Лабораторная работа 3

Деревья решений. Ансамбли решающих деревьев

Выполнили студенты группы 20152 Рыбаков Максим Александрович и Ковалёва Елена Сергеевна

B [27]:
```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.tree import DecisionTreeClassifier, plot_tree
import matplotlib.pyplot as plt
from sklearn.metrics import accuracy_score
import numpy as np
from sklearn.ensemble import GradientBoostingClassifier
from sklearn.model_selection import cross_val_score
from sklearn.ensemble import RandomForestClassifier
from tqdm import tqdm as tqdm
```

B [2]:
```
features = ['fLength', 'fWidth', 'fSize', 'fConc',
            'fConc1', 'fAsym', 'fM3Long', 'fM3Trans',
            'fAlpha', 'fDist', 'class']

df = pd.read_table('magic04.data', sep= ',', names=features)
display(df)

data = df
data['class'][data['class'] == 'g'] = 0
data['class'][data['class'] == 'h'] = 1

data['class'] = pd.DataFrame(data = data['class'], columns = ['class'], dtype = "int64")

display(data)
data_x = data
data_y = data['class']
```

	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	class
0	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.0110	-8.2027	40.0920	81.8828	g
1	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6.3609	205.2610	g
2	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	76.9600	256.7880	g
3	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	10.4490	116.7370	g
4	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	4.6480	356.4620	g
...	...	...	...	...	...	...	...	...	...	...	...
19015	21.3846	10.9170	2.6161	0.5857	0.3934	15.2618	11.5245	2.8766	2.4229	106.8258	h
19016	28.9452	6.7020	2.2672	0.5351	0.2784	37.0816	13.1853	-2.9632	86.7975	247.4560	h
19017	75.4455	47.5305	3.4483	0.1417	0.0549	-9.3561	41.0562	-9.4662	30.2987	256.5166	h
19018	120.5135	76.9018	3.9939	0.0944	0.0683	5.8043	-93.5224	-63.8389	84.6874	408.3166	h
19019	187.1814	53.0014	3.2093	0.2876	0.1539	-167.3125	-168.4558	31.4755	52.7310	272.3174	h

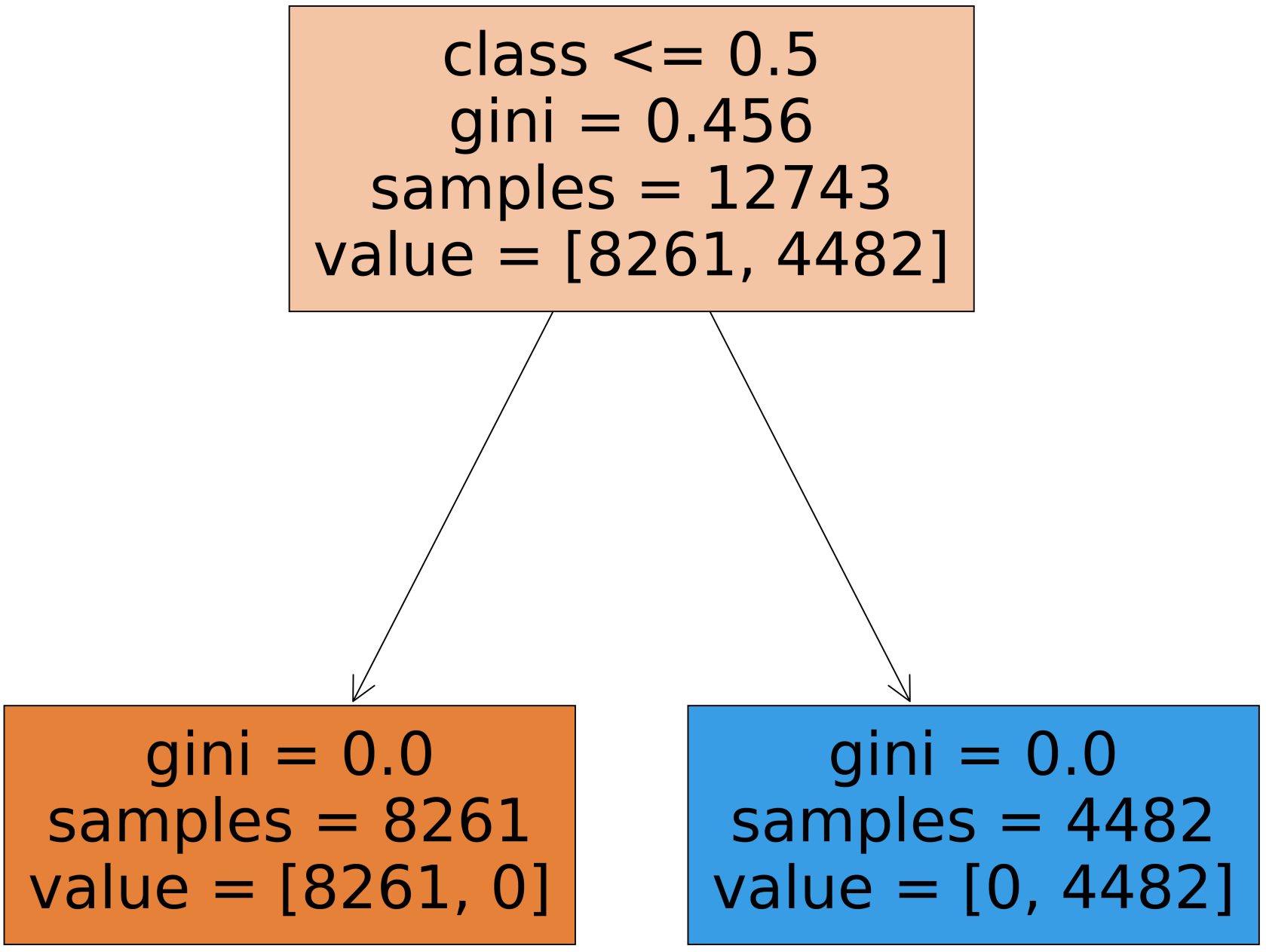
19020 rows × 11 columns

```
<ipython-input-2-41fe325d5d18>:9: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['class'][data['class'] == 'g'] = 0
<ipython-input-2-41fe325d5d18>:10: SettingWithCopyWarning:
A value is trying to be set on a copy of a slice from a DataFrame

See the caveats in the documentation: https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy (https://pandas.pydata.org/pandas-docs/stable/user_guide/indexing.html#returning-a-view-versus-a-copy)
  data['class'][data['class'] == 'h'] = 1
```

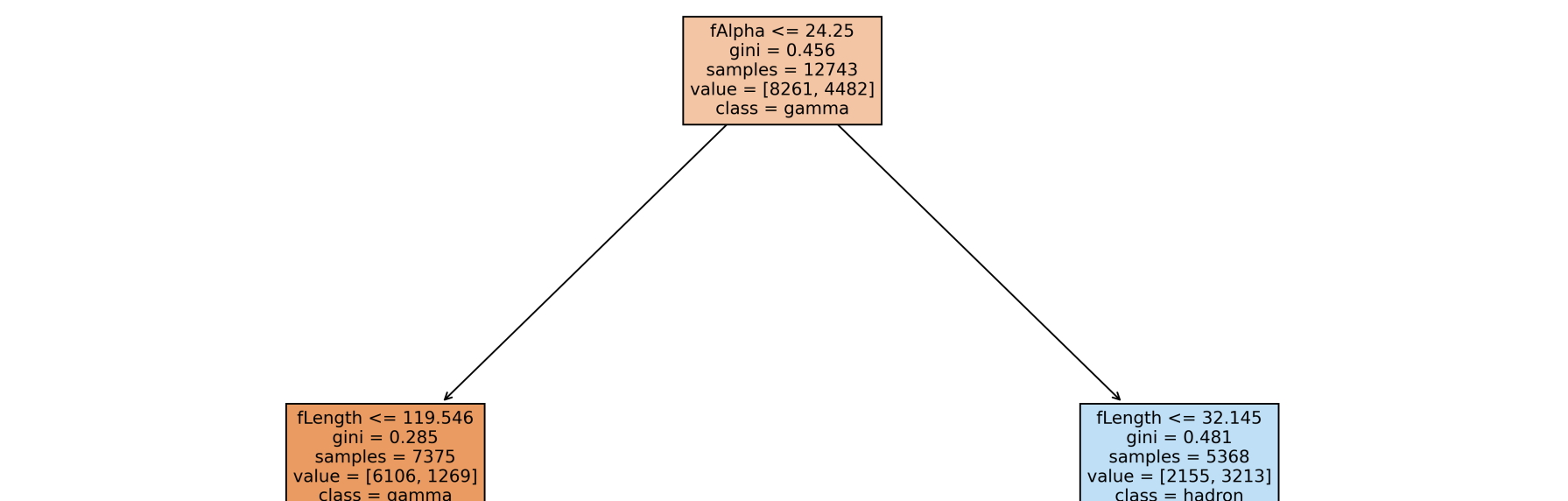
	fLength	fWidth	fSize	fConc	fConc1	fAsym	fM3Long	fM3Trans	fAlpha	fDist	class
0	28.7967	16.0021	2.6449	0.3918	0.1982	27.7004	22.0110	-8.2027	40.0920	81.8828	0
1	31.6036	11.7235	2.5185	0.5303	0.3773	26.2722	23.8238	-9.9574	6.3609	205.2610	0
2	162.0520	136.0310	4.0612	0.0374	0.0187	116.7410	-64.8580	-45.2160	76.9600	256.7880	0
3	23.8172	9.5728	2.3385	0.6147	0.3922	27.2107	-6.4633	-7.1513	10.4490	116.7370	0
4	75.1362	30.9205	3.1611	0.3168	0.1832	-5.5277	28.5525	21.8393	4.6480	356.4620	0
...	...	...	...	...	...	...	...	...	...	...	...
19015	21.3846	10.9170	2.6161	0.5857	0.3934	15.2618	11.5245	2.8766	2.4229	106.8258	1
19016	28.9452	6.7020	2.2672	0.5351	0.2784	37.0816	13.1853	-2.9632	86.7975	247.4560	1
19017	75.4455	47.5305	3.4483	0.1417	0.0549	-9.3561	41.0562	-9.4662	30.2987	256.5166	1
19018	120.5135	76.9018	3.9939	0.0944	0.0683	5.8043	-93.5224	-63.8389	84.6874	408.3166	1
19019	187.1814	53.0014	3.2093	0.2876	0.1539	-167.3125	-168.4558	31.4755	52.7310	272.3174	1

19020 rows × 11 columns

B [3]:
```
X_train, X_test, y_train, y_test = \
 train_test_split(data_x, data_y, test_size=0.33, random_state=42)

decision_tree = DecisionTreeClassifier(max_depth=3)
#decision_tree = DecisionTreeClassifier()

decision_tree.fit(X_train, y_train).predict(X_test)
plt.figure(figsize=(17, 17), dpi=360)
plot_tree(decision_tree, feature_names=X_train.columns, filled=True)
plt.show()
```



B [4]:
```
# data_x = data.drop(columns=['class', 'fWidth', 'fSize', 'fConc',
#             'fConc1', 'fAsym', 'fM3Long', 'fM3Trans', 'fAlpha', 'fDist'])

data_x = data.drop(columns=['class'])

X_train, X_test, y_train, y_test = train_test_split(data_x, data_y, test_size=0.33, random_state=42)

decision_tree.fit(X_train, y_train)
plt.figure(figsize=(17, 17), dpi=360)
plot_tree(decision_tree, feature_names=X_train.columns, filled=True,
class_names=['gamma', 'hadron'])
plt.show()
```

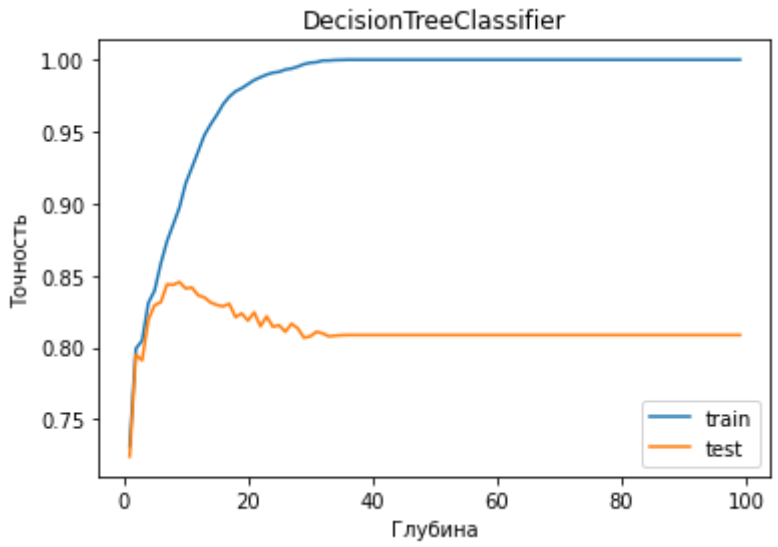


B [5]:
```
decision_tree.fit(X_train, y_train)

print("Traning accuracy: ", accuracy_score(y_train, decision_tree.predict(X_train)))
print("Test accuracy: ", accuracy_score(y_test, decision_tree.predict(X_test)))
```

```
Traning accuracy:  0.8048340265243663
Test accuracy:  0.790982953640274
```

B [6]:
```
train_scores = []
test_scores = []
depths = range(1, 100)
for depth in (depths):
    classifier = DecisionTreeClassifier(max_depth=depth, random_state=42)
    classifier.fit(X_train, y_train)
    train_scores += [accuracy_score(y_train, classifier.predict(X_train))]
    test_scores += [accuracy_score(y_test, classifier.predict(X_test))]

plt.plot(depths, train_scores, label='train')
plt.plot(depths, test_scores, label='test')
plt.xlabel('Глубина')
plt.ylabel('Точность')
plt.legend()
plt.title('DecisionTreeClassifier')
print(f'DecisionTreeClassifier best score: {max(test_scores)} '
f'[max_depth={depths[np.argmax(test_scores)]}]')
```

```
DecisionTreeClassifier best score: 0.845467580054166 [max_depth=9]
```



Задание 2

B [7]:
```
classifier = GradientBoostingClassifier(random_state=42, n_estimators=18)
classifier.fit(X_train, y_train)
for i in range(len(classifier.feature_importances_)):
    print(f'feature {data_x.columns[i]} importance = {classifier.feature_importances_[i]}')

print('train: ', accuracy_score(y_train, classifier.predict(X_train)))
print('test: ', accuracy_score(y_test, classifier.predict(X_test)))
```

```
feature fLength importance = 0.31496643286233333
feature fWidth importance = 0.06404301580173989
feature fSize importance = 0.07621790504290968
feature fConc importance = 0.003931424867728601
feature fConc1 importance = 0.00481112846816798
feature fAsym importance = 0.0
feature fM3Long importance = 0.04280478473939863
feature fM3Trans importance = 0.0
feature fAlpha importance = 0.4729786709328849
feature fDist importance = 0.020246637284836912
train:  0.842972612414659
test:  0.8360681854389039
```

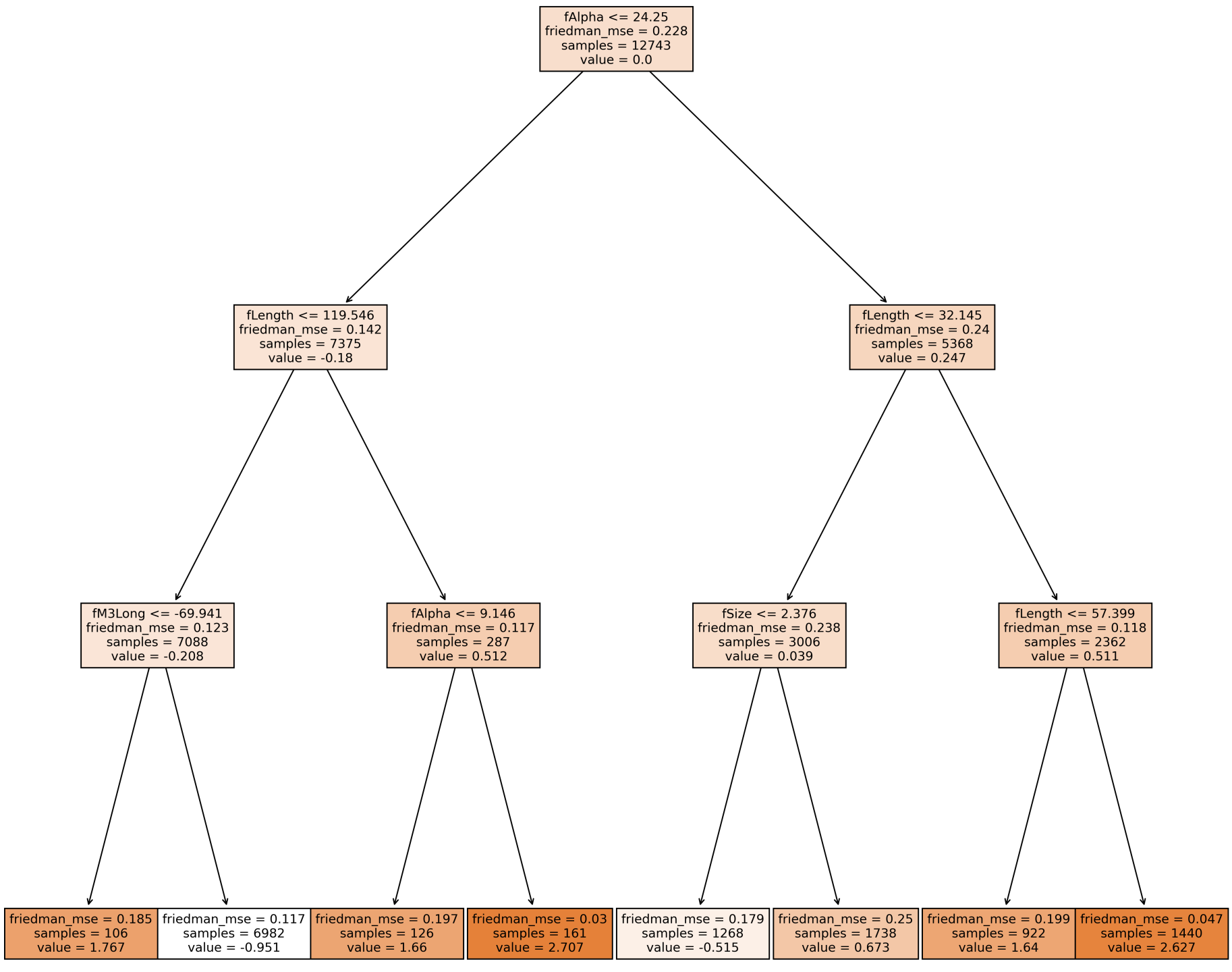
B [8]:
```
dict(zip(X_train.columns, sorted(classifier.feature_importances_, reverse=True)))
```

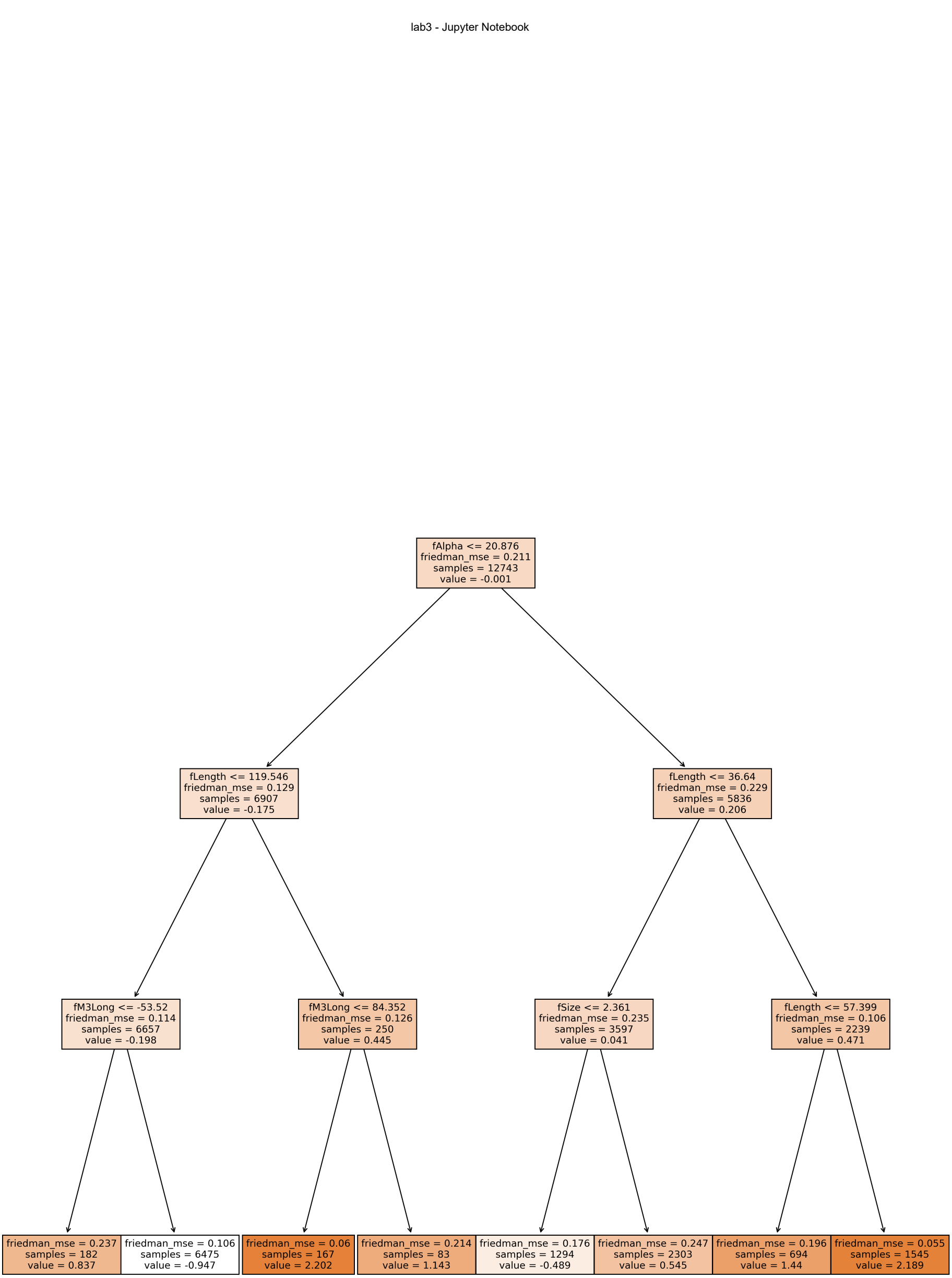
Out[8]:
```
{'fLength': 0.4729786709328849,
 'fWidth': 0.31496643286233333,
 'fSize': 0.07621790504290968,
 'fConc': 0.06404301580173989,
 'fConc1': 0.04280478473939863,
 'fAsym': 0.020246637284836912,
 'fM3Long': 0.00481112846816798,
 'fM3Trans': 0.003931424867728601,
 'fAlpha': 0.0,
 'fDist': 0.0}
```

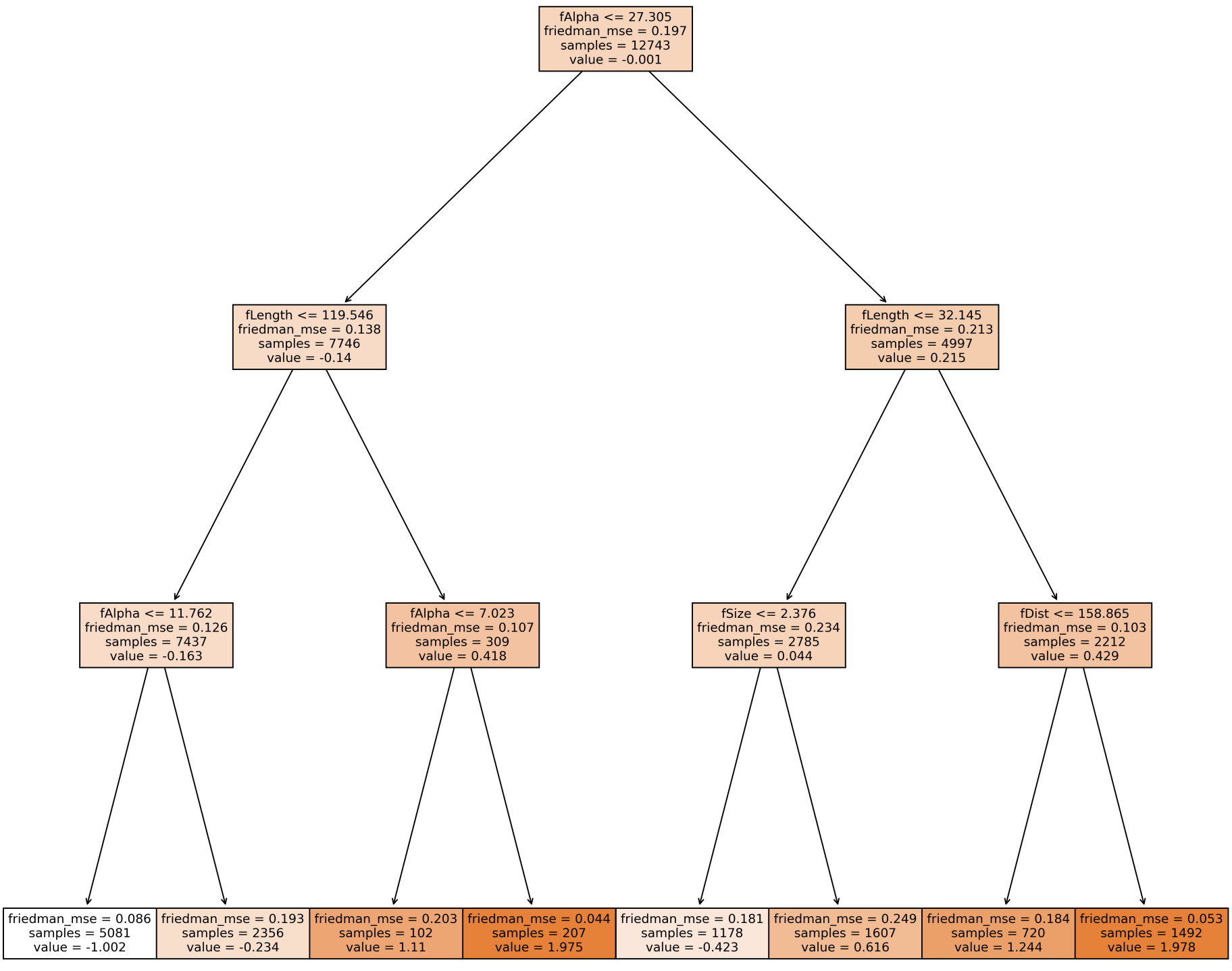
В [9]:
```
print(classifier.estimators_)
```

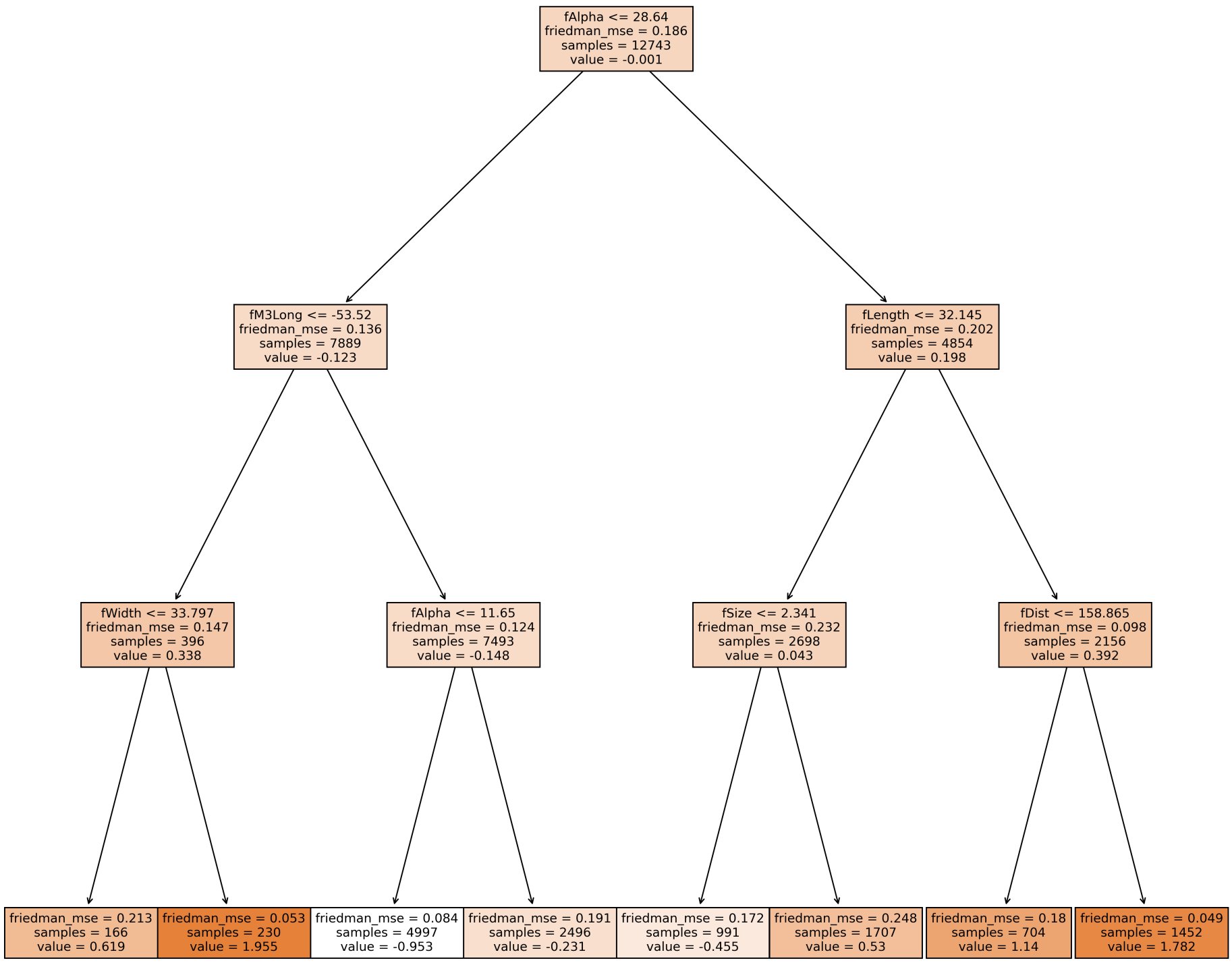
```
[[DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]
 [DecisionTreeRegressor(criterion='friedman_mse', max_depth=3,
                       random_state=RandomState(MT19937) at 0x246FFC4A640)]]
```

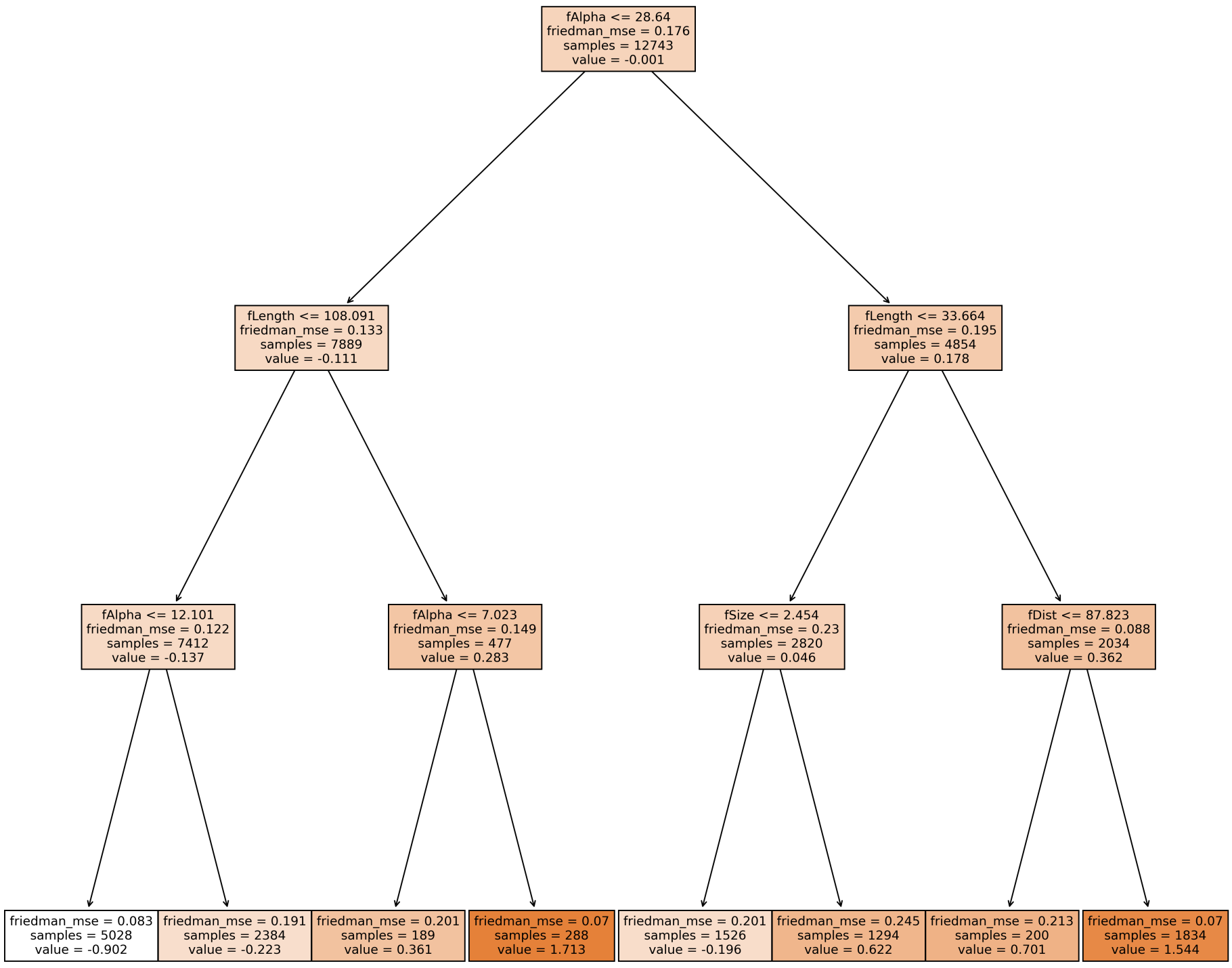
B [10]:
```
X_train,X_test,y_train,y_test=\
train_test_split(data_x,data_y,test_size=0.33,random_state=42)

for i in range(5):
    plt.figure(figsize=(17,17),dpi=360)
    plot_tree(classifier.estimators_[i,0],feature_names=X_train.columns,
              filled=True,class_names=['Fail','Pass'])
    plt.show()
```




fAlpha <= 20.876
friedman_mse = 0.211
samples = 12743
value = -0.001
fLength <= 119.546
friedman_mse = 0.129
samples = 6907
value = -0.175
fLength <= 36.64
friedman_mse = 0.229
samples = 5836
value = 0.206
fM3Long <= -53.52
friedman_mse = 0.114
samples = 6657
value = -0.198
fM3Long <= 84.352
friedman_mse = 0.126
samples = 250
value = 0.445
fSize <= 2.361
friedman_mse = 0.235
samples = 3597
value = 0.041
fLength <= 57.399
friedman_mse = 0.106
samples = 2239
value = 0.471
friedman_mse = 0.237
samples = 182
value = 0.837
friedman_mse = 0.106
samples = 6475
value = -0.947
friedman_mse = 0.06
samples = 167
value = 2.202
friedman_mse = 0.214
samples = 83
value = 1.143
friedman_mse = 0.176
samples = 1294
value = -0.489
friedman_mse = 0.247
samples = 2303
value = 0.545
friedman_mse = 0.196
samples = 694
value = 1.44
friedman_mse = 0.055
samples = 1545
value = 2.189


fAlpha <= 27.305
friedman_mse = 0.197
samples = 12743
value = -0.001
fLength <= 119.546
friedman_mse = 0.138
samples = 7746
value = -0.14
fLength <= 32.145
friedman_mse = 0.213
samples = 4997
value = 0.215
fAlpha <= 11.762
friedman_mse = 0.126
samples = 7437
value = -0.163
fAlpha <= 7.023
friedman_mse = 0.107
samples = 309
value = 0.418
fSize <= 2.376
friedman_mse = 0.234
samples = 2785
value = 0.044
fDist <= 158.865
friedman_mse = 0.103
samples = 2212
value = 0.429
friedman_mse = 0.086
samples = 5081
value = -1.002
friedman_mse = 0.193
samples = 2356
value = -0.234
friedman_mse = 0.203
samples = 102
value = 1.11
friedman_mse = 0.044
samples = 207
value = 1.975
friedman_mse = 0.181
samples = 1178
value = -0.423
friedman_mse = 0.249
samples = 1607
value = 0.616
friedman_mse = 0.184
samples = 720
value = 1.244
friedman_mse = 0.053
samples = 1492
value = 1.978


fAlpha <= 28.64
friedman_mse = 0.186
samples = 12743
value = -0.001
fM3Long <= -53.52
friedman_mse = 0.136
samples = 7889
value = -0.123
fLength <= 32.145
friedman_mse = 0.202
samples = 4854
value = 0.198
fWidth <= 33.797
friedman_mse = 0.147
samples = 396
value = 0.338
fAlpha <= 11.65
friedman_mse = 0.124
samples = 7493
value = -0.148
fSize <= 2.341
friedman_mse = 0.232
samples = 2698
value = 0.043
fDist <= 158.865
friedman_mse = 0.098
samples = 2156
value = 0.392
friedman_mse = 0.213
samples = 166
value = 0.619
friedman_mse = 0.053
samples = 230
value = 1.955
friedman_mse = 0.084
samples = 4997
value = -0.953
friedman_mse = 0.191
samples = 2496
value = -0.231
friedman_mse = 0.172
samples = 991
value = -0.455
friedman_mse = 0.248
samples = 1707
value = 0.53
friedman_mse = 0.18
samples = 704
value = 1.14
friedman_mse = 0.049
samples = 1452
value = 1.782


fAlpha <= 28.64
friedman_mse = 0.176
samples = 12743
value = -0.001
fLength <= 108.091
friedman_mse = 0.133
samples = 7889
value = -0.111
fLength <= 33.664
friedman_mse = 0.195
samples = 4854
value = 0.178
fAlpha <= 12.101
friedman_mse = 0.122
samples = 7412
value = -0.137
fAlpha <= 7.023
friedman_mse = 0.149
samples = 477
value = 0.283
fSize <= 2.454
friedman_mse = 0.23
samples = 2820
value = 0.046
fDist <= 87.823
friedman_mse = 0.088
samples = 2034
value = 0.362
friedman_mse = 0.083
samples = 5028
value = -0.902
friedman_mse = 0.191
samples = 2384
value = -0.223
friedman_mse = 0.201
samples = 189
value = 0.361
friedman_mse = 0.07
samples = 288
value = 1.713
friedman_mse = 0.201
samples = 1526
value = -0.196
friedman_mse = 0.245
samples = 1294
value = 0.622
friedman_mse = 0.213
samples = 200
value = 0.701
friedman_mse = 0.07
samples = 1834
value = 1.544

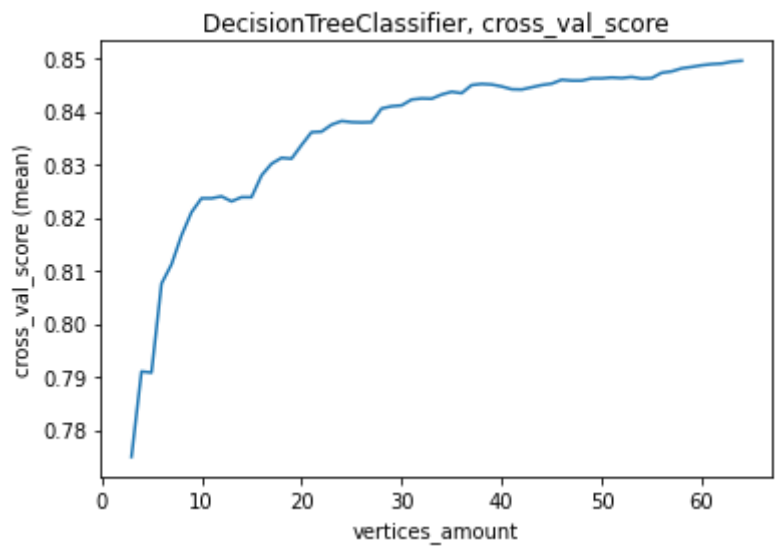
Задание 3

B [11]:
```
scores = []

for depth in (range(3, 65)):
    tree = DecisionTreeClassifier(max_depth=depth, random_state=42)
    tree.fit(X_train, y_train)
    scores += [accuracy_score(y_train, tree.predict(X_train))]

plt.plot(range(3, 65), scores)
plt.title('Decision tree')
plt.xlabel('Number of tree vertices')
plt.ylabel('Accuracy on the training sample')
```

Out[11]:
```
Text(0, 0.5, 'Accuracy on the training sample')
```

B [12]:
```
cross_val_scores = []

vertices_amounts = range(3, 65)
for vertices_amount in (vertices_amounts):
    classifier = DecisionTreeClassifier(random_state=42, max_leaf_nodes=vertices_amount)
    cross_val_scores += [cross_val_score(classifier, data_x, data_y).mean()]

plt.plot(vertices_amounts, cross_val_scores)
plt.xlabel('vertices_amount')
plt.ylabel('cross_val_score (mean)')
plt.title('DecisionTreeClassifier, cross_val_score')

print(f'DecisionTreeClassifier best score: {max(cross_val_scores)} '
    f'[max_leaf_nodes={vertices_amounts[np.argmax(cross_val_scores)]}]')
```

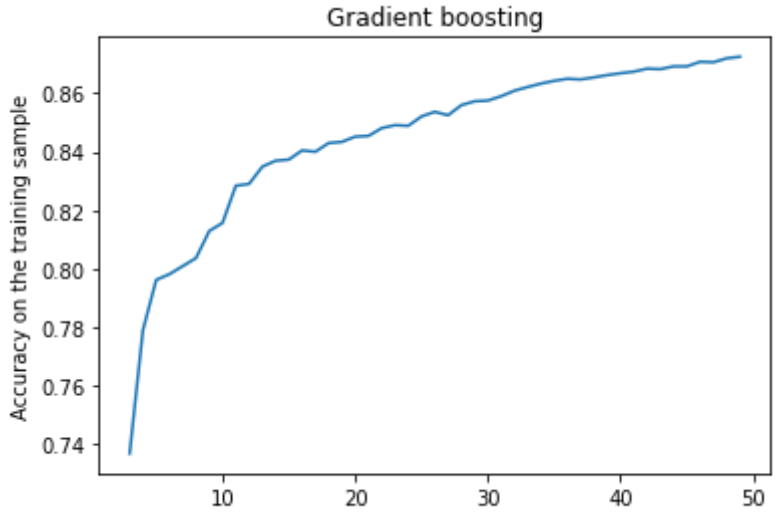
```
DecisionTreeClassifier best score: 0.8495793901156677 [max_leaf_nodes=64]
```



Задание 4

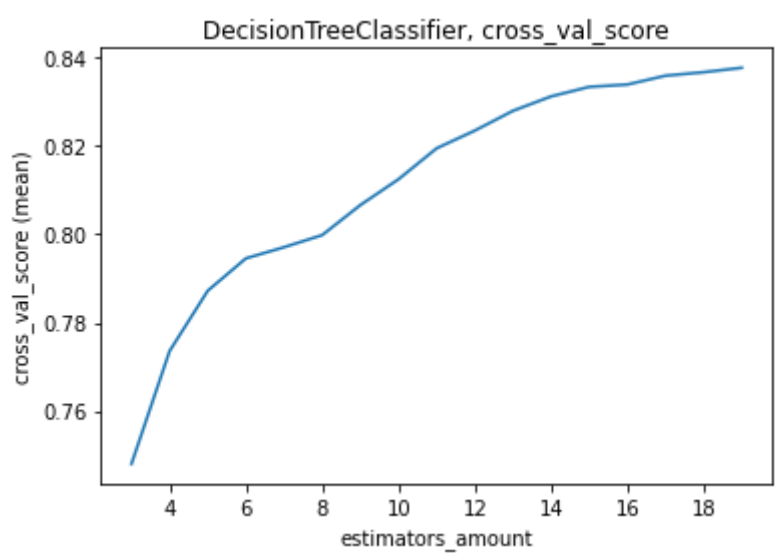
B [18]:
```
boost_scores = []
for num_trees in (range(3, 50)):
    boosting = GradientBoostingClassifier(n_estimators=num_trees, random_state=48)
    boosting.fit(X_train, y_train)
    boost_scores += [accuracy_score(y_train, boosting.predict(X_train))]

plt.plot(range(3, 50), boost_scores)
plt.title('Gradient boosting')
plt.xlabel('Number of trees')
plt.ylabel('Accuracy on the training sample')
```

Out[18]: Text(0, 0.5, 'Accuracy on the training sample')



B [21]:
```
cross_val_scores = []
estimators_amounts = range(3, 20)

for estimators_amount in (estimators_amounts):
    classifier = GradientBoostingClassifier(random_state=42, n_estimators=estimators_amount)
    cross_val_scores += [cross_val_score(classifier, data_x, data_y).mean()]

plt.plot(estimators_amounts, cross_val_scores)
plt.xlabel('estimators_amount')
plt.ylabel('cross_val_score (mean)')
plt.title('DecisionTreeClassifier, cross_val_score')


print(f'GradientBoostingClassifier best score: {max(cross_val_scores)} 'f'[n_estimators={estimators_amounts[np.argmax(cr
```

```
GradientBoostingClassifier best score: 0.8375920084121977 [n_estimators=19]
```



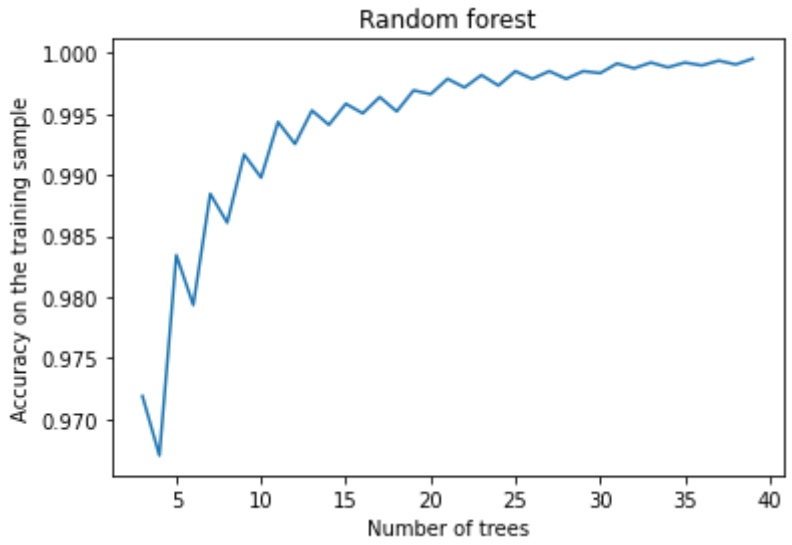
B [22]:
```
clf = GradientBoostingClassifier(n_estimators=20)
cross_val_score(clf, data_x, data_y, cv=5).mean()
```

Out[22]: 0.8388538380651946

Задание 5

B [25]:
```
rfc_scores = []

for num_trees in (range(3, 40)):
    rfc = RandomForestClassifier(n_estimators=num_trees, random_state=48)
    rfc.fit(X_train, y_train)
    rfc_scores += [accuracy_score(y_train, rfc.predict(X_train))]

plt.plot(range(3, 40), rfc_scores)
plt.title('Random forest')
plt.xlabel('Number of trees')
plt.ylabel('Accuracy on the training sample')
```

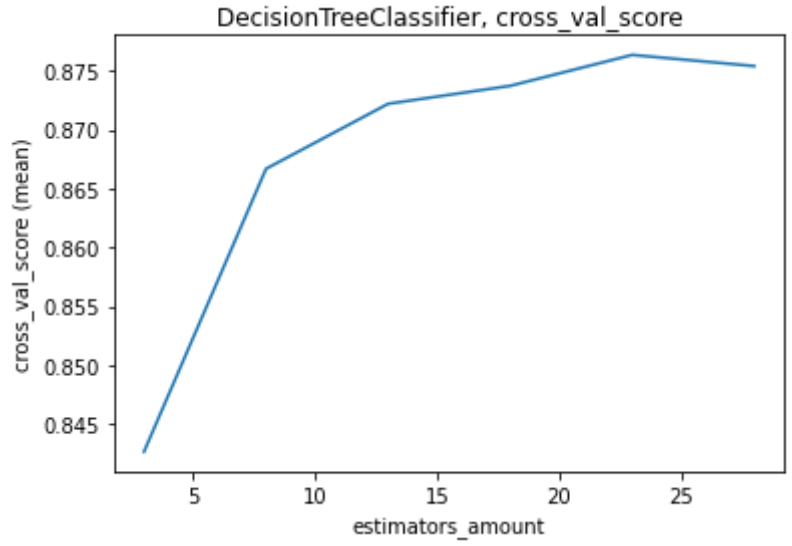
Out[25]: Text(0, 0.5, 'Accuracy on the training sample')



B [31]:
```
cross_val_scores = []
estimators_amounts = range(3, 33, 5)

for estimators_amount in tqdm(estimators_amounts):
    classifier = RandomForestClassifier(random_state=42, n_estimators=estimators_amount)
    cross_val_scores += [cross_val_score(classifier, data_x, data_y).mean()]

plt.plot(estimators_amounts, cross_val_scores)
plt.xlabel('estimators_amount')
plt.ylabel('cross_val_score (mean)')
plt.title('DecisionTreeClassifier, cross_val_score')
print(f'RandomForestClassifier best score: {max(cross_val_scores)} '
    f'[n_estimators={estimators_amounts[np.argmax(cross_val_scores)]}]')
```

100%|██████████| 6/6 [00:16<00:00, 2.76s/it]

RandomForestClassifier best score: 0.8763406940063092 [n_estimators=23]



B [33]:
```
clf = RandomForestClassifier(n_estimators=30)
cross_val_score(clf, data_x, data_y, cv=5).mean()
```

Out[33]: 0.8781808622502629