به نام خدا

دانشکده مهندسی کامپیوتر

تمرینات درس داده‌کاوی

تمرین سری چهارم

طبقه‌بندی-۱ (درخت تصمیم)

دکتر حسین رحمانی

دانشجو:

غزاله بختیاری آزاد

۹۵۵۲۱۰۶۳

فهرست مطالب

۱. پیش‌پردازش داده

۱,۱. وارد کردن داده

- داده‌ها را از فایل مورد نظر بخوانید و در dataframe ذخیره کنید:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
df = pd.read_csv('https://archive.ics.uci.edu/ml/machine-learning-
databases/heart-disease/processed.cleveland.data', header=None)
```

- نام ستون‌ها را به صورت زیر وارد کنید:

```
Columns = ['age', 'sex', 'cp', 'restbp', 'chol', 'fbs', 'restecg', 'thalach',
 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'hd']
```

  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
df.columns = ['age', 'sex', 'cp', 'restbp', 'chol', 'fbs', 'restecg', 'thal
ach', 'exang', 'oldpeak', 'slope', 'ca', 'thal', 'hd']
```

- پیش‌پردازش‌های لازم را روی داده‌ها انجام دهید:
  - راه حل: برای پیش‌پردازش داده‌ها در ابتدا Missing-Valueها را تشخیص می‌دهیم چرا که scikit-learn داده‌های miss شده را نمی‌تواند پردازش کند و در آینده به مشکل برخواهیم خورد. پس به ترتیب مراحل زیر را طی می‌کنیم:
    ۱. با استفاده از دستور زیر نوع هر ستون را می‌یابیم:

```
print(df.dtypes)
```

که خروجی آن برابر است با:

```
age         float64
sex         float64
cp          float64
restbp      float64
chol        float64
fbs         float64
restecg     float64
thalach     float64
exang       float64
oldpeak     float64
slope       float64
ca           object
thal         object
hd            int64
```

۲. برای ستون‌های ca و thal که از نوع object هستند، قطعه کد زیر را اجرا می‌کنیم تا ببینیم این دو ستون چه مقدارهایی را شامل می‌شوند:

```
print(df['ca'].unique())
print(df['thal'].unique())
```

که خروجی آن برابر است با:

```
['0.0' '3.0' '2.0' '?']
['6.0' '3.0' '7.0' '?']
```

همانطور که مشاهده می‌شود در این ستون‌ها مقدار ؟ دیده می‌شود که یعنی در این ستون‌ها مقدارهایی miss شده‌اند.

۳. با استفاده از قطعه کد زیر مشاهده می‌کنیم که چه تعداد از سطرهای ما شامل missing value در یکی از این دو ستون هستند:

```
print(df.loc[(df['ca'] == '?') | (df['thal'] == '?')])
```

که با توجه به خروجی کد ۶ تا از سطرها دارای missing value در یکی از این دو ستون هستند.

۴. حال با استفاده از قطعه کد زیر متوجه می‌شویم که در کل ۳۰۳ سطر داریم که اگر این ۶ سطر را کلا حذف کنیم ۲۹۷ سطر خواهیم داشت که missing-value ندارند:

```
print(len(df_no_missing))
df_no_missing_value = df.loc[(df['ca'] != '?') & (df['thal'] != '?')]
```

یعنی دیتافریم df_no_missing_value داده پیش‌پردازش شده ما است که هیچ missing valueای ندارد.

- ستون مشخص کننده بیماری قلبی (ستون 'hd') را به عنوان ستونی انتخاب کنید که می‌خواهید پیش‌بینی کنید:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
X = df_no_missing_value.drop('hd', axis=1).copy()
y = df_no_missing_value['hd'].copy()
```

- سایر ستون‌ها را به عنوان ویژگی‌هایی انتخاب کنید که برای انجام پیش‌بینی استفاده می‌شود:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
y = df_no_missing_value['hd'].copy()
```

۱,۲. رمزگذاری One-Hot

- با استفاده از تابع "get_dummies()" ستون‌های غیربایتری را رمزگذاری One-Hot کنید:
  - راه حل: گفتیم که با استفاده از dtypes نوع داده‌های هر ستون را می‌توانیم مشاهده کنیم. پس از استفاده از این تابع دیدیم که ستون‌های age، restbp، chol و thalach همگی float64 بودند که منطقی بوده و درست است و به این ستون‌ها دست نمی‌زنیم. اما سایر ستون‌ها که باید categorical باشند اما به دلیل اینکه scikit-learn نمی‌تواند داده‌های categorical را پردازش کند این ستون‌ها را با استفاده از One-Hot به چندین ستون باینری تبدیل می‌کنیم که کد آن به صورت زیر است:

```
X_encoded = pd.get_dummies(X, columns=['cp', 'restecg', 'slope', 'thal'])
```

اکنون به سراغ ستون hd می‌رویم و مشاهده می‌کنیم که داده‌های موجود در آن فقط ۰ و ۱ نیستند بلکه دارای ۵ مقدار از ۰ تا ۴ هستند که ۰ نداشتن مشکلات قلبی و ۱-۴ درجه مختلف مشکلات قلبی را نشان می‌دهند. از آنجا که هدف ما این است که صرفا 'داشتن' یا 'نداشتن' مشکلات قلبی را پیش‌بینی کنیم، با استفاده از قطعه کد زیر ۰ را به نداشتن مشکلات قلبی و ۱-۴ را به داشتن مشکلات قلبی (یعنی مقدار ۱) map می‌کنیم:

```
y_not_zero_index = y > 0
y[y_not_zero_index] = 1
```

۱,۳. ساخت مدل درخت تصمیم

در ادامه به ساخت مدل درخت تصمیم می‌پردازیم.

۱,۳,۱. مقیاس‌بندی داده

- داده‌های تست و آموزش را جدا کنید (random_state=42)

- راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
X_train, X_test, y_train, y_test = train_test_split(X_encoded, y, random_state=42)
```

۱,۳,۲. ساخت نمونه اولیه

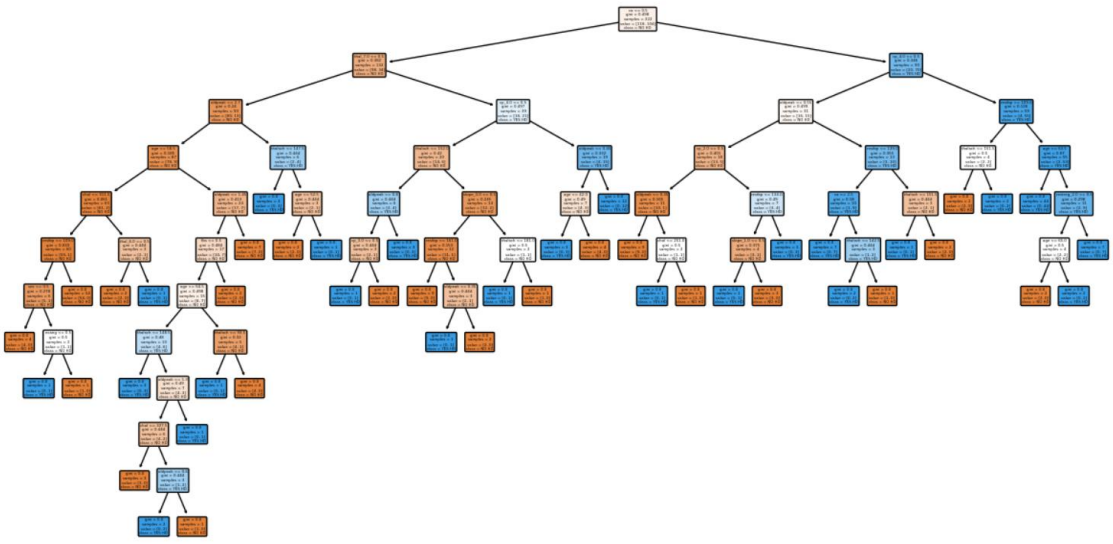
- یک نمونه اولیه از درخت تصمیم با استفاده از داده‌های آموزشی و تست بسازید:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
clf_dt = DecisionTreeClassifier(random_state=42)
clf_dt = clf_dt.fit(X_train, y_train)
```

- ساختار درختی آن را رسم کنید:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
plt.figure(figsize=(15, 7.5))
plot_tree(clf_dt, filled=True, rounded=True, class_names=["NO HD", "YES HD"], feat
ure_names=X_encoded.columns)
plt.show()
```

که نمودار درختی آن به شکل زیر نمایش داده می‌شود:

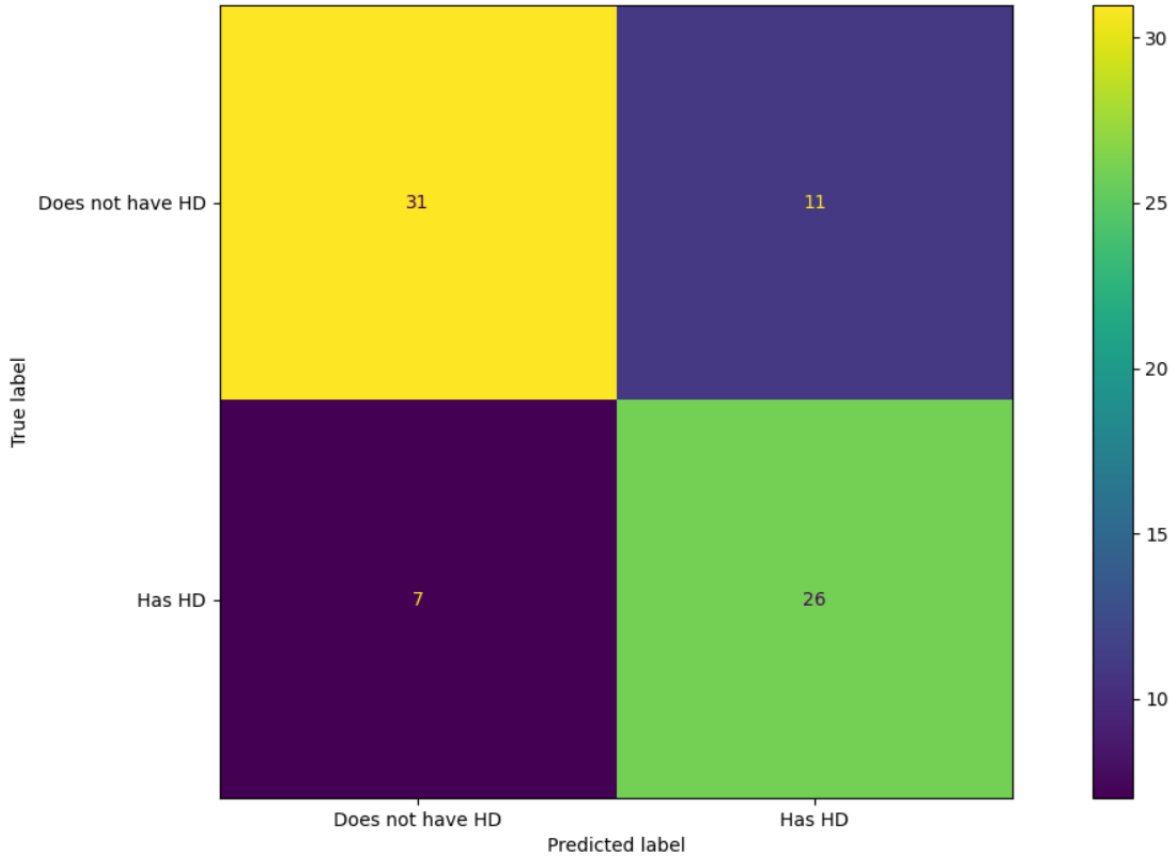


- با استفاده از ماتریس در هم ریختگی (Confusion Matrix) و داده‌های تست عملکرد این درخت تصمیم را ارزیابی کنید:

- راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
plot_confusion_matrix(clf_dt, X_test, y_test, display_labels=["Does not have HD", "Has HD"])
```

و ارزیابی آن به شکل زیر است که نشان می‌دهد از ۴۲ نفری که بیماری قلبی نداشته‌اند، ۳۱ نفر آن‌ها (۷۴٪) درست تشخیص داده شده و از ۳۳ نفری که بیماری قلبی داشته‌اند، ۲۶ نفر آن‌ها (۷۹٪) درست تشخیص داده شده‌اند و این نشان می‌دهد که مدل ما به به داده آموزشی overfit شده‌است و می‌توان آن را بهتر کرد.



۱،۳،۳. هرس درخت تصمیم

- درخت تصمیم را با استفاده از Cost Complexity pruning هرس کنید:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

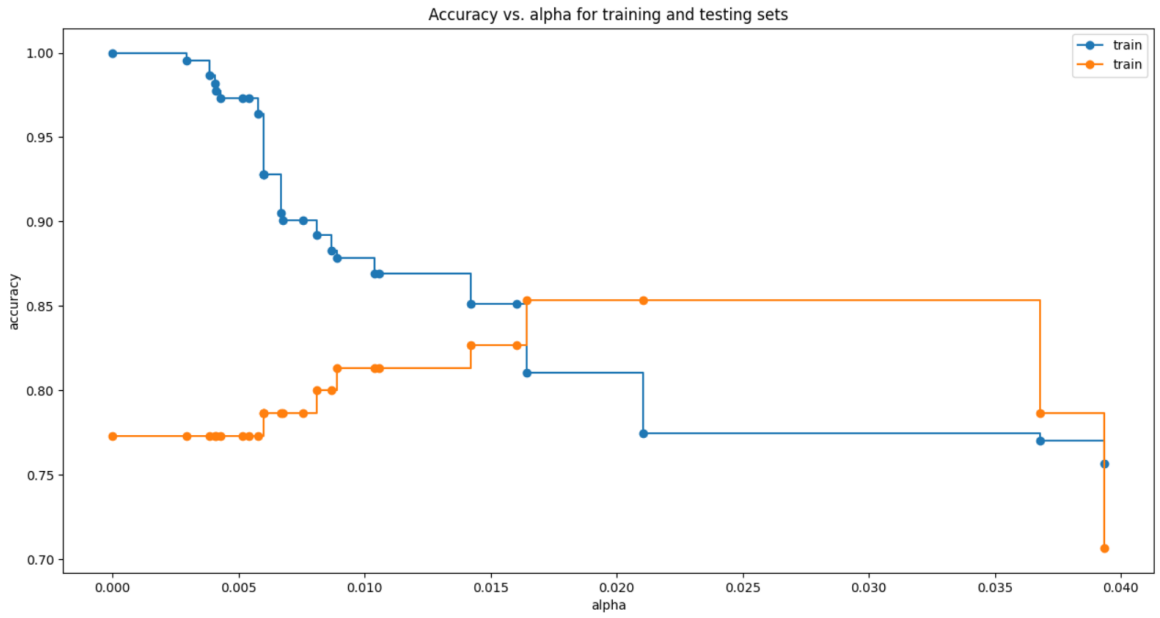
```
path = clf_dt.cost_complexity_pruning_path(X_train, y_train)
ccp_alphas = path.ccp_alphas
```

```
ccp_alphas = ccp_alphas[:-1]

clf_dts = []

for ccp_alpha in ccp_alphas:
    clf_dt = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    clf_dt.fit(X_train, y_train)
    clf_dts.append(clf_dt)
```

- به ازای آلفاهای متفاوت، Accuracy را محاسبه کنید و نمودار آن را برای آموزش و تست در یک شکل رسم کنید. بهترین نتیجه برای کدام مقدار آلفا است؟
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
train_scores = [clf_dt.score(X_train, y_train) for clf_dt in clf_dts]
test_scores = [clf_dt.score(X_test, y_test) for clf_dt in clf_dts]
fig, ax = plt.subplots()
ax.set_xlabel("alpha")
ax.set_ylabel("accuracy")
ax.set_title("Accuracy vs. alpha for training and testing sets")
ax.plot(ccp_alphas, train_scores, marker='o', label='train', drawstyle='steps-post')
ax.plot(ccp_alphas, test_scores, marker='o', label='train', drawstyle='steps-post')
ax.legend()
plt.show()
```

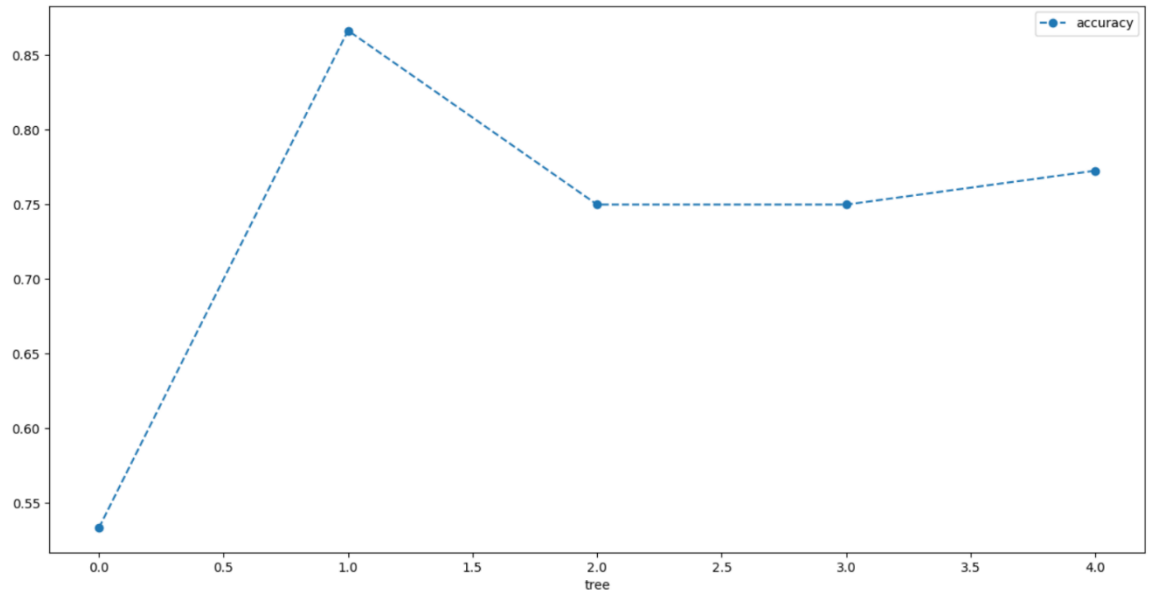
که نمودار آن به شکل زیر است:



- به ازای آلفاهای متفاوت 5-fold cross validation را اجرا کنید و میانگین و واریانس، Accuracy را محاسبه و در نموداری این خروجی‌ها را نمایش دهید:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
clf_dt = DecisionTreeClassifier(random_state=42, ccp_alpha=0.016)

scores = cross_val_score(clf_dt, X_train, y_train, cv=5)
df = pd.DataFrame(data={'tree': range(5), 'accuracy': scores})
df.plot(x='tree', y='accuracy', marker='o', linestyle='--')
plt.show()
```

که نمودار آن به شکل زیر است:

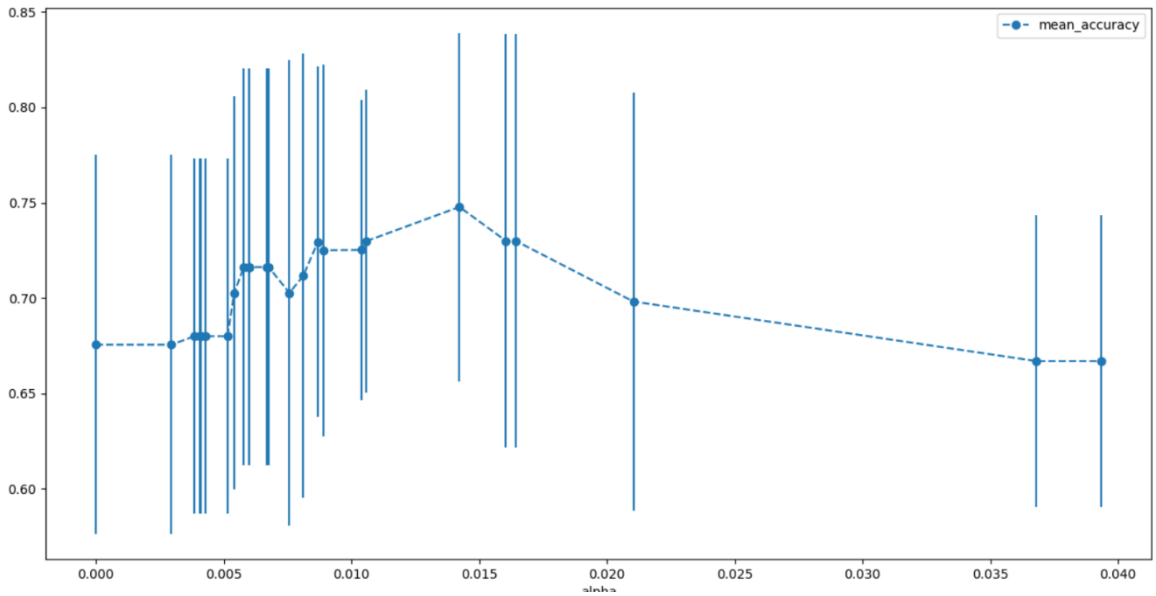


که این نمودار نشان می‌دهد که به ازای داده‌های آموزشی و تست متفاوت مقدار آلفا باعث به دست آمدن Accuracyهای متفاوتی می‌شود یعنی مقدار آلفا به دیتاست‌ها وابسته است. پس به جای استفاده از یک داده آموزشی و تست، از cross validation برای به دست آوردن مقدار بهینه آلفا استفاده می‌کنیم که به صورت زیر است:

```
alpha_loop_values = []

for ccp_alpha in ccp_alphas:
```

```
    clf_dt = DecisionTreeClassifier(random_state=0, ccp_alpha=ccp_alpha)
    scores = cross_val_score(clf_dt, X_train, y_train, cv=5)
    alpha_loop_values.append([ccp_alpha, np.mean(scores), np.std(scores)])
alpha_results = pd.DataFrame(alpha_loop_values, columns=['alpha', 'mean_accura
cy', 'std'])

alpha_results.plot(x='alpha',
                   y='mean_accuracy',
                   yerr='std',
                   marker='o',
                   linestyle='--')
plt.show()
```

که نمودار آن به شکل زیر است:



- با رسم این نمودار بهترین مقدار آلفا را مشخص کنید:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
ideal_ccp_alpha = alpha_results[(alpha_results['alpha'] > 0.014) & (al
            pha_results['alpha'] < 0.015)]['alpha']
print(ideal_ccp_alpha)
ideal_ccp_alpha = float(ideal_ccp_alpha)
```

۱،۳،۴. ساخت، ارزیابی، رسم درخت تصمیم

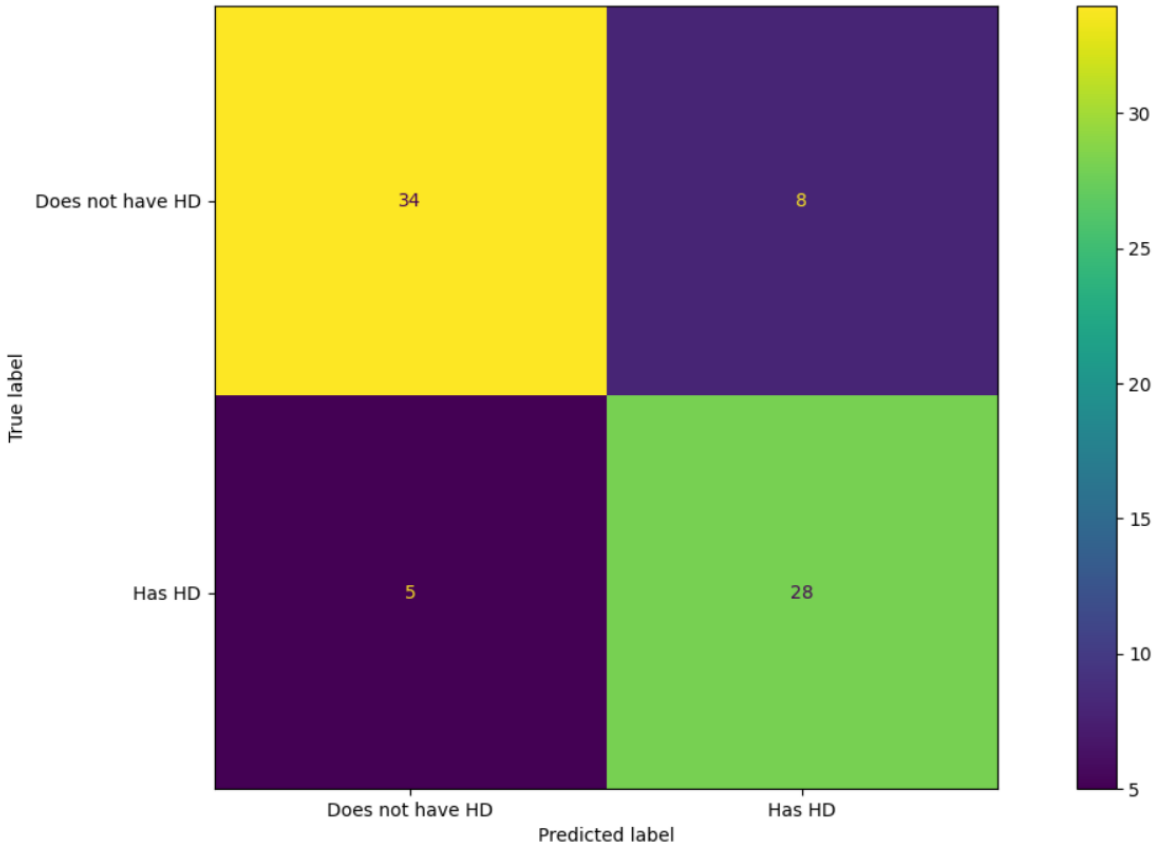
- بر اساس بهترین مقدار آلفا درخت تصمیم جدیدی بسازید:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
clf_dt_pruned = DecisionTreeClassifier(random_state=42, ccp_alpha=ideal_ccp_alpha)
clf_dt_pruned = clf_dt_pruned.fit(X_train, y_train)
```

- به ازای بهترین آلفا به دست آمده ماتریس در هم ریختگی (Confusion Matrix) را رسم کنید:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
plot_confusion_matrix(clf_dt_pruned, X_test, y_test, display_labels=["Does not have
      HD", "Has HD"])
plt.show()
```

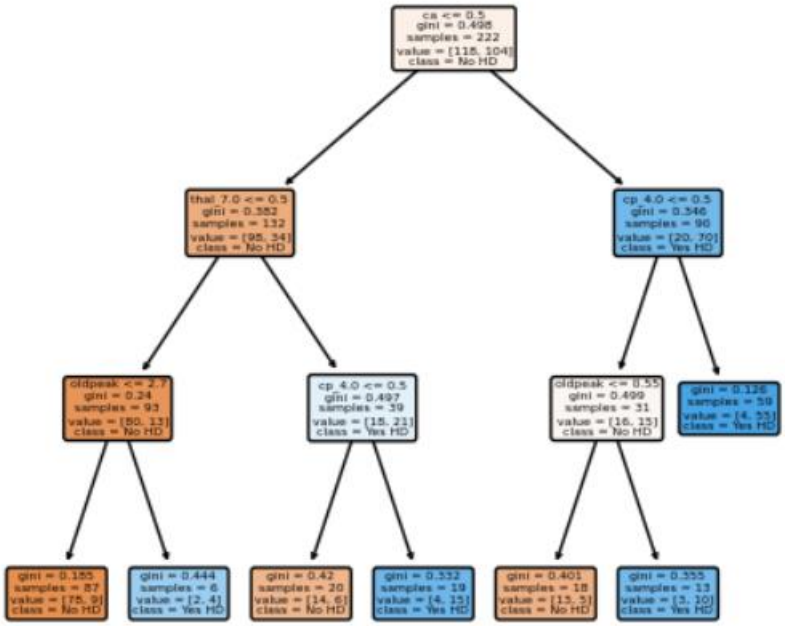
و ارزیابی آن به شکل زیر است که نشان می‌دهد از ۴۲ نفری که بیماری قلبی نداشته‌اند، ۳۴ نفر آن‌ها (۸۱٪) درست تشخیص داده شده و از ۳۳ نفری که بیماری قلبی داشته‌اند، ۲۸ نفر آن‌ها (۸۵٪) درست تشخیص داده شده‌اند و این نشان می‌دهد که مدل ما نسبت به مدل هرس نشده و قبلی عملکرد بهتری داشته است.



- ساختار درختی بعد از هرس شدن را رسم کنید:
  - راه حل: با استفاده از قطعه کد زیر انجام شده‌است:

```
plot_tree(clf_dt_pruned,
          filled=True,
          rounded=True,
          class_names=["No HD", "Yes HD"],
          feature_names=X_encoded.columns)
plt.show()
```

که نمودار درختی بعد از هرس شدن به شکل زیر نمایش داده می‌شود:



مراجع

[1] StatQuest with Josh Starmer