دانشگاه اصفهان

دانشکده مهندسی کامپیوتر

مبانی یادگیری ماشین

استاد درس: دکتر کیانی

پروژه دوم

دسته‌بندی و خوشه‌بندی برگ‌های گیاهان

شیدا عابدپور

۴۰۰۳۶۲۳۰۲۵

تیر ۱۴۰۳

اولین گام پیش از استفاده از مدل‌ها، پیش پردازش روی داده‌ها است. در این پروژه با توجه به اینکه در بسیاری از مدل‌های خوشه‌بندی و نیز دسته‌بندی از معیار فاصله استفاده می‌شود، نرمال کردن داده‌ها بسیار حائز اهمیت است. با توجه به اینکه داده‌ها و مدل‌ها بکار رفته شده ذاتا خطی نیستند، از روش StandardScaler استفاده شده است.

همچنین با توجه به احتمال وجود وابستگی بین ویژگی‌ها، جهت رسیدن به نتیجه بهتر، از کاهش بعد به روش PCA استفاده شده است.(خوشه‌بندی یک روش بی‌نظارت است بنابراین LDA برای آن بی‌معناست.)

پس از انجام پیش‌پردازش‌های اولیه، لازم است تا ابرپارامترهای بهتر برای هر مدل مشخص شود. برای اینکار از GridSearchCV استفاده شده است. این روش تمام ابرپارامترهای تعیین شده را با یکدیگر ترکیب می‌کند و از میان تمام ترکیب‌های ممکن، بهترین آن‌ها را برمی‌گزیند.

در ابتدا به بررسی ابرپارامترهای مدل‌های مختلف دسته‌بندی پرداخته خواهد شد.

مدل‌های بررسی شده در این پروژه جهت دسته‌بندی برگ‌ها عبارت‌اند از:

- KNN
- SVM
- Guassian Naïve Bayes
- Decision Tree
- Random Forest
- AdaBoost
- MLP classifier

KNN(K Nearest Neighbores)

یک روش ساده برای دسته‌بندی است که نیازی به آموزش ندارد و در فاز تست با توجه به k نزدیکترین همسایه هر داده، لیبل آن را تخمین می‌زند. در این روش تعیین معیار فاصله و نیز تعداد همسایه‌ها بسیار در نتیجه نهایی تاثیرگذار هستند، چراکه k خیلی کوچک می‌تواند منجر به نزدیکی به داده پرت شود و k خیلی بزرگ منجر به کاهش دقت مدل.

```
param_grid = {
    'n_neighbors': [3, 5, 7, 9, 11],
    'weights': ['uniform', 'distance'],
    'metric': ['euclidean', 'manhattan', 'minkowski', 'chebyshev']
}

knn = KNeighborsClassifier()
grid_search = GridSearchCV(knn, param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_score:.2f}")

best_knn = grid_search.best_estimator_
y_pred = best_knn.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.2f}")
```

```
Best Parameters: {'metric': 'manhattan', 'n_neighbors': 3, 'weights': 'distance'}
Best Cross-Validation Score: 0.70
Test Accuracy: 0.87
```

SVM(Support Vector Machine)

این الگوریتم بهترین خط جدا کننده را می‌دهد. یکی از چالش‌های این روش آن است که به درستی ضریب جریمه تنظیم شود.

تنظیم پارامترها برای یک ماشین بردار پشتیبان (SVM) شامل یافتن مقادیر بهینه برای چندین ابرپارامتر کلیدی به منظور به حداکثر رساندن عملکرد مدل است. این پارامترها عبارتند از:

۱. C (پارامتر تنظیم‌سازی):

- توضیح: پارامتر C تعادل بین داشتن خطای کم روی داده‌های آموزشی و کمینه کردن نُرم وزن‌ها را کنترل می‌کند.

- تأثیر: مقدار کوچک C به مدل اجازه می‌دهد که داده‌ها را با خطای بیشتری در کلاس‌بندی قرار دهد، که منجر به یک مدل ساده‌تر (و احتمالا بیش‌تر تعمیم‌پذیر) می‌شود. مقدار بزرگ C سعی می‌کند تا خطای آموزشی را کاهش دهد، اما ممکن است منجر به بیش‌برازش (overfitting) شود.

۲. هسته (Kernel):

- توضیح: نوع هسته تعیین‌کننده نوع مرز تصمیم‌گیری است که مدل استفاده می‌کند.

- خطی (linear): مرز تصمیم‌گیری خطی.

- چندجمله‌ای (poly): مرز تصمیم‌گیری چندجمله‌ای.

- تابع پایه شعاعی (rbf یا Gaussian): مرز تصمیم‌گیری غیرخطی با استفاده از تابع گاوسی.

- سیگموید (sigmoid): مرز تصمیم‌گیری غیرخطی با استفاده از تابع سیگموید.

۳. گاما (Gamma):

- توضیح: گاما تعیین می‌کند که تأثیر یک نمونه آموزشی چقدر دور می‌تواند برسد. این پارامتر فقط برای هسته‌های rbf، poly و sigmoid مهم است.

- تأثیر: مقدار کم گاما به این معنی است که تأثیر هر نمونه آموزشی دورتر است و تصمیمات مرزی نرم‌تر هستند. مقدار زیاد گاما به این معنی است که هر نمونه آموزشی تاثیر زیادی در نزدیکی خود دارد که می‌تواند منجر به بیش‌برازش شود.

۴. درجه (Degree):

- توضیح: درجه تابع هسته چندجمله‌ای (اگر از هسته poly استفاده شود).

- تأثیر: این پارامتر تعیین می‌کند که تابع چندجمله‌ای چه درجه‌ای دارد. به عنوان مثال، درجه ۲ منجر به یک مرز تصمیم‌گیری مربعی می‌شود.

```
param_grid = {
    'C': [0.1, 1, 10, 100],
    'kernel': ['linear', 'poly', 'rbf', 'sigmoid'],
    'gamma': ['scale', 'auto'],
    'degree': [2, 3, 4]  # Only relevant for 'poly' kernel
}

svm = SVC()
grid_search = GridSearchCV(svm, param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_score:.2f}")

best_svc = grid_search.best_estimator_
y_pred = best_svc.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.2f}")
```

```
Best Parameters: {'C': 100, 'degree': 2, 'gamma': 'scale', 'kernel': 'linear'}
Best Cross-Validation Score: 0.74
Test Accuracy: 0.88
```

Gaussian Naive Bayes

این مدل یک توزیع نرمال برای هر ویژگی در نظر می‌گیرد و هر ویژگی را مستقل می‌بینند که آن را در مسائلی که ویژگی‌ها مستقل نباشند، ناکارامد می‌کند.

یکی از ساده‌ترین الگوریتم‌های یادگیری ماشین است و تعداد کمتری از ابرپارامترها را برای تنظیم دارد. ابرپارامتر اصلی در GaussianNB به شرح زیر است:

۱. var_smoothing (نرم‌سازی واریانس):

- توضیح: این پارامتر یک مقدار کوچک از واریانس به داده‌ها اضافه می‌کند تا از تقسیم بر صفر جلوگیری کند.

- تأثیر: مقدار نرم‌سازی واریانس باعث می‌شود که مدل در مواجهه با ویژگی‌هایی که دارای واریانس بسیار کم یا نزدیک به صفر هستند، پایدارتر عمل کند. این موضوع می‌تواند به جلوگیری از مشکلات عددی و بهبود تعمیم‌دهی مدل کمک کند.

تنظیم var_smoothing می‌تواند در بهبود عملکرد مدل در شرایط خاص مؤثر باشد، اما GaussianNB به طور کلی به دلیل سادگی و فرضیات ساده‌ای که درباره توزیع نرمال ویژگی‌ها دارد، نیاز به تنظیمات پیچیده زیادی ندارد.

```
param_grid = {
    'var_smoothing': [1e-9, 1e-8, 1e-7, 1e-6, 1e-5]
}

gnb = GaussianNB()
grid_search = GridSearchCV(gnb, param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_score:.2f}")

best_gnb = grid_search.best_estimator_
y_pred = best_gnb.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.2f}")
```

```
Best Parameters: {'var_smoothing': 1e-05}
Best Cross-Validation Score: 0.70
Test Accuracy: 0.81
```

Decision Tree

درخت تصمیم ویژگی‌ها را مستقل می‌بیند و هر بار بر اساس تمایزبخش‌ترین ویژگی جداسازی را انجام می‌دهد. برای بهینه‌سازی عملکرد یک مدل طبقه‌بندی درخت تصمیم (Decision Tree classifier)، تنظیم چندین ابرپارامتر کلیدی ضروری است. این ابرپارامترها به شرح زیر هستند:

۱. criterion (معیار):

- توضیح: این پارامتر تابعی را که برای اندازه‌گیری کیفیت یک تقسیم استفاده می‌شود، تعیین می‌کند.

- "gini": برای محاسبه (Gini impurity).

- "entropy": برای محاسبه (Information Gain).

۲. max_depth (حداکثر عمق):

- توضیح: حداکثر عمق درخت را مشخص می‌کند. محدود کردن عمق درخت به کنترل بیش‌برازش (overfitting) کمک می‌کند.

- تأثیر: درخت‌های عمیق‌تر می‌توانند جزئیات بیشتری را مدل‌سازی کنند ولی خطر بیش‌برازش دارند. درخت‌های کم‌عمق‌تر تعمیم‌پذیری بهتری دارند ولی ممکن است زیربرازش (underfitting) کنند.

۳. min_samples_split (حداقل نمونه‌ها برای تقسیم):

- توضیح: حداقل تعداد نمونه‌های مورد نیاز برای تقسیم یک گره داخلی را مشخص می‌کند.

- تأثیر: افزایش این مقدار می‌تواند به جلوگیری از بیش‌برازش کمک کند، زیرا از تقسیم گره‌ها با تعداد نمونه‌های بسیار کم جلوگیری می‌کند.

۴. min_samples_leaf (حداقل نمونه‌ها در برگ):

- توضیح: حداقل تعداد نمونه‌های مورد نیاز برای وجود در یک برگ را مشخص می‌کند.

- تأثیر: افزایش این مقدار می‌تواند به کاهش بیش‌برازش کمک کند زیرا از تشکیل برگ‌های کوچک و پرنویز جلوگیری می‌کند.

۵. splitter (استراتژی تقسیم):

- توضیح: استراتژی مورد استفاده برای انتخاب تقسیم در هر گره را تعیین می‌کند.

- "best": انتخاب بهترین تقسیم.

- "random": انتخاب بهترین تقسیم به صورت تصادفی.

```python
param_grid = {
    'criterion': ['gini', 'entropy'],
    'max_depth': [None, 10, 20, 30, 40, 50, 100],
    'min_samples_split': [2, 10, 20, 50],
    'min_samples_leaf': [1, 5, 10, 20],
    'splitter': ['best', 'random']
}

dt = DecisionTreeClassifier(random_state=42)
grid_search = GridSearchCV(dt, param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_score:.2f}")

best_dt = grid_search.best_estimator_
y_pred = best_dt.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.2f}")
```

```
Best Parameters: {'criterion': 'entropy', 'max_depth': None, 'min_samples_leaf': 1, 'min_samples_split': 10, 'splitter': 'best'}
Best Cross-Validation Score: 0.53
Test Accuracy: 0.66
```

Random Forest

این مدل حالت ensemble درخت تصمیم است که از تکنیک bagging استفاده می‌کند. این کار باعث میشود هر مدل بر بخشی از ویژگی‌ها و یا بخشی از داده‌ها تمرکز کند و می‌تواند به نتایج بهتری برسد، اما نه لزوما. در صورتی که عیب و خطاهای مدل‌ها اشتراک زیادی داشته باشد، این روش دقت را بهبود نخواهد بخشید. برای بهینه‌سازی عملکرد یک مدل جنگل تصادفی (Random Forest)، تنظیم چندین ابرپارامتر کلیدی ضروری است. این ابرپارامترها به شرح زیر هستند:

۱. n_estimators (تعداد درختان):

- توضیح: تعداد درختان در جنگل را مشخص می‌کند.

- تأثیر: تعداد بیشتر درختان معمولاً باعث افزایش دقت مدل می‌شود، اما زمان محاسباتی و منابع مورد نیاز نیز افزایش می‌یابد. تعداد کمتر درختان می‌تواند منجر به کاهش دقت و تعمیم‌پذیری مدل شود.

۲. max_depth (حداکثر عمق):

- توضیح: حداکثر عمق درخت‌ها را مشخص می‌کند. محدود کردن عمق درخت‌ها به کنترل بیش‌برازش (overfitting) کمک می‌کند.

- تأثیر: درخت‌های عمیق‌تر می‌توانند جزئیات بیشتری را مدل‌سازی کنند ولی خطر بیش‌برازش دارند. درخت‌های کم‌عمق‌تر تعمیم‌پذیری بهتری دارند ولی ممکن است زیربرازش (underfitting) کنند.

۳. min_samples_split (حداقل نمونه‌ها برای تقسیم):

- توضیح: حداقل تعداد نمونه‌های مورد نیاز برای تقسیم یک گره داخلی را مشخص می‌کند.

- تأثیر: افزایش این مقدار می‌تواند به جلوگیری از بیش‌برازش کمک کند، زیرا از تقسیم گره‌ها با تعداد نمونه‌های بسیار کم جلوگیری می‌کند.

۴. min_samples_leaf (حداقل نمونه‌ها در برگ):

- توضیح: حداقل تعداد نمونه‌های مورد نیاز برای وجود در یک برگ را مشخص می‌کند.

- تأثیر: افزایش این مقدار می‌تواند به کاهش بیش‌برازش کمک کند زیرا از تشکیل برگ‌های کوچک و پرنویز جلوگیری می‌کند.

۵. max_features (حداکثر ویژگی‌ها):

- توضیح: تعداد ویژگی‌هایی که برای یافتن بهترین تقسیم در هر گره مورد نظر قرار می‌گیرند را مشخص می‌کند.

- تأثیر: مقدار کمتر ویژگی‌ها باعث کاهش همبستگی بین درخت‌ها و افزایش تعمیم‌پذیری می‌شود. مقدار بیشتر ویژگی‌ها می‌تواند دقت مدل را افزایش دهد ولی ممکن است منجر به بیش‌برازش شود.

۶. bootstrap (بوت‌استرپ):

- توضیح: مشخص می‌کند که آیا نمونه‌های بوت‌استرپ برای ساخت درخت‌ها استفاده می‌شوند یا خیر.

- تأثیر: استفاده از نمونه‌های بوت‌استرپ (اگر True باشد) به افزایش تنوع درخت‌ها و در نتیجه کاهش بیش‌برازش کمک می‌کند. اگر False باشد، تمام نمونه‌ها در هر درخت استفاده می‌شوند که ممکن است منجر به کاهش دقت مدل شود.

```python
param_grid = {
    'n_estimators': [100, 200, 300],
    'max_depth': [None, 10, 20, 30],
    'min_samples_split': [2, 5, 10],
    'min_samples_leaf': [1, 2, 4, 10, 16],
    'max_features': ['sqrt', 'log2'],
    'bootstrap': [True, False]
}

rf = RandomForestClassifier()
grid_search = GridSearchCV(rf, param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_score:.2f}")

best_rf = grid_search.best_estimator_
y_pred = best_rf.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.2f}")
```

```
Best Parameters: {'bootstrap': True, 'max_depth': None, 'max_features': 'sqrt', 'min_samples_leaf': 1, 'min_samples_split': 2, 'n_estimators': 300}
Best Cross-Validation Score: 0.77
Test Accuracy: 0.88
```

AdaBoost

این مدل ensemble درخت تصمیم با تکنیک boosting است. در این روش هربار هر مدل بر نقاط ضعف مدل قبلی تمرکز می‌کند و هربار داده‌های اشتباه دسته‌بندی‌شده با وزن بیشتری انتخاب می‌شوند و نسبت به جنگل تصادفی هوشمندانه‌تر عمل می‌کند. برای بهینه‌سازی عملکرد مدل AdaBoost، تنظیم چندین ابرپارامتر کلیدی ضروری است. این ابرپارامترها به شرح زیر هستند:

۱. n_estimators (تعداد برآوردگرها):

- توضیح: حداکثر تعداد مدل‌های پایه که در فرآیند بوستینگ استفاده می‌شوند را مشخص می‌کند.

- تأثیر: تعداد بیشتر برآوردگرها معمولاً منجر به دقت بیشتر می‌شود، اما می‌تواند زمان محاسباتی و خطر بیش‌برازش را نیز افزایش دهد. تعداد کمتر برآوردگرها ممکن است منجر به زیربرازش (underfitting) شود.

۲. learning_rate (نرخ یادگیری):

- توضیح: وزنی که به هر کلاس‌بندی‌کننده در هر تکرار بوستینگ اعمال می‌شود را مشخص می‌کند.

- تأثیر: مقدار بالاتر نرخ یادگیری باعث افزایش وزن هر برآوردگر و تسریع فرآیند یادگیری می‌شود، اما ممکن است منجر به ناپایداری مدل و بیش‌برازش شود. مقدار پایین‌تر نرخ یادگیری فرآیند یادگیری را کندتر می‌کند و می‌تواند به پایداری و تعمیم‌پذیری بهتر مدل کمک کند.

۳. base_estimator (برآوردگر پایه):

- توضیح: مدل پایه‌ای که از آن مجموعه بوستینگ ساخته می‌شود را مشخص می‌کند. پیش‌فرض این پارامتر DecisionTreeClassifier است.

- تأثیر: انتخاب مناسب برآوردگر پایه بسیار مهم است زیرا عملکرد کلی مدل به کیفیت برآوردگر پایه وابسته است. معمولاً DecisionTreeClassifier با عمق کم (مثل درخت‌های تصمیم با عمق ۱، که به عنوان "stumps" شناخته می‌شوند) به عنوان برآوردگر پایه استفاده می‌شود، اما می‌توان از سایر مدل‌ها نیز استفاده کرد.

```
param_grid = {
    'n_estimators': [50, 100, 200, 300],
    'learning_rate': [0.01, 0.1, 1, 10],
    'estimator__max_depth': [1, 2, 3, 4, 5]
}

adb = AdaBoostClassifier(estimator=DecisionTreeClassifier())
grid_search = GridSearchCV(adb, param_grid, cv=5, scoring='accuracy')
grid_search.fit(x_train, y_train)

best_params = grid_search.best_params_
best_score = grid_search.best_score_
print(f"Best Parameters: {best_params}")
print(f"Best Cross-Validation Score: {best_score:.2f}")

best_adb = grid_search.best_estimator_
y_pred = best_adb.predict(x_test)

accuracy = accuracy_score(y_test, y_pred)
print(f"Test Accuracy: {accuracy:.2f}")

Best Parameters: {'estimator__max_depth': 5, 'learning_rate': 0.1, 'n_estimators': 200}
Best Cross-Validation Score: 0.73
Test Accuracy: 0.79
```

MLP

با توجه به اینکه تعداد داده‌ها و نیز تعداد ویژگی‌ها در این پروژه زیاد نیست، به تناسب نباید شبکه عصبی پیچیده با تعداد لایه‌ها و نورون‌های زیاد استفاده کرد. همچنین دقت نسبتا بالای مدل knn می‌تواند نشان‌گر عدم پیچیدگی مسئله باشد. بدین منظور یک لایه مخفی در نظر گرفته شد. برای بهینه‌سازی مدل شبکه عصبی در Keras، تنظیم چندین ابرپارامتر کلیدی ضروری است. این ابرپارامترها و تأثیر آن‌ها به شرح زیر هستند:

۱. initial_learning_rate (نرخ یادگیری اولیه):

- توضیح: نرخ یادگیری اولیه مقدار گامی است که در به‌روزرسانی وزن‌ها در طی آموزش استفاده می‌شود.

- تأثیر: نرخ یادگیری بالا می‌تواند منجر به نوسان‌های زیاد در تابع هزینه شود و نرخ یادگیری پایین می‌تواند فرآیند یادگیری را بسیار کند کند.

۲. decay_steps (گام‌های کاهش):

- توضیح: تعداد گام‌هایی که بعد از آن نرخ یادگیری کاهش می‌یابد.

- تأثیر: این پارامتر کنترل می‌کند که نرخ یادگیری چقدر سریع کاهش یابد. کاهش سریع‌تر (مقادیر کمتر) می‌تواند به تعادل بهتری بین اکتشاف و بهره‌برداری کمک کند.

۳. decay_rate (نرخ کاهش):

- توضیح: نرخ کاهش نمایی که به نرخ یادگیری اعمال می‌شود.

- تأثیر: مقدار کمتر نرخ کاهش باعث می‌شود که نرخ یادگیری کندتر کاهش یابد و مقدار بیشتر نرخ کاهش باعث کاهش سریع‌تر نرخ یادگیری می‌شود.

۴. Dense layers (لایه‌های چگال):

- units (واحدها): تعداد نرون‌ها در هر لایه چگال.

- توضیح: تعداد نرون‌ها در هر لایه که باید تنظیم شود.

- تأثیر: تعداد نرون‌های بیشتر می‌تواند مدل را توانمندتر کند ولی خطر بیش‌برازش را افزایش می‌دهد.

۵. activation (فعال‌سازی):

- توضیح: تابع فعال‌سازی که در هر لایه استفاده می‌شود.

- تأثیر: توابع فعال‌سازی مانند 'relu' باعث غیرخطی‌سازی مدل و 'softmax' برای خروجی‌های احتمالی در طبقه‌بندی چندکلاسی استفاده می‌شود.

۶. kernel_regularizer (تنظیم‌گر هسته):

- توضیح: به‌کارگیری تنظیم‌سازی L۲ برای جلوگیری از بیش‌برازش.

- تأثیر: تنظیم‌سازی کمک می‌کند تا وزن‌های بزرگ محدود شوند و مدل بیش‌برازش نکند.

۷. BatchNormalization (نرمال‌سازی بچ):

- توضیح: لایه نرمال‌سازی که باعث سرعت بخشیدن به آموزش و پایداری مدل می‌شود.

- تأثیر: نرمال‌سازی بچ به همگرایی سریع‌تر و عملکرد بهتر مدل کمک می‌کند.

۸. Dropout (افت):

- rate (نرخ): نسبت نرون‌هایی که در هر گام آموزشی غیرفعال می‌شوند.

- توضیح: مکانیزمی برای جلوگیری از بیش‌برازش با غیرفعال کردن تصادفی نرون‌ها.

- تأثیر: مقدار مناسب افت به تعمیم‌دهی بهتر مدل کمک می‌کند ولی مقادیر خیلی بالا می‌تواند باعث کاهش عملکرد مدل شود.

۹. optimizer (بهینه‌ساز):

- توضیح: الگوریتمی که برای به‌روزرسانی وزن‌ها استفاده می‌شود. در اینجا از Adam با نرخ یادگیری تنظیم‌شده استفاده می‌شود.

- تأثیر: بهینه‌ساز Adam با تنظیم نرخ یادگیری کمک می‌کند تا فرآیند آموزش سریع‌تر و پایدارتری داشته باشید.

۱۰. loss (تابع هزینه):

- توضیح: تابعی که تفاوت بین خروجی‌های پیش‌بینی شده و واقعی را محاسبه می‌کند. در اینجا از 'categorical_crossentropy' برای طبقه‌بندی چندکلاسی استفاده می‌شود.

- تأثیر: انتخاب صحیح تابع هزینه باعث می‌شود که مدل به درستی آموزش ببیند.

۱۱. early_stopping (توقف زودهنگام):

- توضیح: مکانیسمی برای جلوگیری از آموزش بیش‌ازحد با متوقف کردن آموزش زمانی که عملکرد مدل بر روی مجموعه اعتبارسنجی بهبود نمی‌یابد.

- تأثیر: جلوگیری از بیش‌برازش و صرفه‌جویی در زمان آموزش.

تحلیل و مقایسه عملکرد الگوریتم‌های مختلف دسته‌بندی

	Classifier	Accuracy	Precision	Recall	F1 Score
0	MLP	0.941176	0.964461	0.941176	0.942577
1	Random Forest	0.882353	0.928922	0.882353	0.886275
2	SVM	0.882353	0.922059	0.882353	0.881559
3	KNN	0.867647	0.922059	0.867647	0.874767
4	AdaBoost	0.823529	0.887255	0.823529	0.831956
5	Gaussian Naive Bayes	0.808824	0.862745	0.808824	0.805952
6	Decision Tree	0.647059	0.658824	0.647059	0.615920

MLP بالاترین عملکرد را دارد. این مدل قادر است الگوهای پیچیده را به خوبی یاد بگیرد و طبقه‌بندی کند، که دلیل اصلی برتری آن نسبت به سایر الگوریتم‌ها است. این نشان می‌دهد که MLP قادر است تعادل خوبی بین دقت و بازخوانی حفظ کند و به همین دلیل نمره F۱ بالایی دارد.

Random Forestبه دلیل ترکیب چندین درخت تصمیم‌گیری و کاهش واریانس، عملکرد خوبی دارد. این مدل تعادل خوبی بین دقت و بازخوانی دارد که منجر به نمره F۱ بالا می‌شود. با این حال، کمی پایین‌تر از MLPعمل کرده است.

SVMبا دقت بالا و دقت مناسب، عملکرد خوبی دارد. این مدل در پیدا کردن مرزهای تصمیم‌گیری بهینه بسیار قدرتمند است و می‌تواند در مواردی که داده‌ها به خوبی تفکیک‌پذیر هستند، عملکرد بسیار خوبی داشته باشد.

KNNعملکرد خوبی دارد، اما کمی پایین‌تر از SVM و Random Forest قرار گرفته است. این مدل به دلیل ساده بودن و وابستگی به تعداد k همسایه نزدیک، ممکن است در مواجهه با داده‌های پر نویز دچار مشکل شود.

AdaBoostبا ترکیب مدل‌های ضعیف‌تر و تقویت آن‌ها، عملکرد مناسبی دارد، اما کمی پایین‌تر از KNN قرار گرفته است. این مدل در صورتی که داده‌های نویزی زیادی وجود داشته باشد، ممکن است دچار مشکل شود.

Gaussian Naive Bayesبه دلیل فرضیات ساده‌ای که درباره توزیع داده‌ها دارد، معمولاً عملکرد کمتری نسبت به مدل‌های پیچیده‌تر دارد. این مدل برای داده‌هایی که توزیع نرمال دارند مناسب‌تر است.

Decision Treeبه دلیل توانایی بیش‌برازش بر روی داده‌های آموزشی و عدم تعمیم‌پذیری خوب، پایین‌ترین عملکرد را دارد. این مدل به تنهایی و بدون تنظیمات خاص ممکن است دچار بیش‌برازش شدید شود.

نتیجه‌گیری:

MLP بهترین عملکرد را دارد که نشان می‌دهد شبکه‌های عصبی با لایه‌های چندگانه قادر به یادگیری و تعمیم‌دهی بهتر الگوهای پیچیده هستند.

Random Forest و SVM نیز عملکرد بسیار خوبی دارند که نشان‌دهنده قدرت آن‌ها در ترکیب مدل‌ها و پیدا کردن مرزهای تصمیم‌گیری است.

KNN و AdaBoost عملکرد مناسبی دارند ولی نسبت به مدل‌های پیچیده‌تر کمی ضعیف‌تر عمل می‌کنند.

Gaussian Naive Bayes به دلیل فرضیات ساده‌تر در مورد توزیع داده‌ها عملکرد کمتری دارد.

Decision Tree به تنهایی عملکرد کمتری دارد که نشان‌دهنده نیاز به روش‌های ترکیبی مانند Random Forest برای بهبود عملکرد است.

در ادامه به بررسی مدل‌های خوشه‌بندی پرداخته خواهد شد.

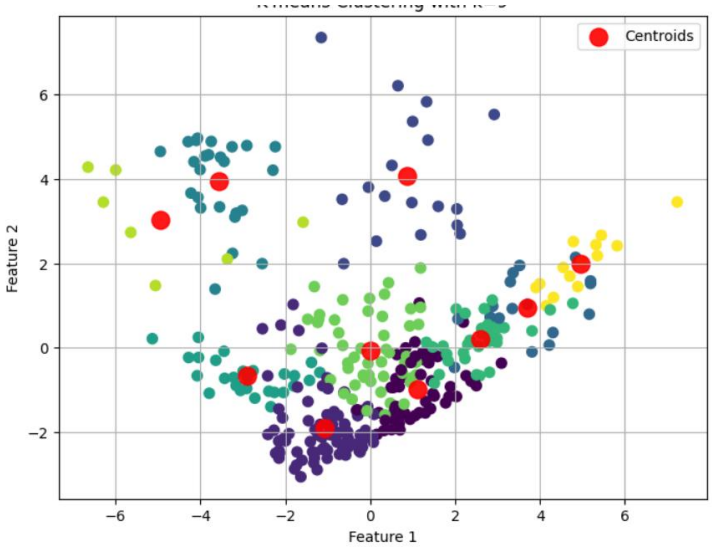
مدل‌های بررسی شده عبارت‌اند از:

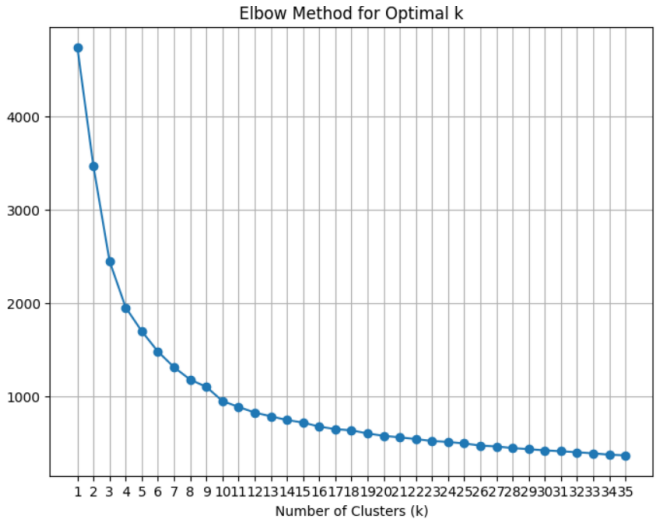
- K-Means
- AgglomerativeClustering
- DBSCAN
- OPTICS
- Gussian Mixture

K-Means

از ساده‌ترین روش‌های خوشه‌بندی است. تعیین معیار فاصله و نیز تعداد خوشه‌ها از چالش‌های آن است. به دلیل وابستگی زیاد به مراکز اولیه ناپایدار است و همچنین نسبت به داده نویزی بسیار حساس است.

جهت تعیین مقدار مناسب تعداد خوشه‌ها از تکنیک elbow استفاده شده است. به ازای kهای مختلف فاصله درون کلاستری محاسبه می‌شود و با توجه به نتایج، ۱۰ انتخاب شد.





Agglomerative Clustering

1.n_clusters (تعداد خوشه‌ها):

-توضیح: تعداد خوشه‌هایی که باید پیدا شود.

-تأثیر: تعداد کمتر خوشه‌ها ممکن است باعث ادغام بیش از حد داده‌ها و تعداد بیشتر خوشه‌ها ممکن است باعث پراکندگی بیش از حد داده‌ها شود.

2.Linkages (انواع لینک‌ها):

-Ward: کمینه‌سازی مجموع مربعات فاصله‌ها درون هر خوشه.

-Complete: بیشینه فاصله بین نقاط در خوشه‌ها.

-Average: میانگین فاصله بین نقاط در خوشه‌ها.

-Single: کمینه فاصله بین نقاط در خوشه‌ها.

تأثیر: انواع مختلف لینک‌ها می‌توانند منجر به تشکیل خوشه‌های مختلف شوند، بسته به نحوه تعریف فاصله بین خوشه‌ها.

3.Affinities (معیارهای فاصله):

-Euclidean: فاصله اقلیدسی.

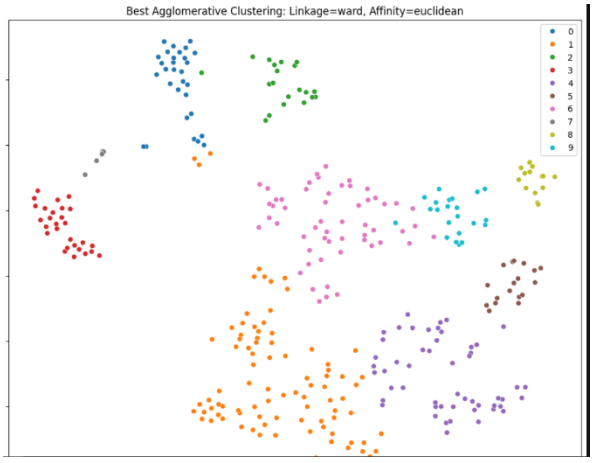
-l1: فاصله مانهایتن (L1).

-l2: فاصله اقلیدسی (L2).

-Manhattan: فاصله مانهایتن.

-Cosine: فاصله کسینوسی.

-تأثیر: انتخاب معیار فاصله مناسب برای داده‌های مختلف می‌تواند تأثیر قابل‌توجهی بر کیفیت خوشه‌بندی داشته باشد.



DBSCAN

بر اساس تراکم نمونه‌ها در فضای ویژگی‌ها عمل می‌کند. این الگوریتم به صورت خاص برای داده‌هایی که ممکن است شامل نویز باشند و خوشه‌هایی با اشکال و اندازه‌های متفاوت داشته باشند مناسب است.

Eps.1 (اپسیلون):

-توضیح: حداکثر فاصله بین دو نمونه برای اینکه یکی به عنوان همسایه دیگری در نظر گرفته شود.

-تأثیر: تعیین کننده اندازه ناحیه همسایگی. مقدار کمتر باعث تشکیل خوشه‌های کوچکتر و بیشتر باعث تشکیل خوشه‌های بزرگتر می‌شود. انتخاب نادرست می‌تواند منجر به تشخیص نویز یا ایجاد خوشه‌های غیرواقعی شود.

min_samples.۲ (حداقل نمونه‌ها):

-توضیح: حداقل تعداد نمونه‌ها در یک ناحیه eps برای اینکه نقطه به عنوان نقطه هسته‌ای در نظر گرفته شود.

-تأثیر: تعیین کننده تراکم مورد نیاز برای تشکیل یک خوشه. مقدار کمتر منجر به تشکیل خوشه‌های بیشتر و با تراکم کمتر می‌شود، در حالی که مقدار بیشتر نیاز به تراکم بالاتر دارد و ممکن است خوشه‌های کوچکتر و نویز بیشتری تشخیص داده شود.

```
eps_values = [0.1, 0.2, 0.3, 0.5, 1, 2, 3, 4, 4.25, 4.5, 4.75, 4.9]
min_samples_values = [3, 5, 7, 8, 9, 10, 11, 12]

best_eps_dbscan = None
best_min_samples_dbscan = None
best_score_dbscan = -1

for eps in eps_values:
    for min_samples in min_samples_values:
        dbscan = DBSCAN(eps=eps, min_samples=min_samples)
        labels_dbscan = dbscan.fit_predict(x_train)

        if len(set(labels_dbscan)) > 1:
            silhouette_avg_dbscan = silhouette_score(x_train, labels_dbscan)
            if silhouette_avg_dbscan > best_score_dbscan:
                best_score_dbscan = silhouette_avg_dbscan
                best_eps_dbscan = eps
                best_min_samples_dbscan = min_samples

print("Best parameters for DBSCAN:")
print(f"eps={best_eps_dbscan}, min_samples={best_min_samples_dbscan}")
print(f"Best silhouette score: {best_score_dbscan:.2f}")
```

```
Best parameters for DBSCAN:
eps=4.25, min_samples=8
Best silhouette score: 0.52
```

OPTICS (Ordering Points To Identify the Clustering Structure)

یک الگوریتم خوشه‌بندی است که بر پایه ترتیب اولویت دادن به نقاط برای شناسایی ساختار خوشه‌ها عمل می‌کند. این الگوریتم به طور خاص برای تشخیص خوشه‌هایی با اندازه‌ها و شکل‌های متفاوت، در مواجهه با داده‌هایی که ممکن است حاوی نویز باشند، مناسب است.

۱.Eps (اپسیلون):

-توضیح: حداکثر فاصله بین دو نقطه برای تشخیص همسایگی.

-تأثیر: این پارامتر تعیین می‌کند که چقدر دور از یک نقطه هستیم تا به عنوان همسایه آن در نظر گرفته شود. افزایش این مقدار منجر به ادغام خوشه‌های بیشتر و کمتر شناسایی نویز می‌شود.

۲.min_samples (حداقل نمونه‌ها):

-توضیح: حداقل تعداد نقاط مورد نیاز برای تشکیل یک خوشه یا نقطه هسته‌ای.

-تأثیر: افزایش این پارامتر باعث شناسایی خوشه‌های کوچکتر و نادر اما پایدارتر می‌شود، در حالی که کاهش آن ممکن است باعث شناسایی خوشه‌های بزرگتر و با تراکم بالاتر شود.

۳.xi (پارامتر xi):

-توضیح: پارامتری که میزان انحراف مجاز از یک نقطه هسته‌ای را کنترل می‌کند.

-تأثیر: این پارامتر می‌تواند تأثیر بر ترتیب و دقت شناسایی خوشه‌ها داشته باشد. انتخاب مقدار مناسب برای xi می‌تواند منجر به شناسایی دقیق‌تر خوشه‌ها و کاهش اثر نویز در نتایج خوشه‌بندی شود.

```
eps_values = [1e-5, 1e-4, 0.001]
min_samples_values = [20, 25, 30, 40, 45]
xi_values = np.linspace(0.01, 0.1, 10)

best_eps_optics = None
best_min_samples_optics = None
best_xi_optics = None
best_score_optics = -1

for eps in eps_values:
    for min_samples in min_samples_values:
        for xi in xi_values:
            optics = OPTICS(eps=eps, min_samples=min_samples, xi=xi)
            labels_optics = optics.fit_predict(x_train)

            if len(set(labels_optics)) > 1:
                silhouette_avg_optics = silhouette_score(x_train, labels_optics)
                if silhouette_avg_optics > best_score_optics:
                    best_score_optics = silhouette_avg_optics
                    best_eps_optics = eps
                    best_min_samples_optics = min_samples
                    best_xi_optics = xi

print("\nBest parameters for OPTICS:")
print(f"eps={best_eps_optics}, min_samples={best_min_samples_optics}, xi={best_xi_optics}")
print(f"Best silhouette score: {best_score_optics:.2f}")
```

```
Best parameters for OPTICS:
eps=1e-05, min_samples=40, xi=0.030000000000000006
Best silhouette score: 0.35
```

Guassian Mixture

برای هر خوشه یک گاوسی فیت میکند و برای مسائلی مناسب است که دارای توزیع گاوسی باشند.

n_components.1:

-توضیح: تعداد توزیع‌های گوسی (خوشه‌ها) که برای مدل‌سازی داده‌ها استفاده می‌شود.

-تأثیر: افزایش این پارامتر می‌تواند منجر به شناسایی خوشه‌های بیشتر و دقیق‌تری از داده‌ها شود، در حالی که کاهش آن ممکن است باعث ادغام خوشه‌های مشابه و کاهش دقت خوشه‌بندی شود.

covariance_type.۲:

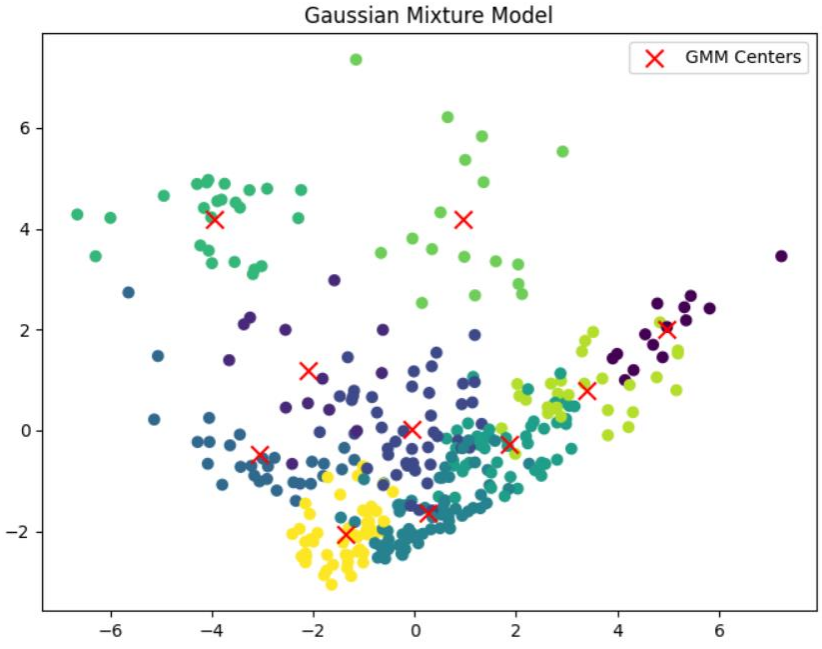
-توضیح: نوع ماتریس کوواریانس برای هر خوشه، که نحوهٔ ارتباط متغیرها را در هر خوشه تعیین می‌کند.

-تأثیر: انتخاب نوع مناسب کوواریانس می‌تواند به بهبود عملکرد مدل کمک کند. مقادیر ممکن شامل "full" (ماتریس کوواریانس کامل)، "tied" (ماتریس کوواریانس مشترک برای همهٔ خوشه‌ها)، "diag" (ماتریس کوواریانس قطری) و "spherical" (کوواریانس کروی) هستند.

random_state.۳:

-توضیح: برای تولید نتایج قابل تکرار، این پارامتر برای تنظیم تولید اعداد تصادفی در مدل GMM استفاده می‌شود.

-تأثیر: تنظیم این پارامتر باعث می‌شود که نتایج آموزش مدل در هر بار اجرا تکرارپذیر باشد.



مقایسه مدل‌های خوشه‌بندی مختلف

برای مقایسه مدل‌های خوشه‌بندی مختلف و تحلیل علل احتمالی برتری یک مدل نسبت به دیگری، معیارهای مختلفی می‌توانند استفاده شوند. در اینجا معیاری که برای مقایسه استفاده شده، Silhouette Score است که نشان می‌دهد که خوشه‌بندی هر مدل چقدر جداپذیر و خوشه‌ها درونی همجواری دارند. به طور کلی، امتیاز Silhouette Score نزدیک به ۱ بهترین نتیجه را نشان می‌دهد و امتیازی نزدیک به ۱- نشان‌دهنده خوشه‌بندی ضعیف‌تر است.

	0	1
0	DBSCAN	0.516671
1	OPTICS	0.347394
2	KMeans	0.327922
3	Agglomerative	0.316300
4	GMM	0.256321

بر اساس امتیازهای داده شده در جدول، به نظر می‌رسد که DBSCAN عملکرد بهتری نسبت به دیگر مدل‌ها داشته باشد، زیرا دارای امتیاز Silhouette Score بالاتری است. در مقابل، GMM دارای امتیاز کمترین Silhouette Score است که نشان‌دهنده کیفیت پایین‌تر خوشه‌بندی آن می‌باشد.

علت احتمالی برتری: DBSCAN بر اساس چگالی نقاط و نواحی مختلف، خوشه‌بندی را انجام می‌دهد. این الگوریتم قادر است خوشه‌هایی با شکل‌ها و اندازه‌های متفاوت را تشخیص دهد و نقاط نویز را تشخیص داده و از خوشه‌بندی آنها خارج کند. علاوه بر این، DBSCAN به طور خودکار تعداد خوشه‌ها را تعیین می‌کند و نیازی به تعیین تعداد خوشه‌ها مانند برخی از مدل‌های دیگر ندارد.

بنابراین، برای داده‌هایی که خوشه‌های غیر منظم و با اندازه‌ها و اشکال متنوع دارند، DBSCANممکن است عملکرد بهتری داشته باشد زیرا به طور خاص برای این نوع داده‌ها طراحی شده است و به طور خودکار می‌تواند تعداد و شکل خوشه‌ها را تشخیص دهد.

برای بررسی ارتباط خوشه‌های نهایی با دسته‌ها، می‌توانیم برای هر خوشه، تعداد اعضای هر دسته را بررسی کنیم. خوشه‌بندی از ۰ تا ۹ شماره‌گذاری شده است و دسته‌ها از ۱ تا ۳۶ شماره‌گذاری شده‌اند. برای هر خوشه، تعداد داده‌هایی که به هر دسته تعلق دارند را بررسی می‌کنیم.

بر اساس جدول ارائه شده، می‌توانیم نتایج زیر را بگیریم:

-برای خوشه ۰:

- دسته‌های ۱۱، ۱۵، ۲۲، و ۳۶ هر کدام یک نمونه.

- برای خوشه ۱:

- دسته‌های ۲، ۳، ۴، ۵، ۸، ۱۰، ۱۲، ۱۳، ۲۵، ۲۷، ۲۸، ۳۰، ۳۱، ۳۲، و ۳۳

- برای خوشه ۲:

- دسته‌های ۳، ۶، ۷، ۱۳، ۱۴، ۱۵، ۲۲، ۲۳، ۲۴، ۲۵، ۲۶، ۲۷، ۲۸، ۳۰، ۳۱، ۳۲، و ۳۳

- برای خوشه ۳:

- دسته‌های ۱، ۲، ۳، ۴، ۵، ۶، ۷، ۸، ۹، ۱۰، ۱۱، ۱۲، ۱۳، ۱۴، ۱۵، ۲۲، ۲۳، ۲۴، ۲۵، ۲۶، ۲۷، ۲۸، ۲۹، ۳۰، ۳۱، ۳۲، ۳۳، ۳۴، ۳۵، و ۳۶ دارای نمونه‌هایی هستند.

- برای خوشه ۴:

- دسته‌های ۵ و ۳۴ هر کدام یک نمونه.

- برای خوشه ۵:

- دسته‌های ۶ و ۳۶ هر کدام یک نمونه.

- برای خوشه ۶:

- دسته‌های ۶ و ۳۶ هر کدام یک نمونه.

- برای خوشه ۷:

- دسته‌های ۵، ۷، ۱۱، ۱۵، ۲۲، ۲۳، و ۲۵ هر کدام یک نمونه.

- برای خوشه ۸:

- دسته‌های ۱۰، ۲۵، ۳۱، و ۳۶ هر کدام یک نمونه.

- برای خوشه ۹:

- دسته‌های ۱۰، ۲۲، ۲۳، ۲۴، ۲۵، ۲۶، ۳۱، و ۳۲ هر کدام یک نمونه.

از تحلیل جدولی که ارائه شده است، می‌توانیم نتایج و الگوهایی را که بین خوشه‌ها و دسته‌ها وجود دارد، استخراج کنیم:

۱. الگوهای وجود دارنده:

- برخی از دسته‌ها به طور قابل توجهی در چندین خوشه حضور دارند. به عنوان مثال، دسته‌های ۳، ۶، ۷، ۱۳، ۱۴، ۱۵، ۲۲، ۲۳، ۲۴، ۲۵، ۲۶، ۲۷، ۲۸، ۳۰، ۳۱، ۳۲، و ۳۳ در بیش از یک خوشه قرار دارند. این نشان می‌دهد که این دسته‌ها دارای ویژگی‌هایی هستند که می‌توانند به صورت مشابه در خوشه‌های مختلف شناسایی شوند.

۲. تمرکز برخی دسته‌ها در یک یا دو خوشه:

- برخی دسته‌ها می‌توانند در یک یا دو خوشه به صورت گسترده تر شناسایی شوند. به عنوان مثال، دسته‌های ۱۱، ۱۵، ۲۲ و ۳۶ هر کدام فقط در یک خوشه قرار دارند، که نشان می‌دهد وجود یک ارتباط مستقیم بین ویژگی‌های این دسته‌ها و خوشه‌بندی انجام شده است.

۳. وجود خوشه‌های خاص برای برخی دسته‌ها:

- برخی دسته‌ها مانند دسته‌های ۶ و ۷ تنها در یک خوشه قرار دارند، که این می‌تواند به این معنی باشد که ویژگی‌های خاصی در این دسته‌ها وجود دارد که آنها را از دیگر دسته‌ها متمایز می‌کند.

۴. تنوع خوشه‌بندی:

- در برخی از خوشه‌ها، تنوع بیشتری در مورد دسته‌ها وجود دارد. به عنوان مثال، خوشه ۳ که دارای دسته‌های ۱ تا ۱۲ است، نشان می‌دهد که این خوشه برای یافتن یک الگوی خاص از ویژگی‌ها استفاده شده است.

این تحلیل نشان می‌دهد که الگوریتم خوشه‌بندی انتخابی، با توجه به نحوه‌ی طبقه‌بندی داده‌ها به دسته‌ها، قادر به تفکیک و شناسایی الگوهای مختلف داده‌ها بوده است. هرچه تعداد دسته‌هایی که در یک خوشه قرار گرفته‌اند بیشتر باشد، نشان از کیفیت خوب خوشه‌بندی و توانایی الگوریتم در تفکیک داده‌ها دارد.