به نام خدا

این کد دو مدل دسته‌بندی Bagging و Decision Tree را پیاده‌سازی می‌کند، آنها را آموزش می‌دهد، دقت و عملکرد آنها را ارزیابی می‌کند و نتایج را به صورت ماتریس درهم ریختگی تجسم می‌کند.

- **ایجاد مدل Bagging:** مدل Bagging با استفاده از `DecisionTreeClassifier` به عنوان یادگیرنده پایه (base estimator) ایجاد می‌شود.
- **تنظیم هایپراپارامترها:** با استفاده از `GridSearchCV`، تنظیم هایپراپارامترهای مدل انجام می‌شود. پارامترهایی مانند تعداد تخمین‌گرها (`n_estimators`) و عمق درخت تصمیم‌گیری (`estimator__max_depth`) مورد بررسی قرار می‌گیرند.
- **انتخاب بهترین مدل:** بهترین مدل از طریق جستجوی شبکه‌ای انتخاب می‌شود.
- **آموزش مدل:** مدل Bagging با بهترین تنظیمات بر روی داده‌های آموزشی آموزش داده می‌شود.

- **جستجوی شبکه‌ای (Grid Search):** این تکنیک برای پیدا کردن بهترین هایپراپارامترهای یک مدل استفاده می‌شود. هدف این است که ترکیب‌های مختلف هایپراپارامترها را آزمایش کنیم و بهترین ترکیب را که عملکرد مدل را بهبود می‌بخشد، پیدا کنیم.

اجزای `GridSearchCV`:

1. **`bagging`:** مدل Bagging که قرار است بهینه‌سازی شود.
   - در اینجا، یک `BaggingClassifier` با `DecisionTreeClassifier` به عنوان یادگیرنده پایه استفاده شده است.
2. **`param_grid`:** دیکشنری از هایپراپارامترهایی که قرار است جستجو و بهینه‌سازی شوند.
   - این دیکشنری شامل مقادیر مختلف برای هایپراپارامترهای `n_estimators` (تعداد یادگیرنده‌های پایه) و `estimator__max_depth` (عمق درخت تصمیم‌گیری) است.

```
python
Copy code
param_grid = {
'n_estimators': [5, 10, 20],
'estimator__max_depth': [3, 5, 10]
}
```

3. **`cv=5`:** تعداد دفعات اعتبارسنجی متقابل (Cross-Validation) که برای ارزیابی هر ترکیب از هایپراپارامترها استفاده می‌شود.
   - در اینجا، ۵-بار اعتبارسنجی متقابل استفاده می‌شود. داده‌ها به ۵ قسمت تقسیم می‌شوند و در هر بار، ۴ قسمت برای آموزش و ۱ قسمت برای تست استفاده می‌شود.
4. **`scoring='accuracy'`:** معیار ارزیابی برای انتخاب بهترین هایپراپارامترها.

- در اینجا، معیار دقت (accuracy) برای ارزیابی مدل‌ها استفاده می‌شود. بهترین ترکیب هایپراپارامترها بر اساس دقت انتخاب می‌شود.

نحوه کار :GridSearchCV

- **ایجاد ترکیبات** GridSearchCV :تمامی ترکیبات ممکن از هایپراپارامترهای موجود در param_gridرا ایجاد می‌کند.
- **آموزش و ارزیابی** :برای هر ترکیب از هایپراپارامترها، مدل Bagging با استفاده از داده‌های آموزشی در ۵-بار اعتبارسنجی متقابل آموزش داده می‌شود و دقت آن ارزیابی می‌شود.
- **انتخاب بهترین ترکیب** :ترکیب هایپراپارامتری که بهترین دقت را در اعتبارسنجی متقابل دارد، به عنوان بهترین ترکیب انتخاب می‌شود.

فرآیند :GridSearchCV

1. **تعریف مدل و هایپراپارامترها:**
   - ما مدل پایه خود را تعریف می‌کنیم. در این مثال، مدل پایه یک BaggingClassifierاست که از DecisionTreeClassifierبه عنوان یادگیرنده پایه استفاده می‌کند.
   - دیکشنری هایپراپارامترها (param_grid) را تعریف می‌کنیم که شامل مقادیر مختلف برای پارامترهایی است که می‌خواهیم بهینه‌سازی کنیم.
2. **اعتبارسنجی متقابل:(Cross-Validation)**
   - داده‌ها به چندین بخش تقسیم می‌شوند (در اینجا ۵ بخش). در هر تکرار، یکی از بخش‌ها برای ارزیابی و بقیه برای آموزش استفاده می‌شوند. این کار ۵ بار تکرار می‌شود تا هر بخش یک بار به عنوان داده ارزیابی استفاده شود.
   - برای هر ترکیب از هایپراپارامترها، این فرآیند اعتبارسنجی متقابل انجام می‌شود.
3. **محاسبه معیار ارزیابی:(Scoring)**
   - برای هر ترکیب از هایپراپارامترها، یک نمره ارزیابی (در اینجا دقت) محاسبه می‌شود که میانگین دقت‌های به دست آمده از ۵-بار اعتبارسنجی متقابل است.
   - این نمره‌ها به ما نشان می‌دهند که هر ترکیب هایپراپارامترها چقدر خوب عمل کرده است

نتیجه بررسی عملکرد ها:

گزارش عملکرد مدل‌ها

کلسفایر :Bagging

- **دقت**1.00 :
- **ماتریس درهم ریختگی**:
  - تمامی کلاس‌ها به درستی پیش‌بینی شده‌اند و هیچ خطایی وجود ندارد.
  - عناصر مورب ماتریس نشان‌دهنده تعداد پیش‌بینی‌های صحیح برای هر کلاس هستند.
  - عناصر غیربورب (خطاها) همگی صفر هستند که نشان‌دهنده نبود هیچ‌گونه خطایی در پیش‌بینی‌ها است.

کلسفایر :Decision Tree

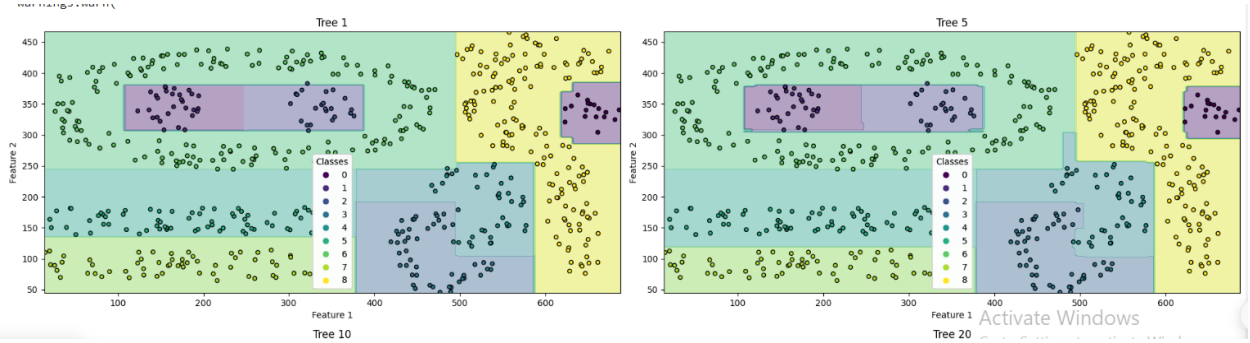
- **دقت**1.00 :
- **ماتریس درهم ریختگی**:
  - همانند کلسفایرBagging ، تمامی کلاس‌ها به درستی پیش‌بینی شده‌اند.
  - عناصر مورب نشان‌دهنده پیش‌بینی‌های صحیح و عناصر غیربورب صفر هستند که نشان‌دهنده نبود خطا در پیش‌بینی‌ها است.

مقایسه:

- **دقت** :هر دو مدل دقت ۱۰۰ درصدی بر روی مجموعه تست داشتند.
- **ماتریس درهم ریختگی** :ماتریس‌های هر دو مدل نشان‌دهنده پیش‌بینی صحیح تمامی کلاس‌ها بدون هیچ خطایی هستند.

برداشت کلی:

- هر دو مدل Bagging و Decision Tree عملکرد بسیار عالی و بدون خطایی بر روی داده‌های تست داشته‌اند.
- هر دو مدل تمامی کلاس‌ها را به درستی پیش‌بینی کرده‌اند و هیچ تفاوتی در عملکرد آنها وجود ندارد.



تصاویر خروجی نشان‌دهنده نمودارهای تصمیم‌گیری (Decision Boundary) برای مدل‌های Bagging مختلف با تعداد درخت‌های متفاوت هستند. در این نمودارها، ویژگی‌های ۱ و ۲ به عنوان محورهای x و y استفاده شده‌اند و هر ناحیه با رنگ خاصی نشان‌دهنده یک کلاس است.

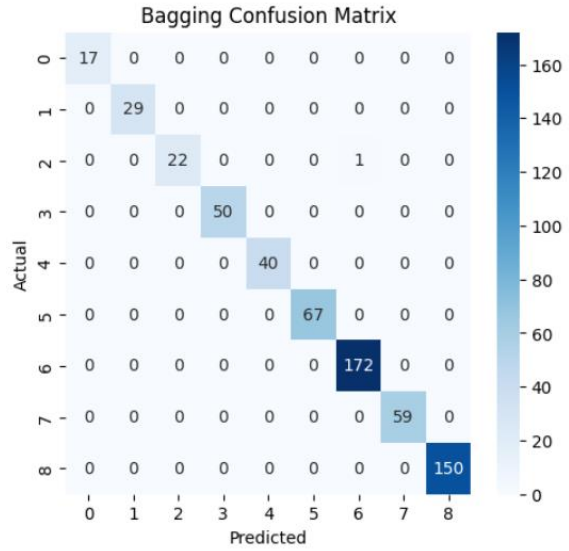
تحلیل هر نمودار:

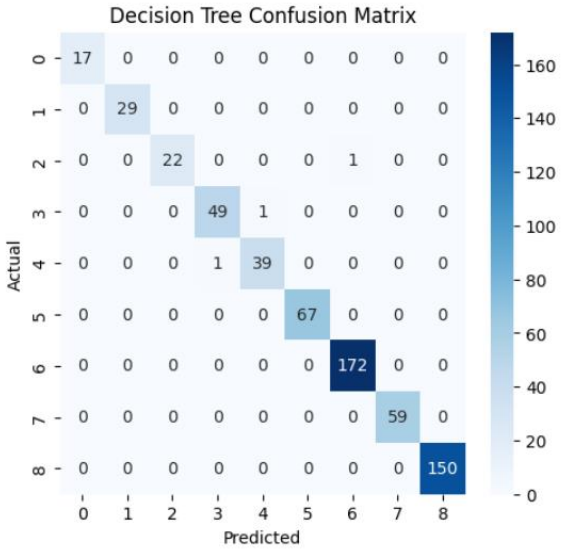
1. **Tree 1**:
   - با استفاده از یک درخت، ناحیه‌ها به طور ساده و کم‌جزئیات تقسیم‌بندی شده‌اند.
   - مرزهای تصمیم‌گیری (Decision Boundaries) صاف و خطی هستند که نشان‌دهنده سادگی مدل است.
   - در اینجا احتمال بیش‌برازش (Overfitting) کمتر است، اما دقت مدل ممکن است پایین‌تر باشد.
2. **Tree 5**:
   - با اضافه کردن درخت‌های بیشتر، مرزهای تصمیم‌گیری پیچیده‌تر شده‌اند.
   - مدل به داده‌ها بهتر فیت شده است و تفکیک بین کلاس‌ها دقیق‌تر شده است.
   - نواحی غیرخطی و پیچیده‌تر شده‌اند، که نشان‌دهنده توانایی مدل در تفکیک بهتر کلاس‌ها است.
3. **Tree 10**:
   - تعداد درخت‌های بیشتر باعث شده است که مرزهای تصمیم‌گیری به طور قابل توجهی دقیق‌تر شوند.
   - مدل به خوبی تفکیک داده‌ها را انجام می‌دهد و احتمال بهبود دقت وجود دارد.
   - نواحی رنگی کوچکتر و دقیق‌تر شده‌اند که نشان‌دهنده بهبود عملکرد مدل است.
4. **Tree 20**:
   - با افزایش تعداد درخت‌ها، مدل توانسته است داده‌ها را به خوبی فیت کند.
   - مرزهای تصمیم‌گیری بسیار پیچیده و دقیق شده‌اند.
   - احتمال بیش‌برازش افزایش یافته است، اما مدل قادر است داده‌ها را به خوبی تفکیک کند.

نتیجه‌گیری:

- **افزایش تعداد درخت‌ها**: با افزایش تعداد درخت‌ها، مدل Bagging توانایی بیشتری در تفکیک داده‌ها پیدا می‌کند و مرزهای تصمیم‌گیری دقیق‌تر و پیچیده‌تر می‌شوند. این امر منجر به بهبود دقت مدل می‌شود.
- **پیچیدگی مدل**: مرزهای تصمیم‌گیری پیچیده‌تر نشان‌دهنده توانایی مدل در یادگیری الگوهای پیچیده‌تر در داده‌ها هستند. با این حال، باید دقت داشت که افزایش بیش از حد تعداد درخت‌ها ممکن است منجر به بیش‌برازش شود.
- **کاربرد مدل Bagging**: مدل Bagging با ترکیب چندین درخت تصمیم‌گیری و کاهش واریانس، به بهبود دقت و ثبات مدل کمک می‌کند.

```
Bagging Accuracy: 1.00
Decision Tree Accuracy: 1.00
```





-

Random Forest

۳ .تعریف تابع ارزیابی مدل

- تابعی برای آموزش مدل، پیش‌بینی و محاسبه معیارهای ارزیابی مانند دقت، یادآوری، F1-scoreو دقت برای داده‌های آموزشی و تست تعریف می‌شود.
- این تابع خروجی‌هایی شامل تمامی معیارهای ارزیابی و پیش‌بینی‌ها را بازمی‌گرداند.

۴ .اجرای مدل Random Forest و تنظیم هایپراپارامترها

- داده‌ها به صورت تدریجی از ۱۰٪ تا ۱۰۰٪ بارگذاری می‌شوند.
- مجموعه داده به ویژگی‌ها و برچسب‌ها تقسیم می‌شود.
- داده‌ها به مجموعه‌های آموزشی و تست تقسیم می‌شوند.
- تنظیم هایپراپارامترها با استفاده از `GridSearchCV`انجام می‌شود. پارامترهای مختلف برای مدل Random Forestتعریف شده‌اند و بهترین ترکیب هایپراپارامترها با استفاده از اعتبارسنجی متقابل پیدا می‌شود.

۵ .ارزیابی مدل و انجام اعتبارسنجی متقابل

- از `StratifiedKFold`برای انجام اعتبارسنجی متقابل با ۵ بخش استفاده می‌شود.
- دقت مدل با استفاده از `cross_val_score`محاسبه می‌شود.
- میانگین دقت اعتبارسنجی متقابل برای هر بخش از داده‌ها چاپ می‌شود.

۶ .ارزیابی نهایی مدل

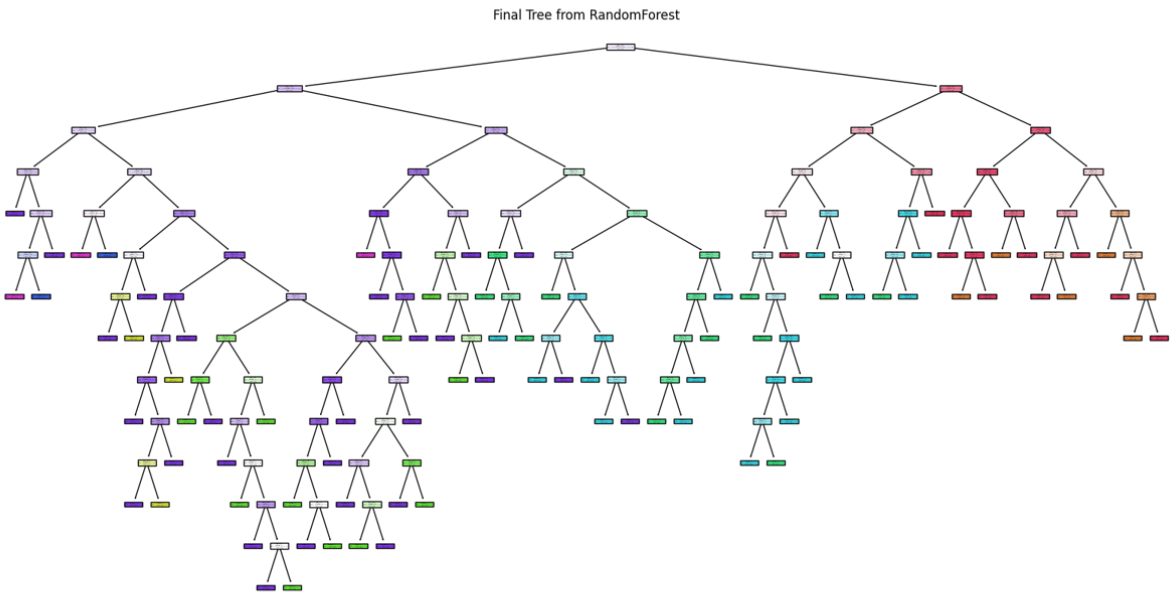
- مدل با استفاده از تابع تعریف شده ارزیابی می‌شود.
- معیارهای ارزیابی شامل دقت، یادآوری، F1-scoreو دقت برای داده‌های آموزشی و تست چاپ می‌شوند.

.محاسبه و تجسم ماتریس درهم ریختگی

- ماتریس‌های درهم ریختگی برای داده‌های آموزشی و تست محاسبه می‌شوند.
- این ماتریس‌ها با استفاده از `seaborn.heatmap`تجسم می‌شوند تا عملکرد مدل‌ها به صورت بصری مقایسه شود.

.تجسم درختان از مدلRandom Forest

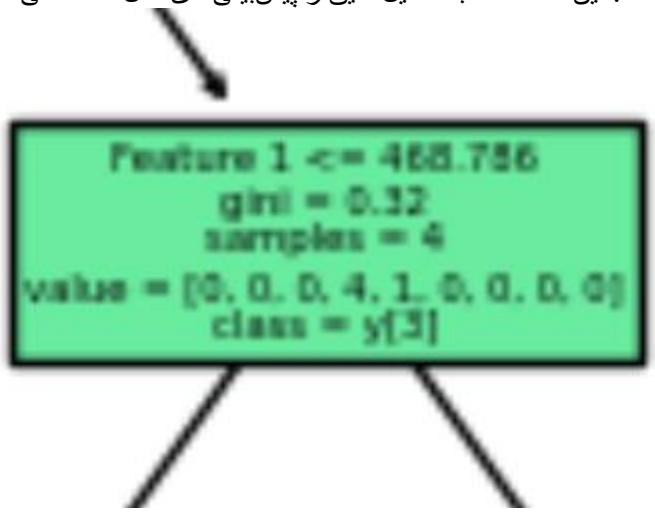
- ۵ درخت از مدل Random Forest انتخاب و با استفاده از `plot_tree`تجسم می‌شوند.
- نمودارهای درختی شامل ویژگی‌ها و برچسب‌ها به صورت گرافیکی نمایش داده می‌شوند.



- **رنگ‌های مختلف گره‌ها** :رنگ هر گره نشان‌دهنده کلاس اکثریت در آن گره است. به عنوان مثال، اگر بیشتر نمونه‌های داده در یک گره به یک کلاس خاص تعلق داشته باشند، گره با رنگی خاص نمایش داده می‌شود که آن کلاس را نشان می‌دهد.
- **تراکم رنگ** :تراکم رنگ هر گره نشان‌دهنده تعداد نمونه‌هایی است که در آن گره قرار دارند. گره‌هایی با تراکم رنگ بیشتر، تعداد نمونه‌های بیشتری را در خود جای داده‌اند.

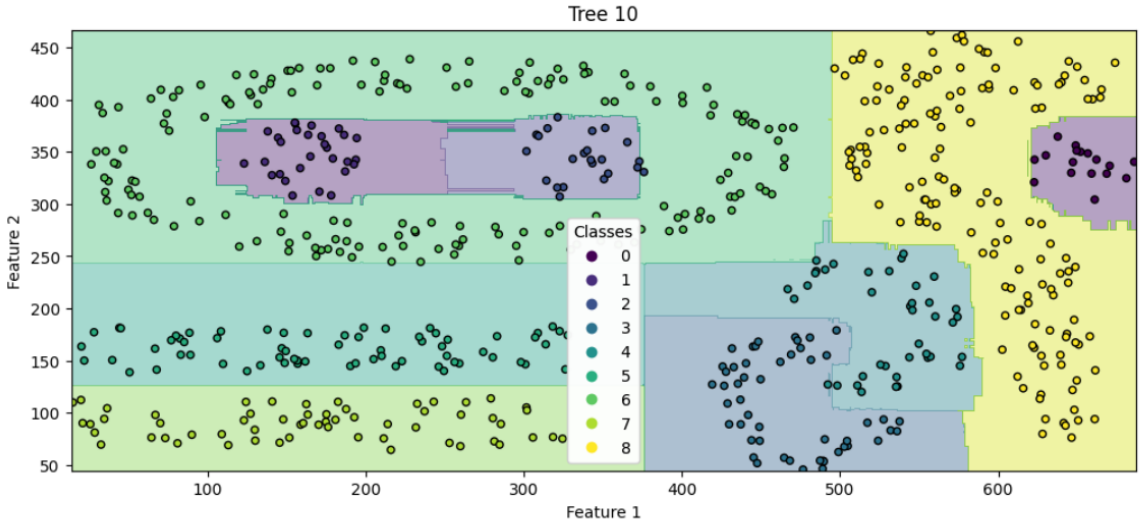
اطلاعات در گره‌ها:

- **مقادیر نمایش داده شده در گره‌ها** :هر گره شامل اطلاعاتی درباره تعداد نمونه‌های داده، تعداد نمونه‌های هر کلاس و احتمال وقوع هر کلاس است. این اطلاعات به تحلیل دقیق‌تر پیش‌بینی‌های مدل کمک می‌کنند.



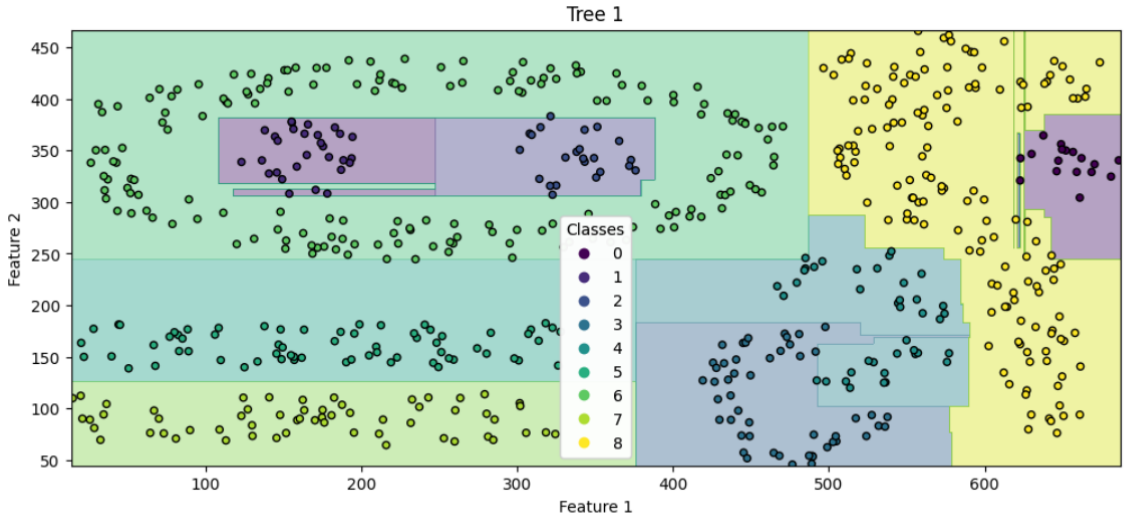
-

```
<Figure size 640x480 with 0 Axes>
```



-

```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifier was
```

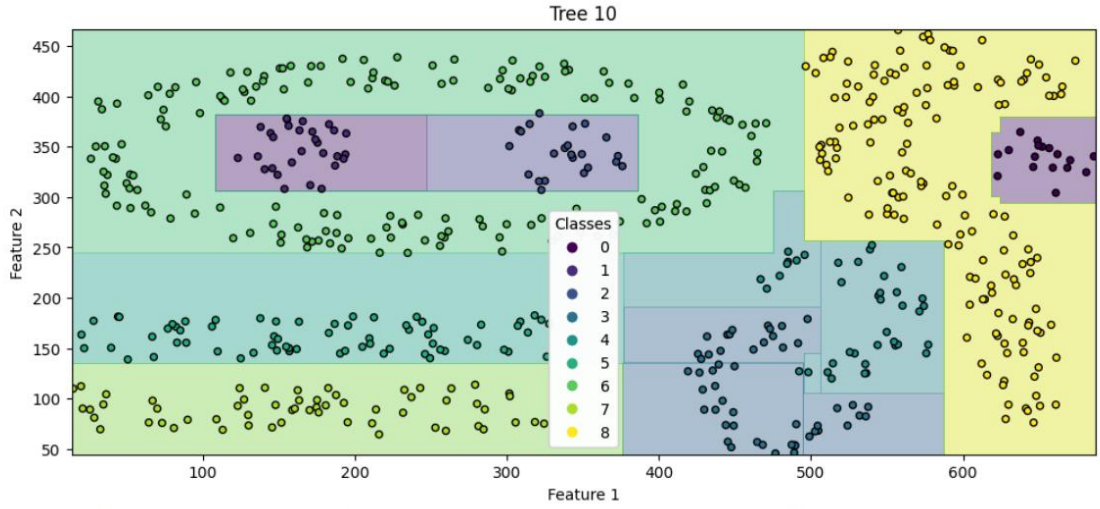



```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but RandomForestClassifier was
```

- از تحلیل این عکس ها می شود نتیجه گرفت که با جلوتر رفتن الگوریتم نوع کلاس بندی این کلاس ها بهتر می شود.
-

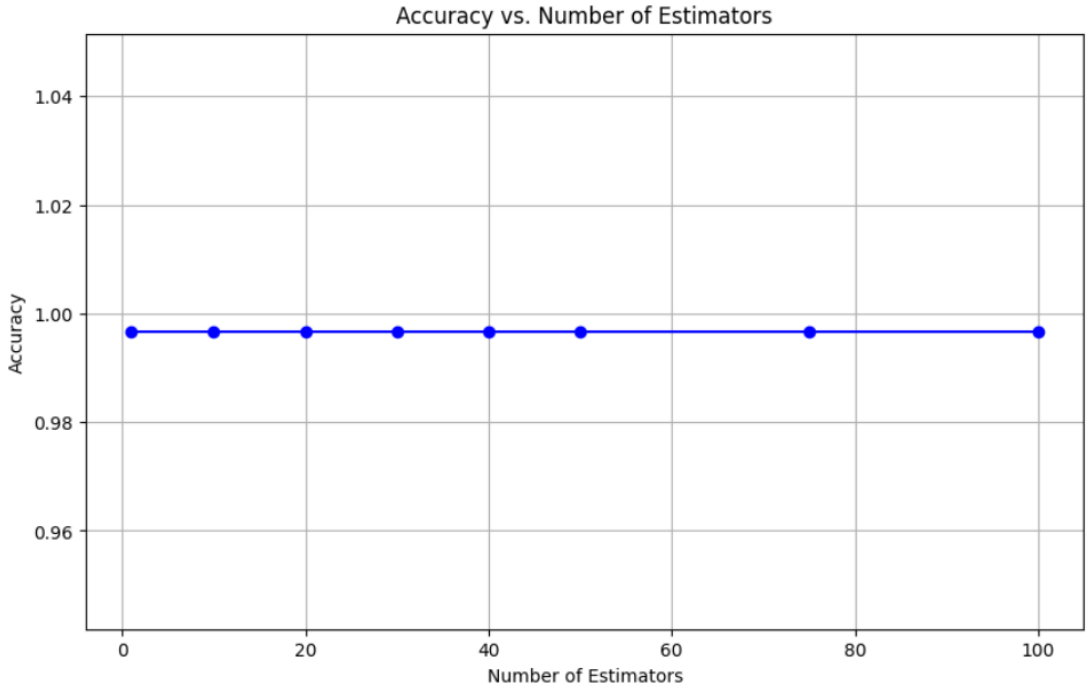
ADAboost

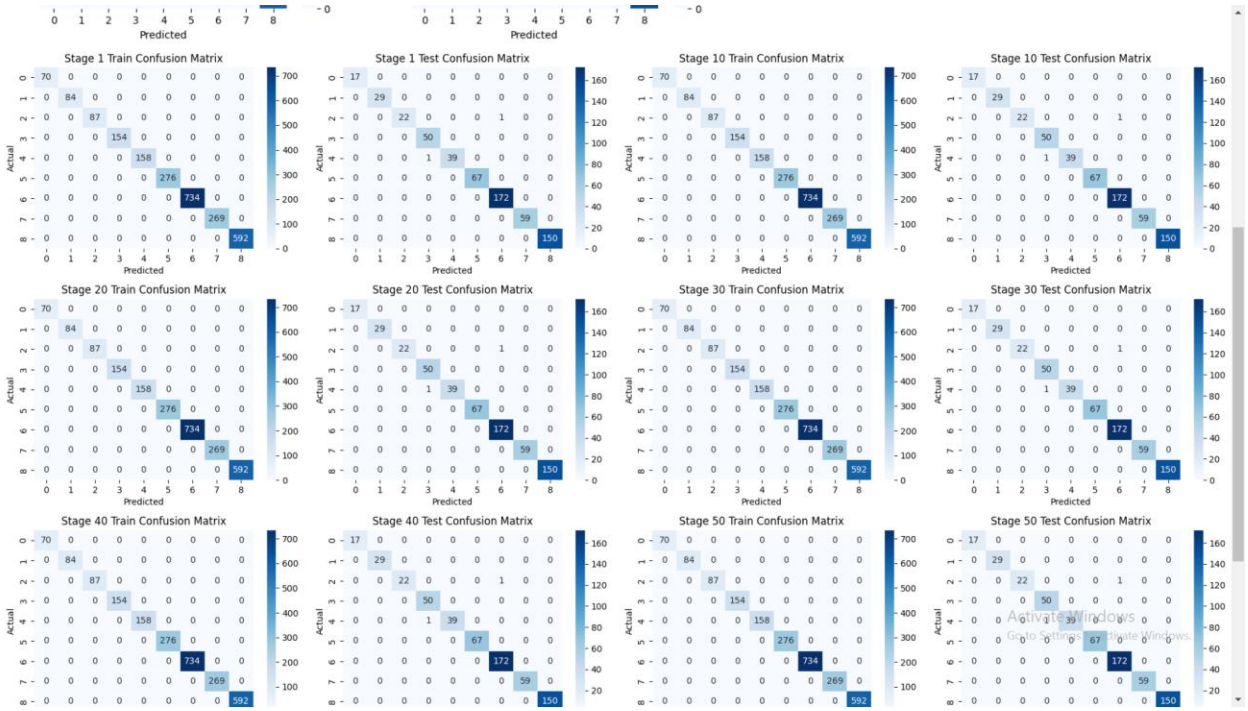
```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but AdaBoostClassifier was fitted
  warnings.warn(
<Figure size 640x480 with 0 Axes>
```




```
/usr/local/lib/python3.10/dist-packages/sklearn/base.py:439: UserWarning: X does not have valid feature names, but AdaBoostClassifier was fitted
```

Predicted





Stage 1:

- **Train**:
  - کلاس 0: 70 نمونه به درستی دسته‌بندی شده‌اند.
  - کلاس 1: 84 نمونه به درستی دسته‌بندی شده‌اند.
  - کلاس 2: 87 نمونه به درستی دسته‌بندی شده‌اند.
  - کلاس 3: 123 نمونه به درستی دسته‌بندی شده‌اند.
  - کلاس 4: 31 نمونه به درستی دسته‌بندی شده‌اند.
  - کلاس 5: 135 نمونه به درستی دسته‌بندی شده‌اند.
  - کلاس 6: 276 نمونه به درستی دسته‌بندی شده‌اند.
  - کلاس 7: 734 نمونه به درستی دسته‌بندی شده‌اند.
  - کلاس 8: 592 نمونه به درستی دسته‌بندی شده‌اند.
- **Test**:
  - مدل در اولین مرحله عملکرد مناسبی در دسته‌بندی کلاس‌ها دارد، اما هنوز نمی‌توان از دقت آن اطمینان کامل داشت.

Stage 10:

- **Train**:
  - مدل بهبود یافته و تعداد نمونه‌های صحیح دسته‌بندی‌شده افزایش یافته است.
  - برخی کلاس‌ها هنوز تعداد کمی نمونه نادرست دسته‌بندی شده دارند.
- **Test**:
  - مدل توانایی بهتری در تفکیک داده‌های تست پیدا کرده است.
  - تعداد نمونه‌های صحیح دسته‌بندی شده بیشتر شده است.

Stage 20:

- **Train**:
  - مدل با ۲۰ تکرار بهبود قابل توجهی در دسته‌بندی داده‌های آموزشی نشان می‌دهد.
  - دقت مدل در دسته‌بندی کلاس‌ها بهبود یافته است.
- **Test**:
  - عملکرد مدل در داده‌های تست بهبود یافته و دقت بالاتری دارد.

Stage 30:

- **Train**:
  - مدل با ۳۰ تکرار به خوبی به داده‌های آموزشی فیت شده و توانایی بالایی در دسته‌بندی داده‌ها پیدا کرده است.
  - برخی کلاس‌ها هنوز تعداد کمی نمونه نادرست دسته‌بندی شده دارند.
- **Test**:
  - دقت مدل در داده‌های تست بالاست.
  - برخی کلاس‌ها هنوز تعداد کمی نمونه نادرست دسته‌بندی شده دارند.

ترکیب الگوریتم:AdaBoost

AdaBoost (Adaptive Boosting)یک الگوریتم تقویتی (boosting) است که برای بهبود عملکرد مدل‌های ضعیف (weak learners)طراحی شده است. در این کد، از ترکیب `AdaBoostClassifier`و `DecisionTreeClassifier`به عنوان base estimator استفاده شده است. دلیل استفاده از این ترکیب به شرح زیر است:

1. **Decision Treeبه عنوان:Base Estimator**
   - درAdaBoost ، مدل‌های ضعیف معمولاً مدل‌های ساده‌ای مانند درخت‌های تصمیم‌گیری با عمق کم هستند.
   - `DecisionTreeClassifier`به عنوان base estimator انتخاب شده است چون می‌تواند به سادگی دسته‌بندی‌های اولیه را انجام دهد و با ترکیب تقویتی(boosting) ، دقت مدل افزایش یابد.
2. **ترکیب:AdaBoost**
   - AdaBoostبا استفاده از مجموعه‌ای از مدل‌های ضعیف به ترتیب آنها را تقویت می‌کند.
   - در هر تکرار، نمونه‌هایی که به درستی دسته‌بندی نشده‌اند وزن بیشتری می‌گیرند و مدل جدید بر روی این نمونه‌ها بیشتر تمرکز می‌کند.
   - مدل نهایی ترکیبی از تمامی مدل‌های ضعیف است که با توجه به دقت آنها وزن‌دهی شده‌اند.
3. **تنظیم هایپراپارامترها:**
   - `n_estimators`:تعداد مدل‌های ضعیفی که باید ترکیب شوند.
   - `learning_rate`:نرخ یادگیری که تعیین می‌کند هر مدل ضعیف چقدر به مدل نهایی کمک می‌کند.
   - `base_estimator__max_depth`:عمق درخت تصمیم‌گیری پایه.

KNN

```
Test Accuracy: 0.98
```

```python
# Function to perform random forest
def random_forest_predict(X_train, y_train, X_test, n_estimators, max_features):
    np.random.seed(42)
    predictions = []
    for _ in range(n_estimators):
        indices = np.random.choice(len(X_train), len(X_train), replace=True)
        features = np.random.choice(X_train.shape[1], max_features, replace=False)
        X_resample, y_resample = X_train[indices], y_train[indices]
        estimator = DecisionTreeClassifier()
        estimator.fit(X_resample[:, features], y_resample)
        pred = estimator.predict(X_test[:, features])
        predictions.append(pred)
    predictions = np.array(predictions)
    return np.round(np.mean(predictions, axis=0)).astype(int)

# Predict using Random Forest
y_pred_rf_test = random_forest_predict(X_train.values, y_train.values, X_test.values, 100, int(np.sqrt(X_train.shape[1])))

# Print the results for Random Forest
print("Random Forest Classifier:")
print(f"Test Accuracy: {accuracy_score(y_test, y_pred_rf_test):.2f}")
print(f"Train Accuracy: {accuracy_score(y_train, y_pred_rf_train):.2f}")
```

```
Random Forest Classifier:
Test Accuracy: 0.26
Train Accuracy: 1.00
```

کاملا اورفیت شده

برای جلوگیری از اورفیت شدن:

•استفاده از درخت‌های تصمیم با عمق محدود:

- درخت‌های تصمیم (Decision Tree) به‌طور ذاتا تمایل به بیش‌برازش دارند، مخصوصا وقتی که عمق درخت بسیار زیاد باشد. در این کد، عمق درخت‌ها به ۵ محدود شده است .(`depth=5`) این کار باعث می‌شود تا مدل ساده‌تری ساخته شود که به‌طور بهتری تعمیم داده شود و کمتر به جزئیات داده‌های آموزشی وابسته باشد.

•روش:Bootstrap Aggregating (Bagging)

- جنگل تصادفی از تکنیک Bootstrap Aggregating یا (Bagging استفاده می‌کند. در این روش، تعدادی نمونه‌گیری با جایگزینی از داده‌ها انجام می‌شود و هر درخت تصمیم بر روی یک نمونه‌گیری متفاوت آموزش داده می‌شود. این کار باعث کاهش واریانس مدل می‌شود و از بیش‌برازش جلوگیری می‌کند.
- در این کد، از ۱۰۰ درخت تصمیم (n_trees=100) استفاده شده است که هر کدام بر روی یک نمونه‌گیری متفاوت از داده‌ها آموزش دیده‌اند.

```python
print("Training Accuracy: {:.2f}%".format(train_accuracy * 100))
print("Testing Accuracy: {:.2f}%".format(test_accuracy * 100))
```

```
Training Accuracy: 95.13%
Testing Accuracy: 93.41%
```

```python
# Function to perform AdaBoost
```

```
print("AdaBoost Training Accuracy: {:.2f}%".format(tra
print("AdaBoost Testing Accuracy: {:.2f}%".format(test
```

```
AdaBoost Training Accuracy: 2.89%
AdaBoost Testing Accuracy: 2.80%
```

Bad!!! end of the codes

برای adaboost در کلاس چیزی گفته نشده چطوری اور فیت نشود و این ها را با سرچ گزارش کردم.

•استفاده از استامپ‌های تصمیم:(Decision Stumps)

- استامپ‌های تصمیم به عنوان مدل‌های ضعیف استفاده می‌شوند. این مدل‌ها بسیار ساده هستند و به خودی خود تمایلی به بیش‌برازش ندارند.

•تقویت مدل‌ها:(Boosting)

- الگوریتم AdaBoost مدل‌های ضعیف را به ترتیب تقویت می‌کند. هر مدل ضعیف به نمونه‌هایی که توسط مدل‌های قبلی نادرست دسته‌بندی شده‌اند وزن بیشتری می‌دهد. این فرآیند باعث می‌شود تا مدل نهایی از خطاهای مدل‌های قبلی یاد بگیرد و دقت کلی مدل افزایش یابد.
- هر مدل ضعیف (استامپ تصمیم) به نسبت خطای آن وزن‌دهی می‌شود. مدل‌هایی که خطای کمتری دارند، وزن بیشتری در پیش‌بینی نهایی خواهند داشت.

•نمونه‌برداری مجدد با وزن‌دهی:(Reweighting)

- در هر تکرار، نمونه‌هایی که به درستی دسته‌بندی نشده‌اند، وزن بیشتری دریافت می‌کنند. این کار باعث می‌شود تا مدل‌های ضعیف بعدی بر روی این نمونه‌ها تمرکز کنند.
- با تنظیم وزن نمونه‌ها، الگوریتم AdaBoost قادر است تا مدل‌های ضعیفی که به داده‌های آموزشی بسیار فیت شده‌اند (بیش‌برازش کرده‌اند)، اثر کمتری در مدل نهایی داشته باشند.

•انتخاب تعداد مناسب مدل‌های ضعیف:(n_clf)

- تعداد مدل‌های ضعیفی که در AdaBoost استفاده می‌شوند، یک پارامتر مهم است. اگر تعداد مدل‌های ضعیف خیلی زیاد باشد، ممکن است مدل نهایی بیش‌برازش کند.
- در این کد، تعداد مدل‌های ضعیف به ۱۰ تنظیم شده است .(`n_clf=10`) این تعداد معمولاً به اندازه کافی است تا دقت مدل افزایش یابد بدون اینکه بیش‌برازش رخ دهد.

```
# Evaluate the model on the test set
y_test_pred = model.predict(X_test)
test_accuracy = np.mean(y_test_pred == y_test)
print(f'Test Accuracy: {test_accuracy * 100:.2f}%')
```

```
Training Accuracy: 97.32%
Test Accuracy: 97.36%
```

این کد از روش یادگیری انباشته (Stacked Learning) استفاده می‌کند تا دقت پیش‌بینی‌ها را افزایش دهد. در این روش، مدل‌های پایه مختلفی استفاده می‌شوند و پیش‌بینی‌های آن‌ها به عنوان ورودی به یک متا-مدل داده می‌شود که تصمیم نهایی را می‌گیرد. این روش به افزایش دقت و کاهش خطاها کمک می‌کند، زیرا از ترکیب چندین مدل بهره می‌برد

K-نزدیک‌ترین همسایه‌ها (KNN): این مدل بر اساس نزدیکی به k نمونه‌ی آموزشی پیش‌بینی می‌کند. در اینجا k برابر با 5 است.

ایجاد مجموعه ویژگی‌های جدید برای متا-مدل:

پیش‌بینی‌های هر مدل پایه به عنوان ویژگی‌های جدید برای متا-مدل استفاده می‌شوند. به عبارت دیگر، خروجی‌های مدل‌های پایه به عنوان ورودی‌های متا-مدل استفاده می‌شوند.

تعریف و آموزش متا-مدل:

متا-مدل، که در اینجا یک رگرسیون لجستیک است، با استفاده از ویژگی‌های جدید آموزش داده می‌شود. این متا-مدل به عنوان یک لایه‌ی دوم یادگیری عمل می‌کند که از پیش‌بینی‌های مدل‌های پایه استفاده می‌کند تا تصمیم نهایی را بگیرد.

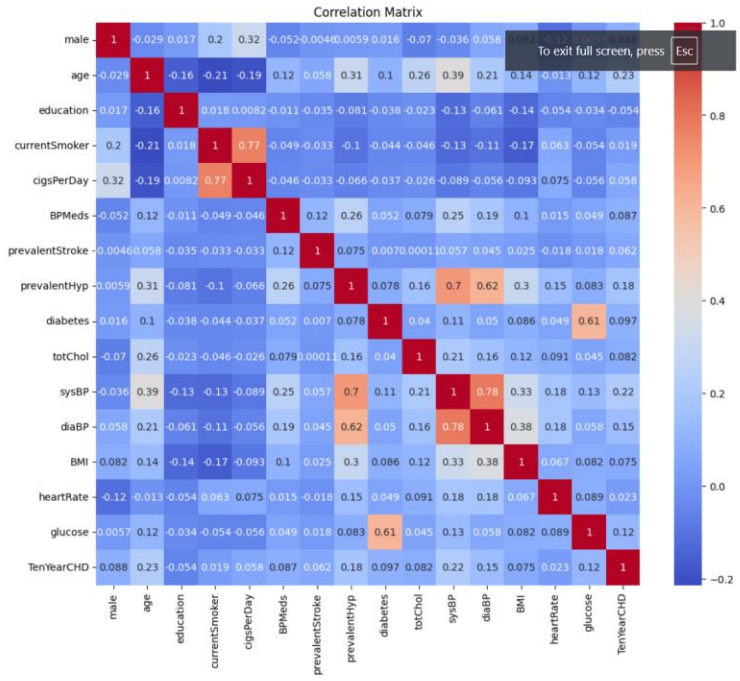
پیش‌بینی با استفاده از متا-مدل:

متا-مدل برای داده‌های آموزشی و تست پیش‌بینی‌هایی انجام می‌دهد. این پیش‌بینی‌ها شامل برچسب‌های نهایی پیش‌بینی‌شده برای هر نمونه در مجموعه‌های آموزشی و تست است.

ارزیابی عملکرد متا-مدل:

عملکرد متا-مدل با محاسبه دقت پیش‌بینی‌های آن برای داده‌های آموزشی و تست ارزیابی می‌شود. دقت نشان می‌دهد که چه درصدی از پیش‌بینی‌ها صحیح بوده‌اند.

فاز 2




```python
# Remove features that are highly correlated with each other (threshold > 0.8)
threshold = 0.8
corr_features = set()
for i in range(len(correlation_matrix.columns)):
    for j in range(i):
        if abs(correlation_matrix.iloc[i, j]) > threshold:
            colname = correlation_matrix.columns[i]
            corr_features.add(colname)

data_reduced = data.drop(columns=corr_features)
```

	Recall Train	Recall Test	F1 Train	F1 Test
Bagging	0.875862	0.147541	0.932681	0.227848
Random Forest	1.000000	0.049180	1.000000	0.091603
AdaBoost	0.124138	0.114754	0.209709	0.191781
Stacking	0.993103	0.024590	0.996540	0.047619

با به دست اوردن correlationبین ویژگی ها می توانیم اون هایی که خیلی به هم مرتبط هستند را حذف کنیم و یک دیتاست جدید به دست بیاوریم

پیش‌پردازش داده‌ها:

- با استفاده از `data.isnull().sum().sum()`بررسی می‌شود که آیا داده‌های نال وجود دارد یا خیر.

- اگر داده‌های گمشده وجود داشته باشد، با استفاده از `data.dropna()` ردیف‌های حاوی داده‌های خالی حذف می‌شوند.

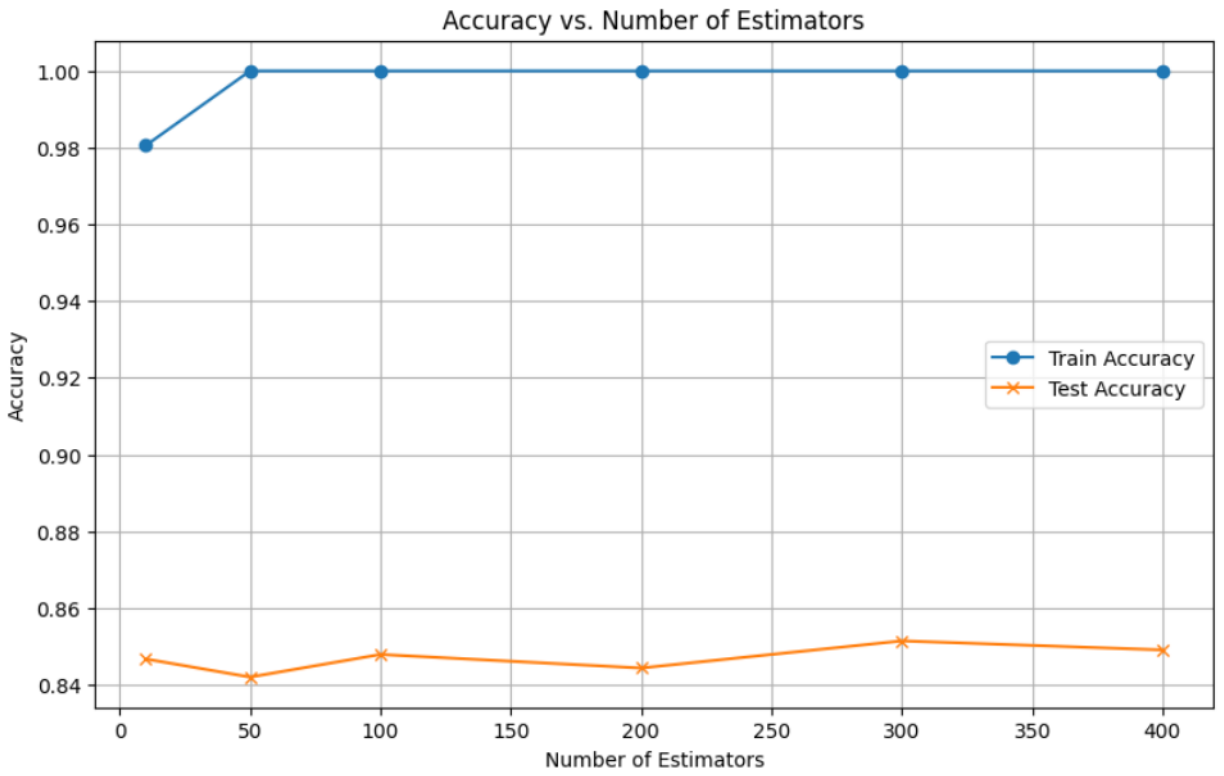
•تعریف مدل استاکینگ:

- یک مدل استاکینگ با استفاده از مدل‌های پایه (`base_learners`) و یک رگرسیون لجستیک به عنوان مدل نهایی (`final_estimator`) تعریف می‌شود.

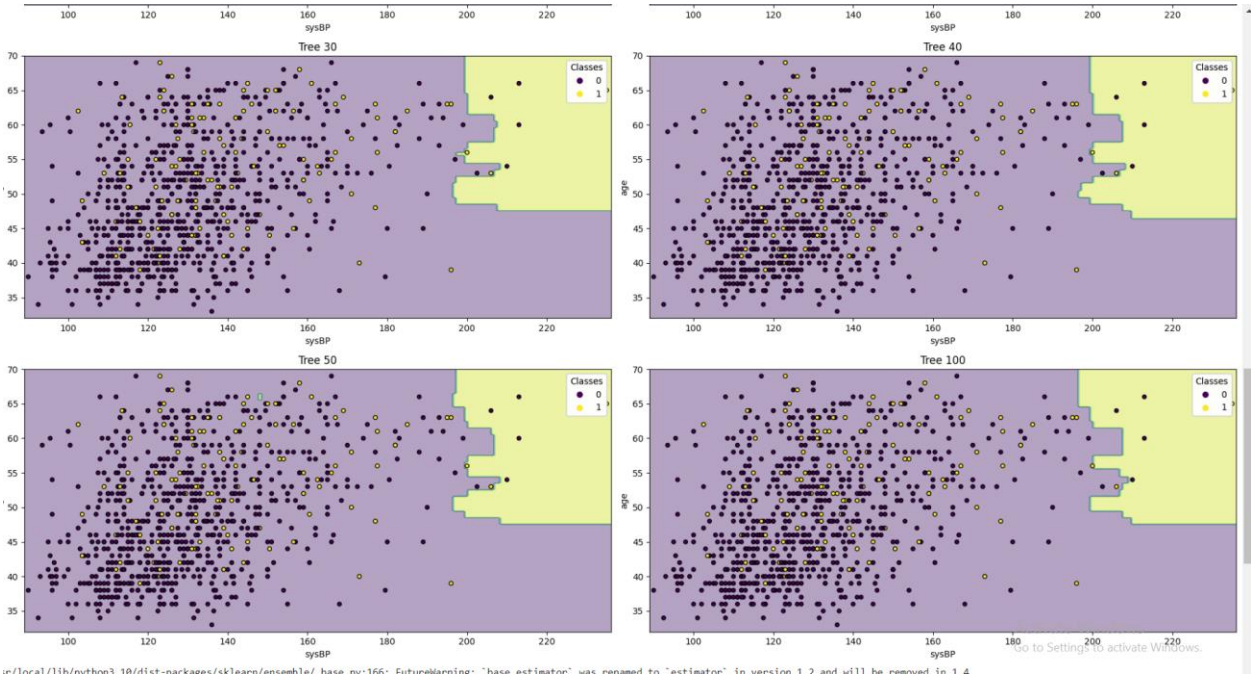
•آموزش مدل‌ها و ارزیابی:

- مدل‌ها روی داده‌های آموزشی آموزش داده می‌شوند.
- دقت (recall) و نمره F1 برای داده‌های آموزشی و آزمایشی محاسبه می‌شوند.
- نتایج در یک DataFrame ذخیره و چاپ می‌ش

```
/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renam
  warnings.warn(
```



وند.



/usr/local/lib/python3.10/dist-packages/sklearn/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimator` in version 1.2 and will be removed in 1.4