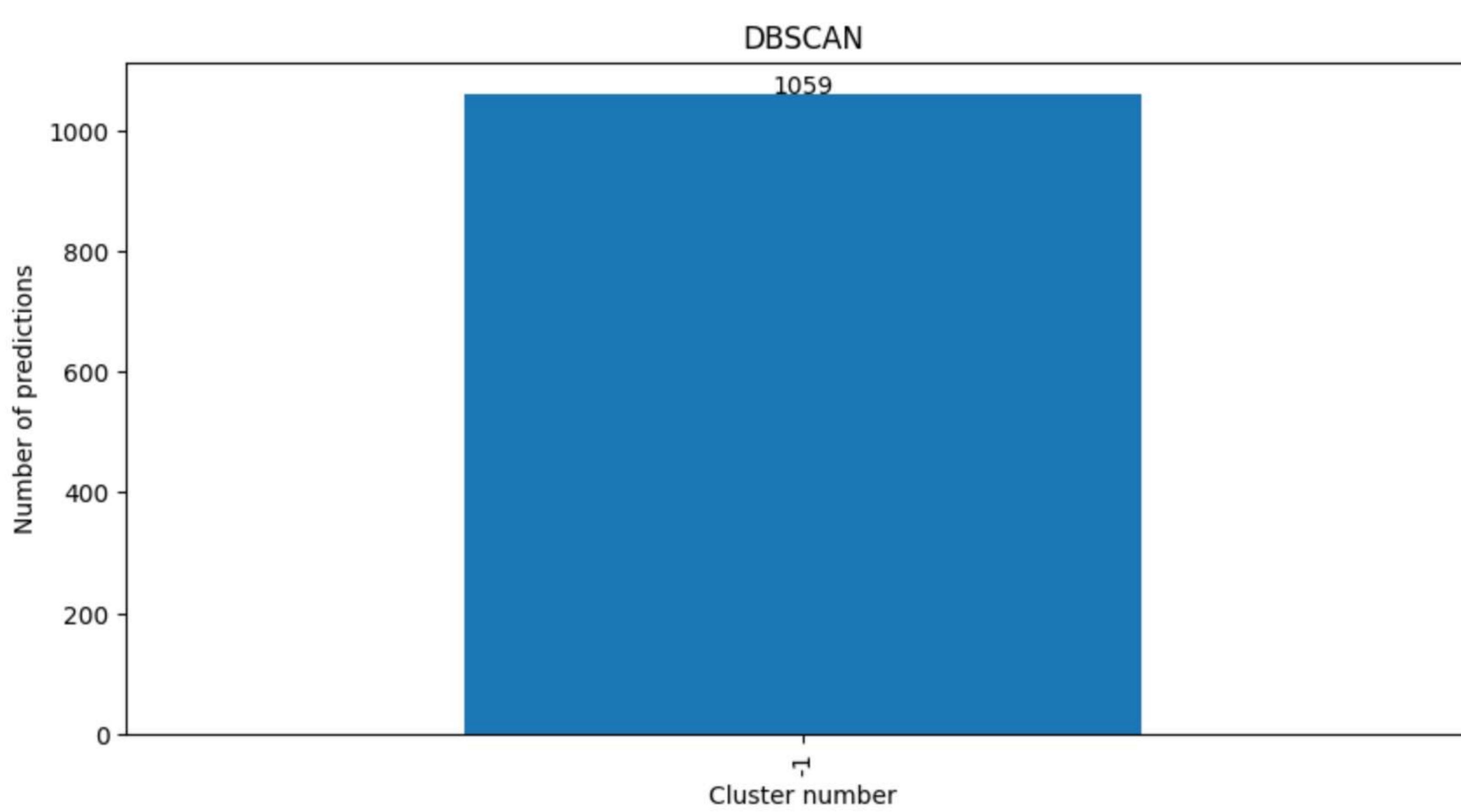
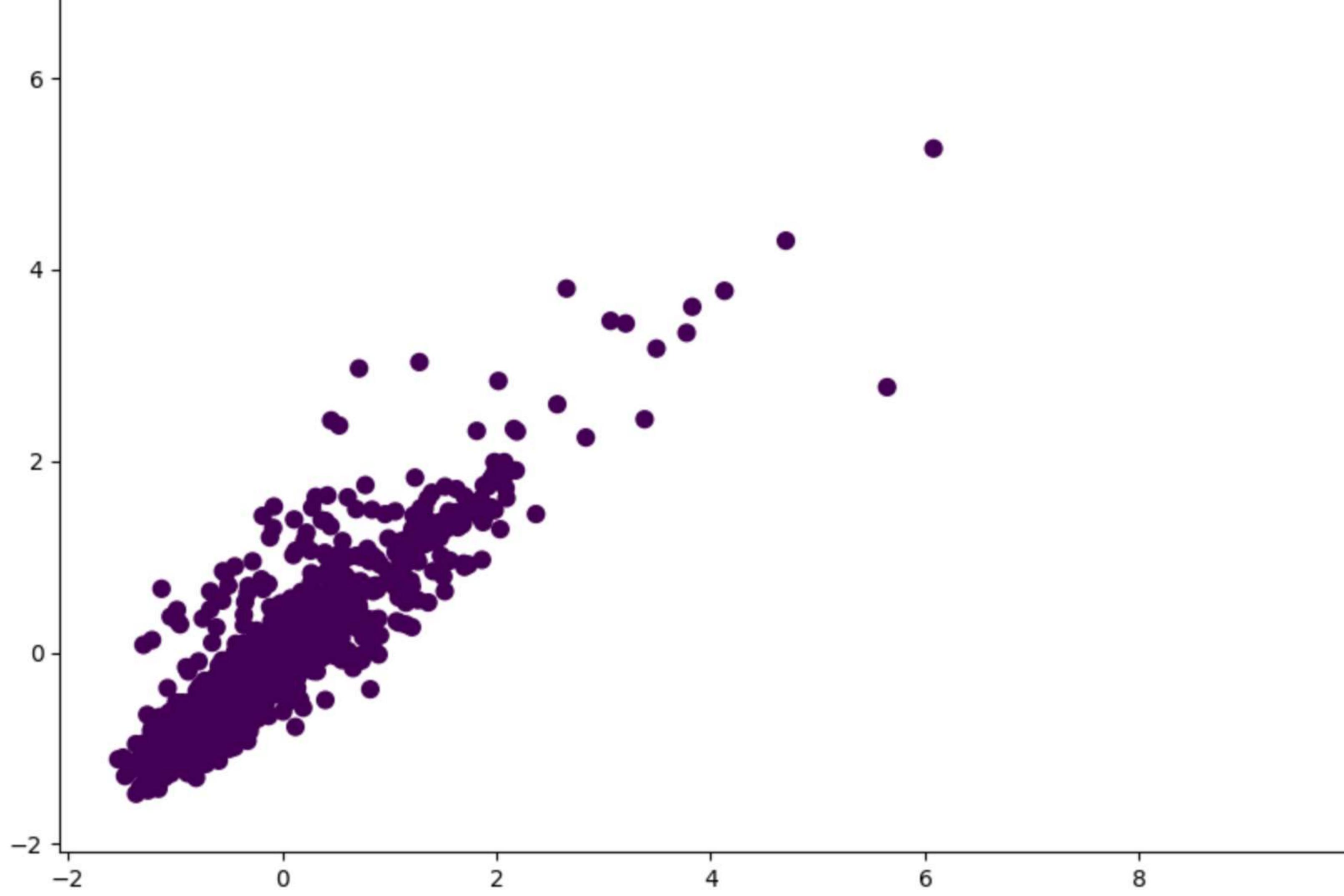


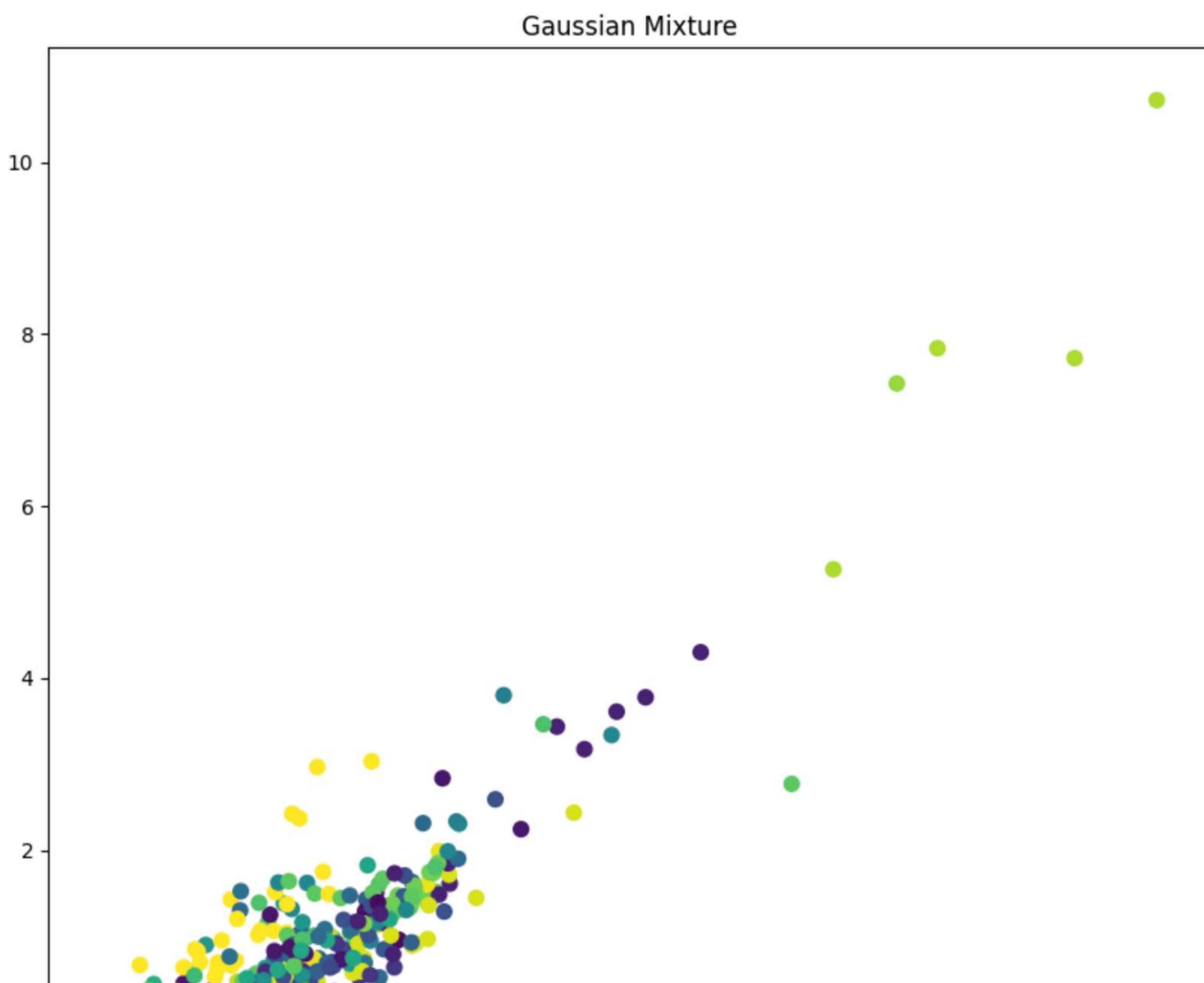
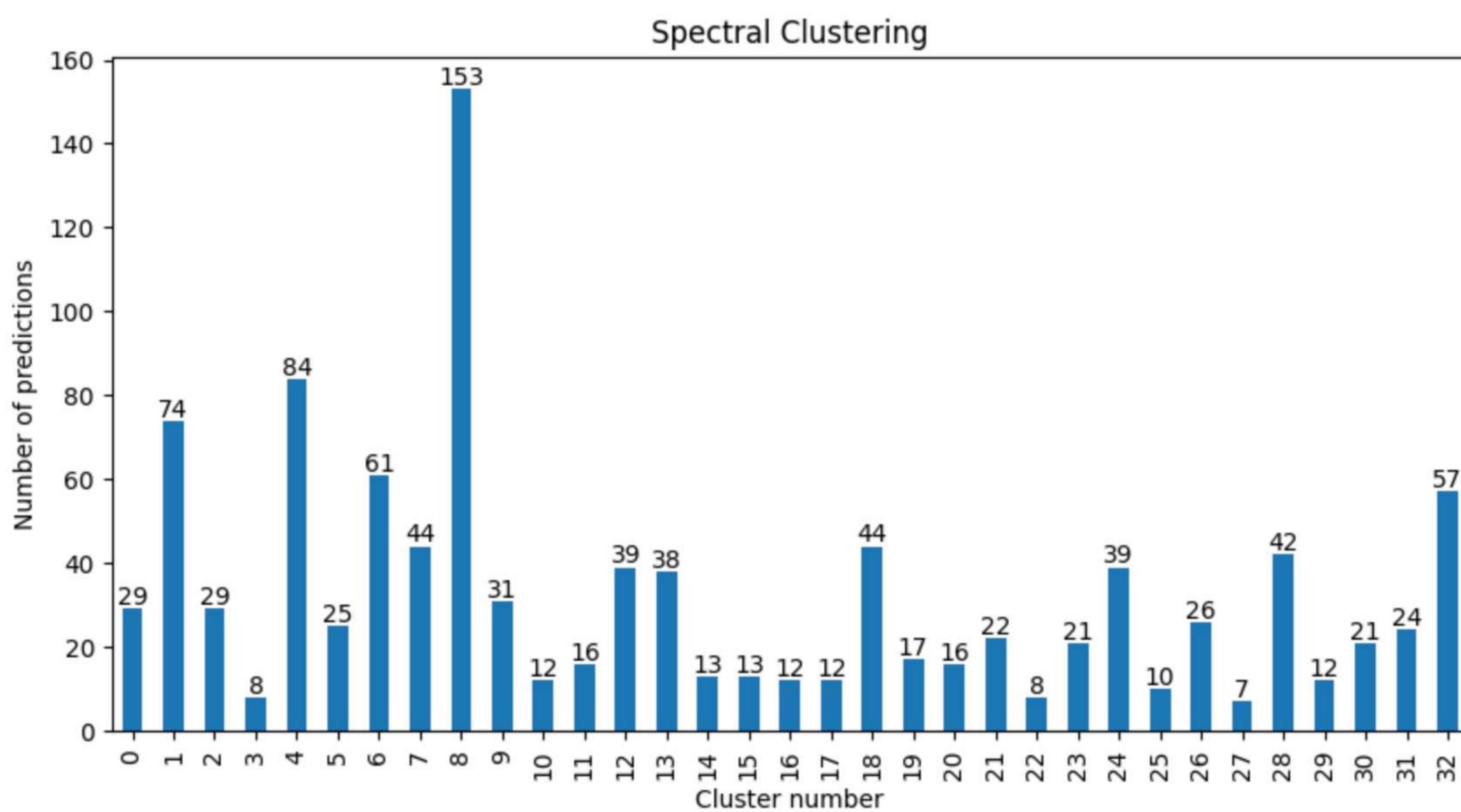
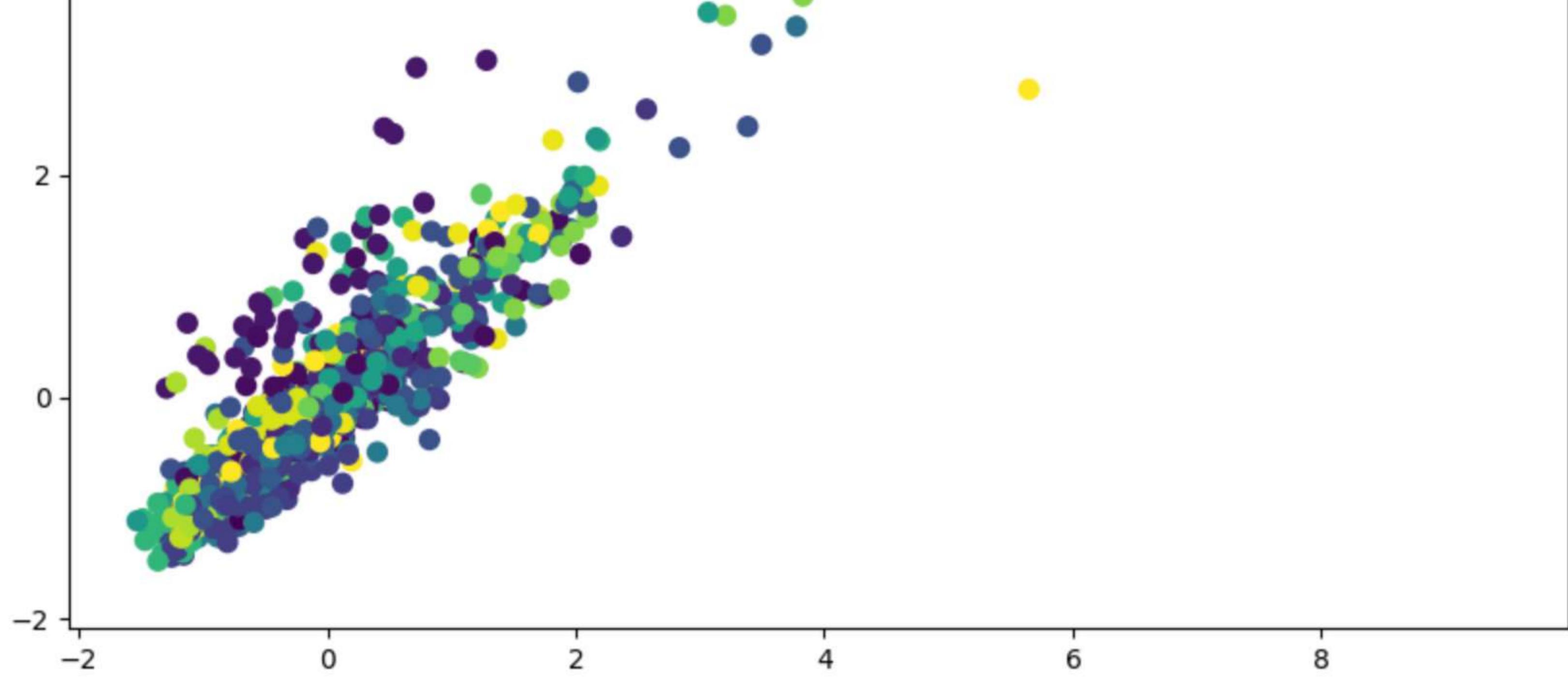
Agglomerative Clustering

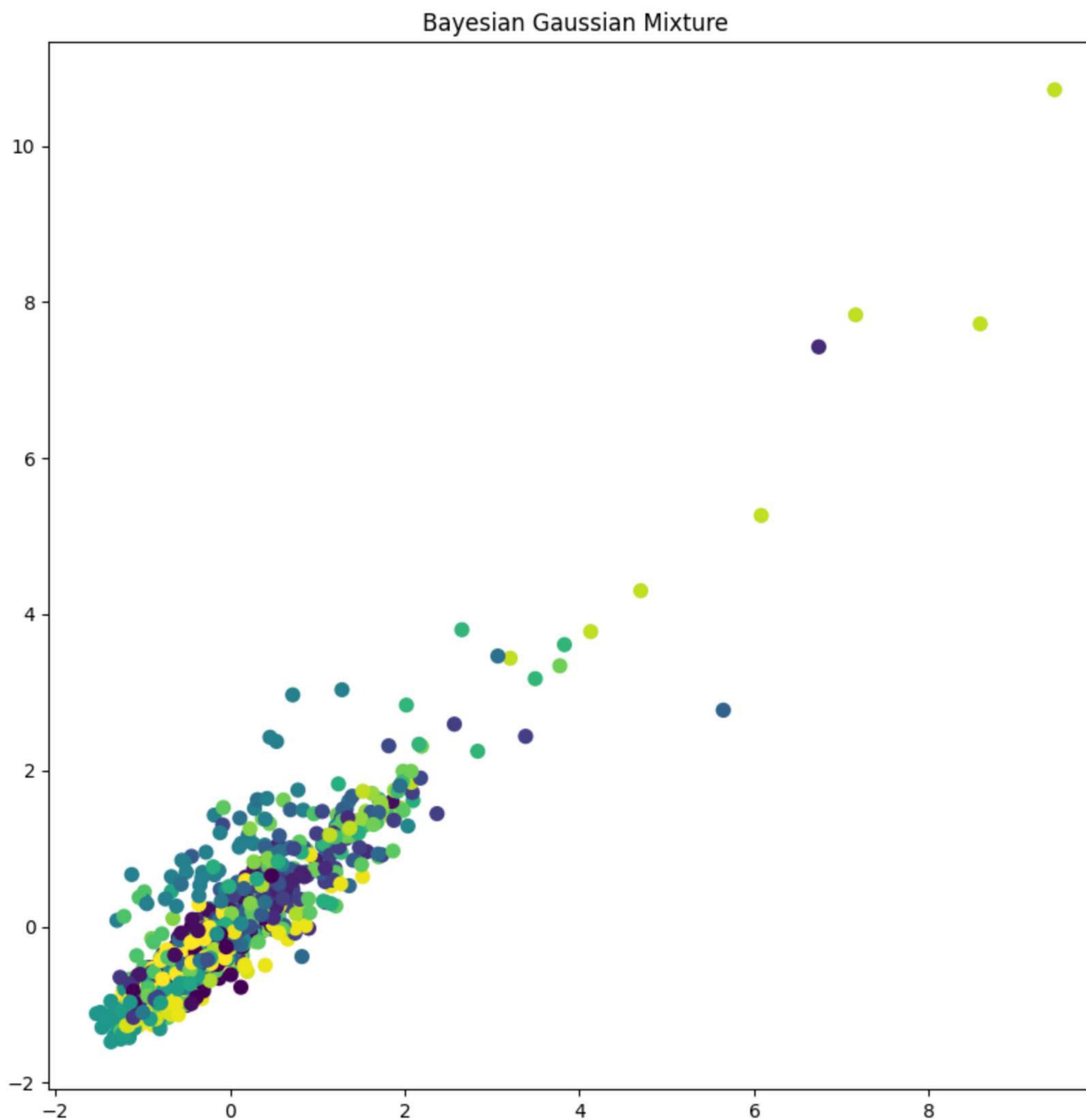
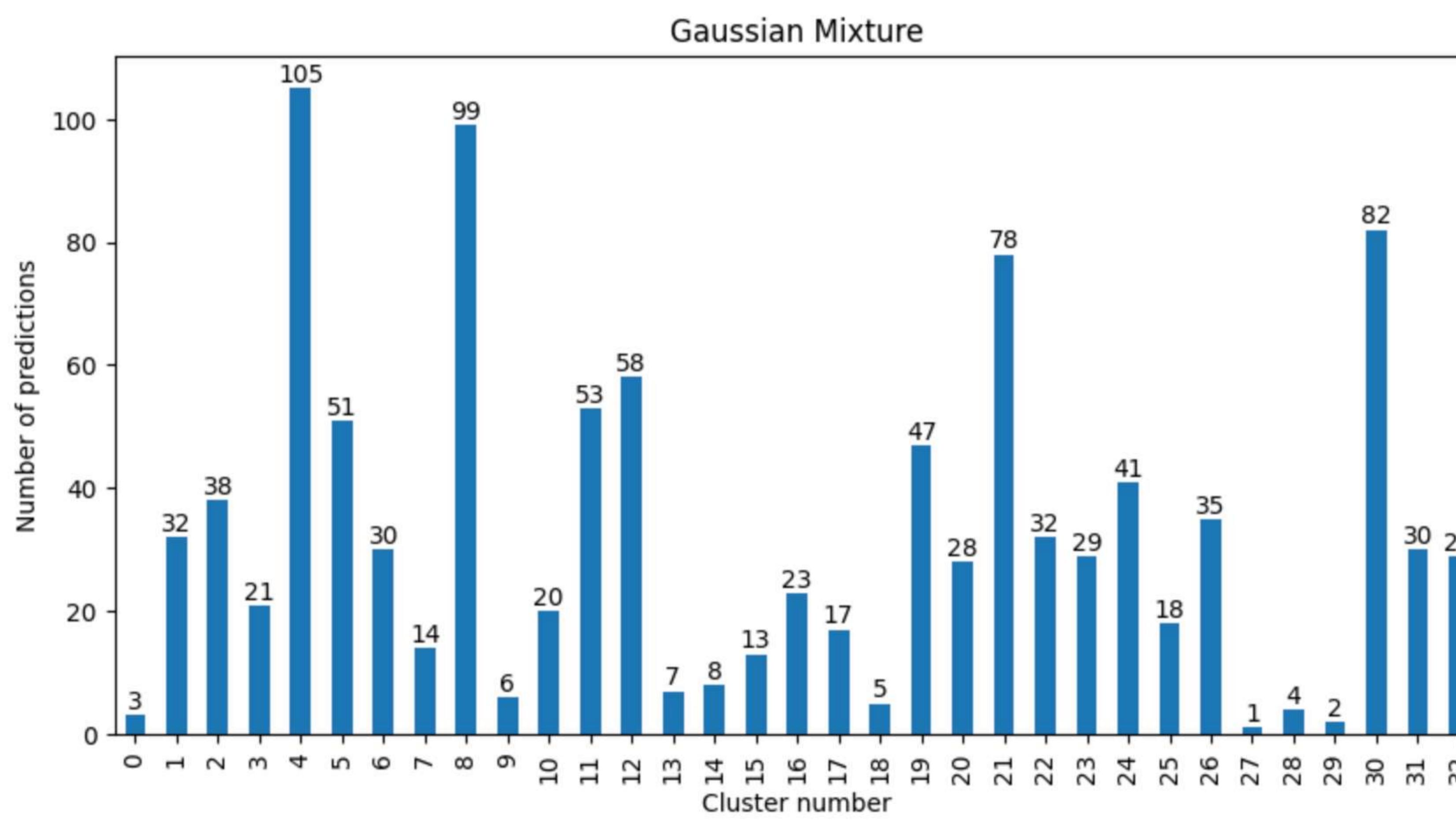
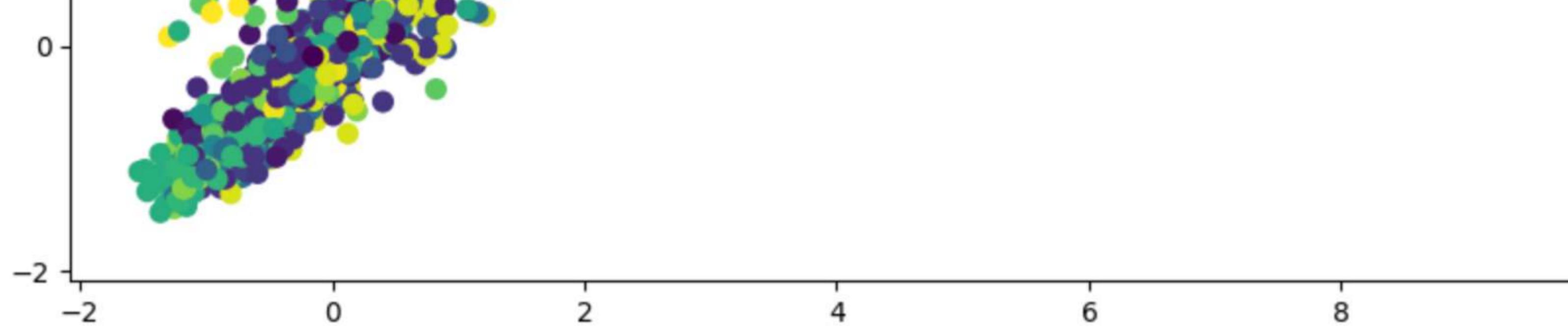


DBSCAN

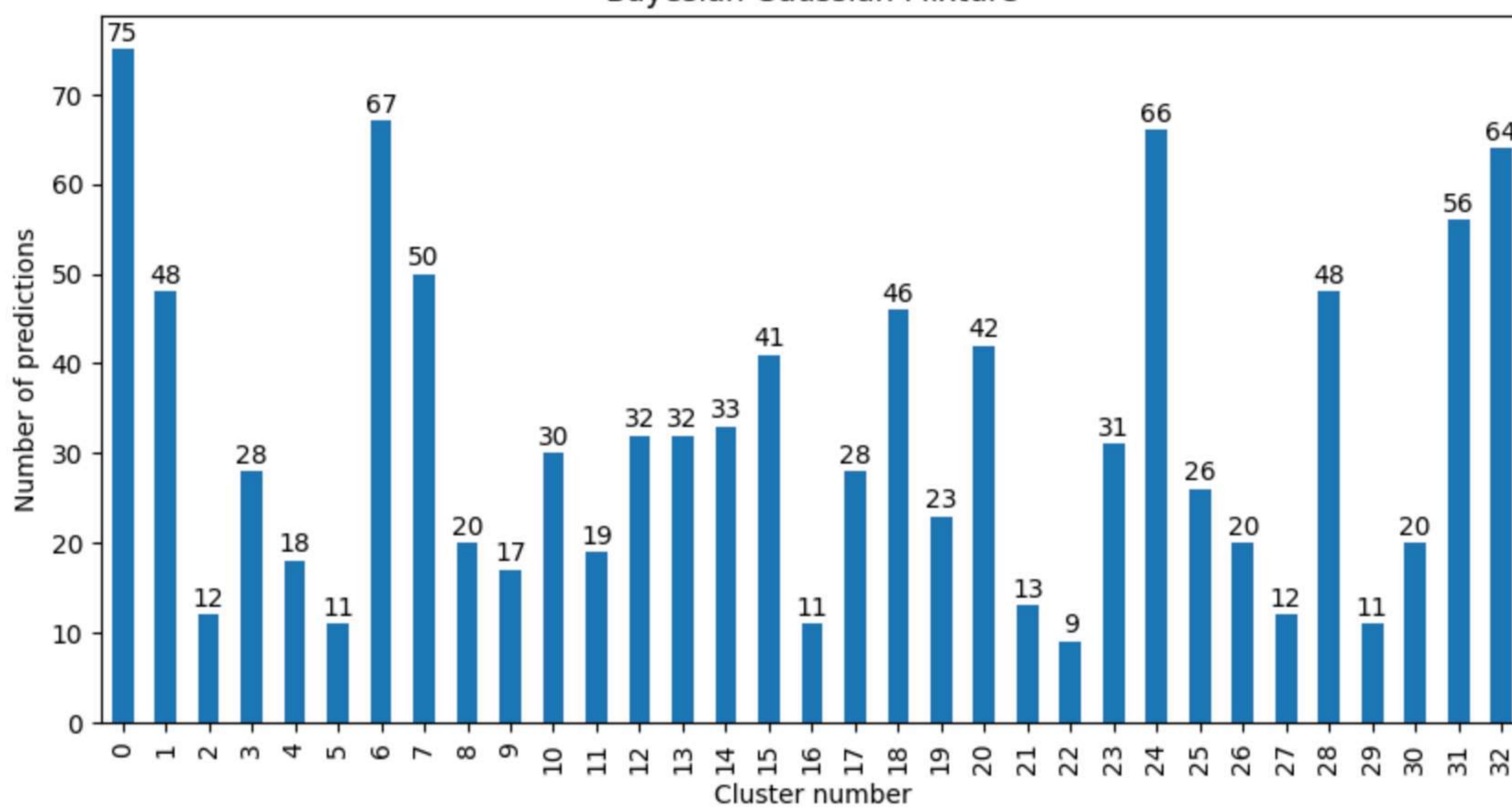








Bayesian Gaussian Mixture



DBSCAN (Gürültülü Uygulamalar İçin Yoğunluğa Dayalı Uzaysal Kümeleme) Algoritması Hakkında

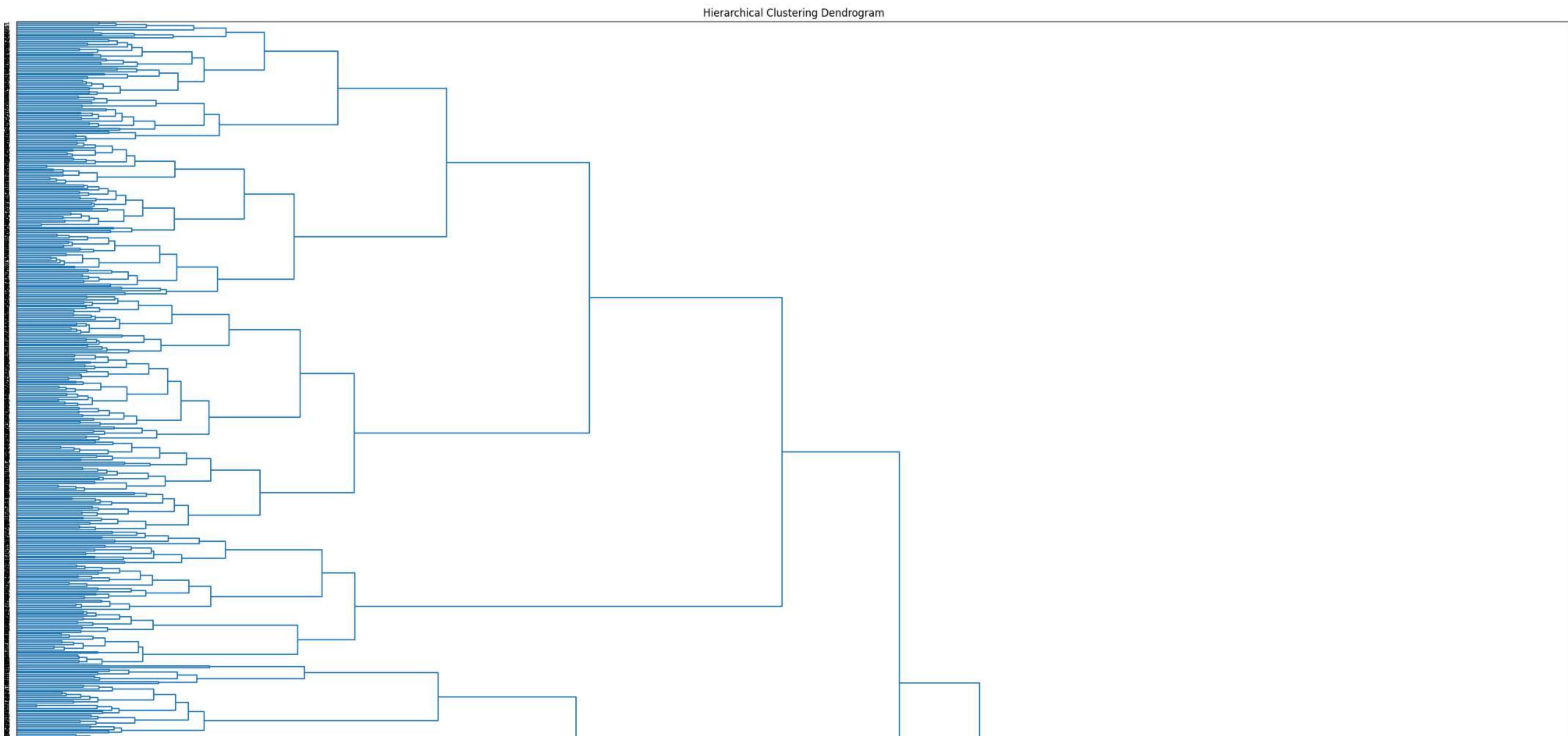
DBSCAN (Density-Based Spatial Clustering of Applications with Noise) algoritması, yoğunluk tabanlı bir algoritmadır. Bu algoritma, veri noktalarını yoğunluklarına göre kümelere ayırır. DBSCAN, belirli bir mesafedeki (Eps parametresi olarak adlandırılan) komşu noktalarının yüksek sayısına sahip "çekirdek noktaları" belirleyerek başlar ve sonra kümeyi, çekirdek noktalarından erişilebilen tüm noktaları da içerecek şekilde genişletir. DBSCAN, yoğunluk tabanlı bir algoritma olduğu için, kümeleme algoritmaları arasında en esnek olanıdır. DBSCAN, küresel olmayan veya açıkça ayrılmış olan veriyi modellemek için özellikle yararlıdır. DBSCAN, veri kümesindeki gürültüyü ayıklamak için de kullanılabilir.

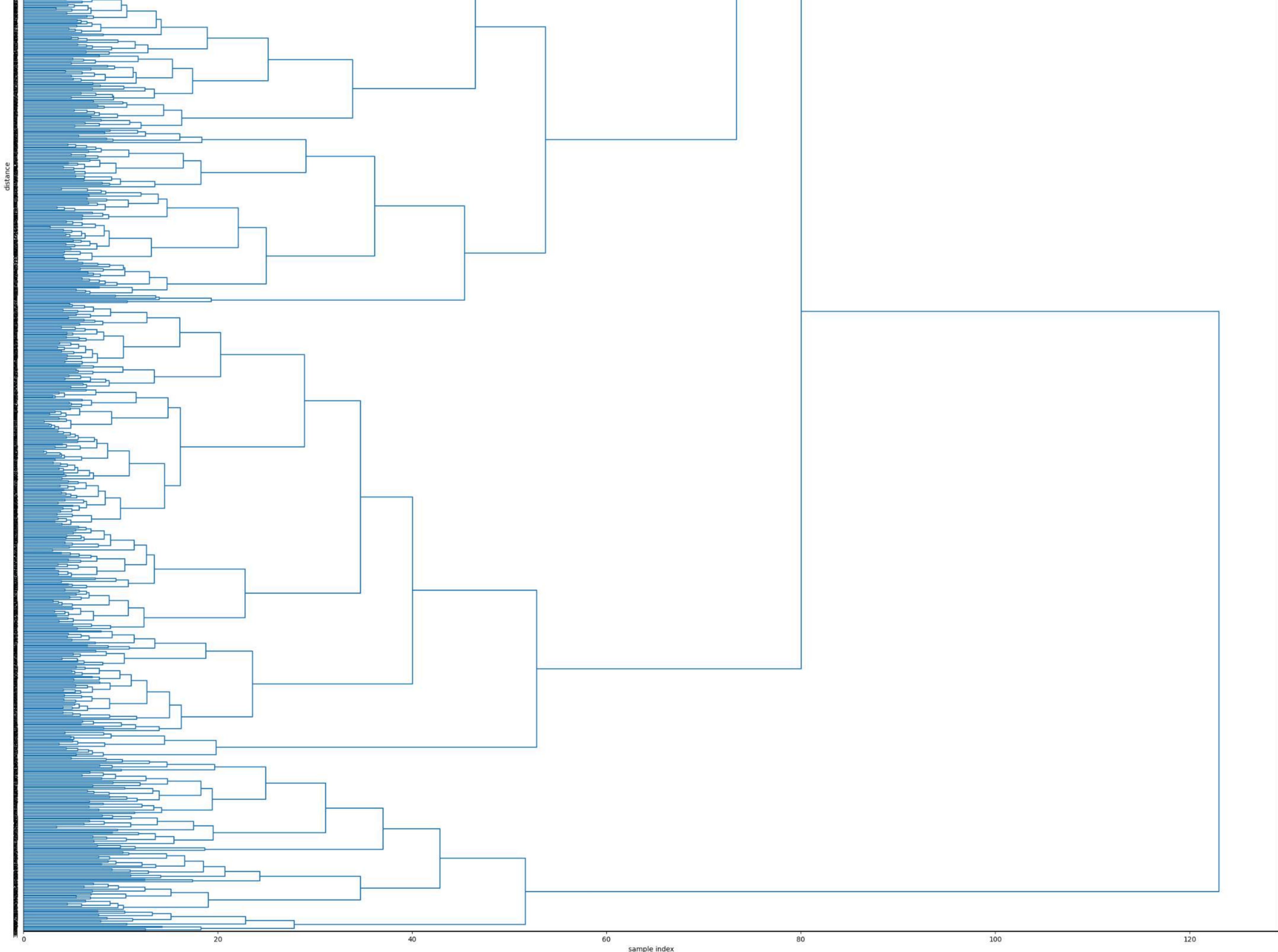
Bu ödev için kullanılan dataseti için DBSCAN algoritması başarısız olmuştur. Bu yüzden bu algoritma ile ilgili detaylı bir inceleme yapılmamıştır.

Hierarchical Clustering

In [17]:

```
def hierarchical_clustering(X, n_clusters, distance_threshold, orientation = 'top'):  
    Z = linkage(X, 'ward')  
    plt.figure(figsize = (n_clusters, 40))  
  
    plt.title('Hierarchical Clustering Dendrogram')  
    plt.xlabel('sample index')  
    plt.ylabel('distance')  
    dendrogram(Z, leaf_rotation = 90., leaf_font_size = 8., color_threshold = distance_threshold, orientation = orientation)  
    plt.axhline(y = distance_threshold, c = 'k')  
    plt.show()  
    hierarchical_clusters = fcluster(Z, n_clusters, criterion = 'maxclust')  
    return hierarchical_clusters  
  
hierarchical_clusters = hierarchical_clustering(X, 33, 0.5, 'right')
```



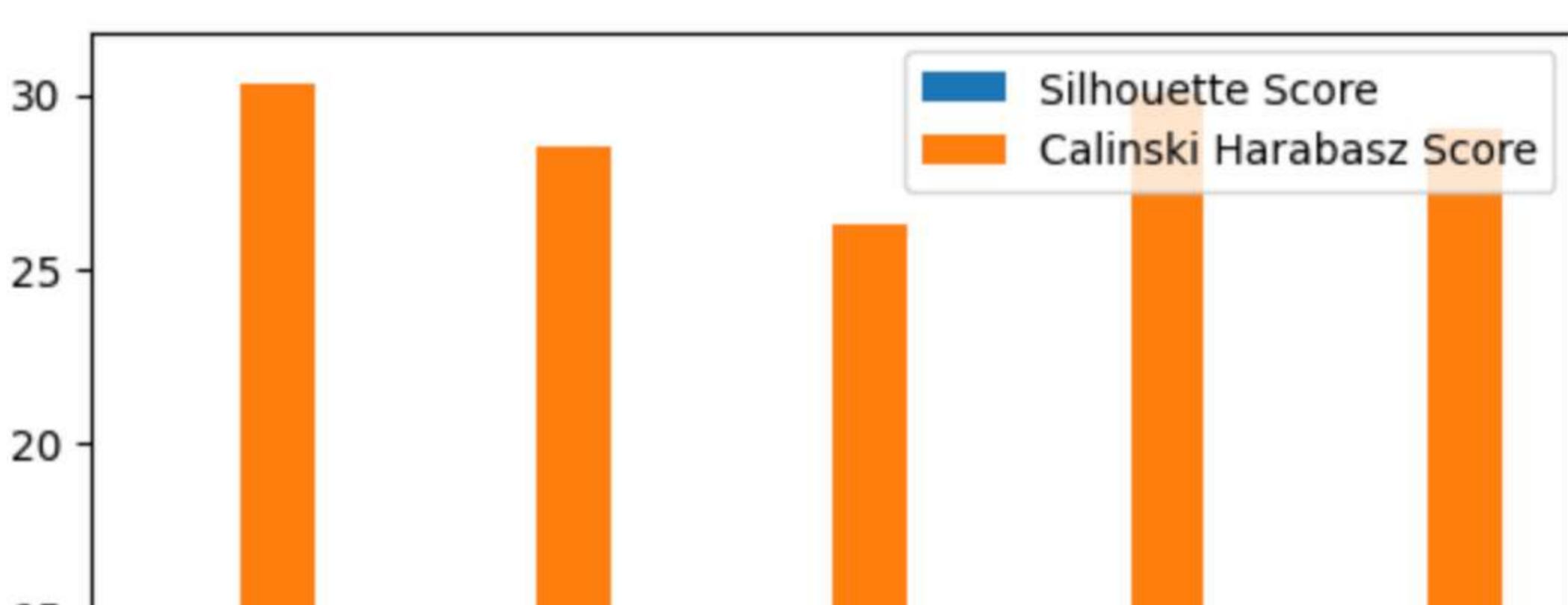


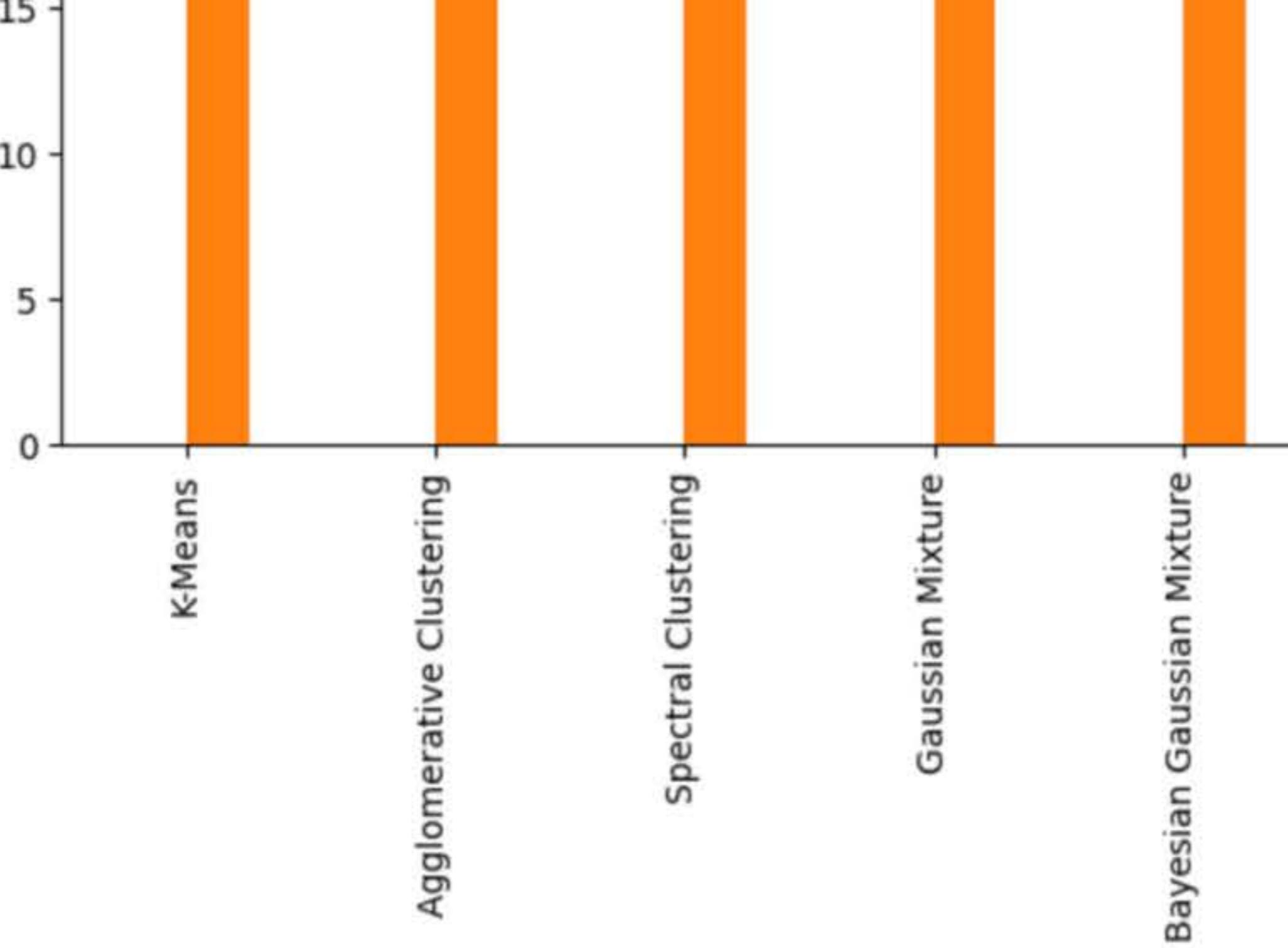
```
In [18]: fig = go.Figure(data = [go.Scatter3d(x = X.iloc[:, 35], y = X.iloc[:, 52], z = X.iloc[:, 55], mode = 'markers', marker = dict(color = hie...  
fig.show()
```

```
In [19]: metrics = {}  
  
for name, model_dict in clusters.items():  
    if name == 'DBSCAN':  
        continue  
    metrics[name] = {'Silhouette Score': silhouette_score(X, model_dict['predictions']),  
                    'Calinski Harabasz Score': calinski_harabasz_score(X, model_dict['predictions'])}  
  
metrics = pd.DataFrame(metrics)  
metrics
```

	K-Means	Agglomerative Clustering	Spectral Clustering	Gaussian Mixture	Bayesian Gaussian Mixture
Silhouette Score	0.051278	0.039112	0.013994	0.048977	0.038981
Calinski Harabasz Score	30.289962	28.532293	26.296655	30.036507	29.015281

```
In [20]: metrics.T.plot.bar()  
plt.xticks(rotation = 90)  
plt.show()
```





c-Kümeleme: Kümeleme yöntemlerinden en başarılı olanların araştırılması ve 3 tanesinin seçilerek detaylı incelenmesi beklenmektedir. Kümeleme sonunda değerlendirmeye aşamasında örneklerin sınıf etiketlerinin kullanılarak çıkan kümelerin etiketlenmesi ve kümeleme başarılarının karşılaştırılması gerekmektedir.

In [21]:

```
metrics = {}

for name, model_dict in clusters.items():
    if name == 'DBSCAN':
        continue
    metrics[name] = {'Adjusted Rand Score': adjusted_rand_score(y, model_dict['predictions']),
                    'Adjusted Mutual Info Score': adjusted_mutual_info_score(y, model_dict['predictions']),
                    'Homogeneity Score': homogeneity_score(y, model_dict['predictions']),
                    'Completeness Score': completeness_score(y, model_dict['predictions']),
                    'V Measure Score': v_measure_score(y, model_dict['predictions'])}

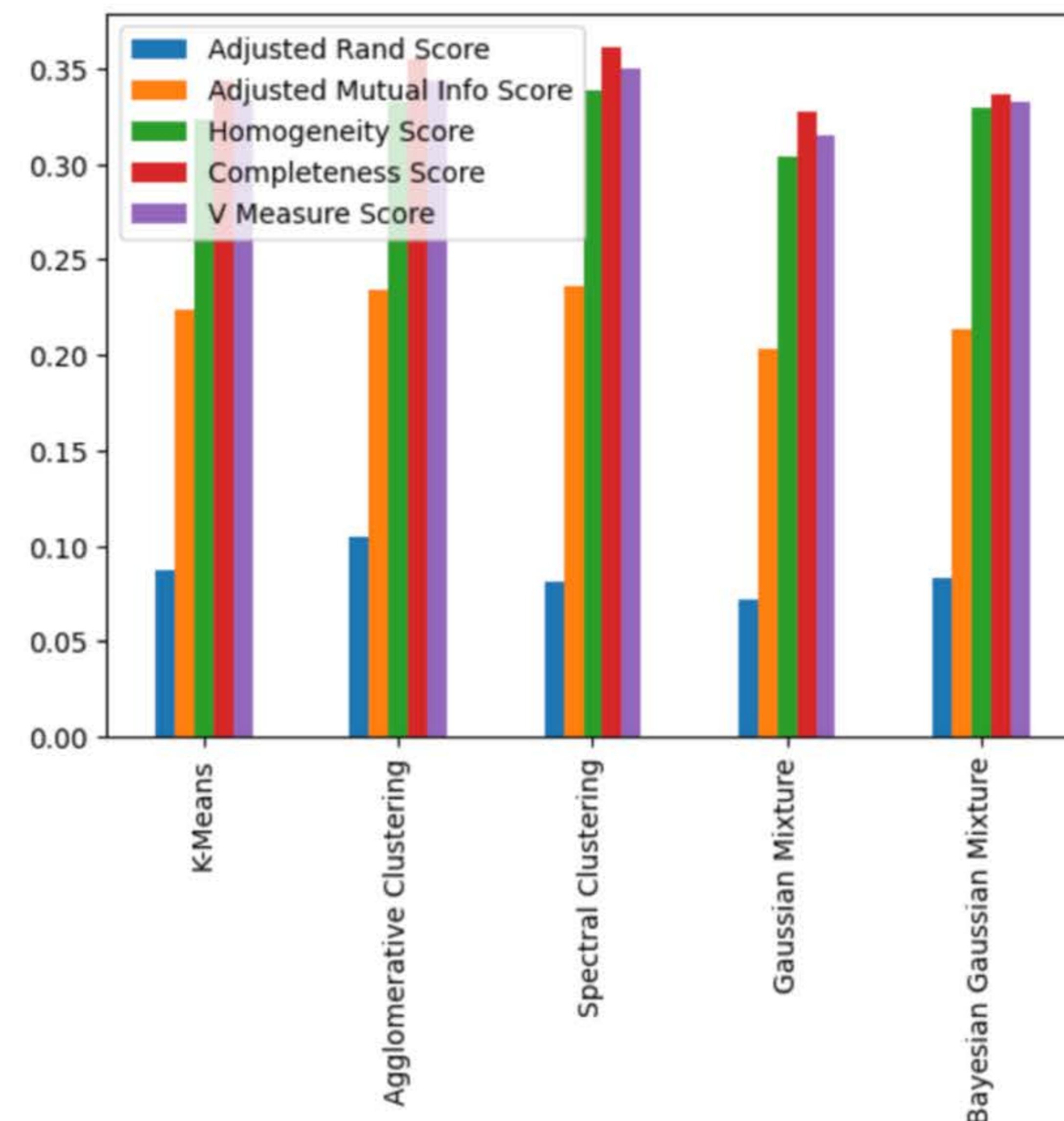
metrics = pd.DataFrame(metrics)
metrics
```

Out[21]:

	K-Means	Agglomerative Clustering	Spectral Clustering	Gaussian Mixture	Bayesian Gaussian Mixture
Adjusted Rand Score	0.087325	0.105278	0.081563	0.071795	0.083710
Adjusted Mutual Info Score	0.223559	0.233733	0.235828	0.203617	0.213605
Homogeneity Score	0.323507	0.332513	0.339127	0.304097	0.329145
Completeness Score	0.344189	0.354798	0.361027	0.327592	0.336088
V Measure Score	0.333528	0.343294	0.349735	0.315407	0.332580

In [22]:

```
metrics.T.plot.bar()
plt.xticks(rotation = 90)
plt.show()
```



In []:

In []: