**Московский государственный технический**

**университет им. Н.Э. Баумана**

Факультет «Информатика с системы управления»

Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Технологии машинного обучения»

Отчёт по лабораторной работе №5

Выполнил: Проверил:

студент группы РТ5-61Б преподаватель каф. ИУ5

Кузнецов А.В. Гапанюк Ю.Е.

Подпись и дата: Подпись и дата:

2023 г.

# Ансамбли моделей машинного обучения.

## Описание датасета

Ирисы фишера - выбрвнный датасет, он отлично подходит для решения задач классификации

import numpy as np  
import pydotplus  
import pandas as pd  
from typing import Dict, Tuple  
import matplotlib.pyplot as plt  
from scipy import stats  
from sklearn.model\_selection import train\_test\_split  
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier  
from sklearn.metrics import accuracy\_score, balanced\_accuracy\_score  
from sklearn.metrics import precision\_score, recall\_score, f1\_score, classification\_report  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import mean\_absolute\_error, mean\_squared\_error, mean\_squared\_log\_error, median\_absolute\_error, r2\_score   
from sklearn.metrics import roc\_curve, roc\_auc\_score  
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export\_graphviz  
import seaborn as sns  
from sklearn import preprocessing  
from sklearn import datasets  
from sklearn import utils  
from sklearn.ensemble import BaggingClassifier  
from IPython.display import Image  
from sklearn.ensemble import AdaBoostClassifier  
from io import StringIO   
from sklearn.linear\_model import LinearRegression  
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor  
from heamy.estimator import Regressor, Classifier  
from heamy.pipeline import ModelsPipeline  
from heamy.dataset import Dataset  
  
ds = datasets.load\_iris()  
ds.data[:5]  
np.unique(ds.target)  
iris\_df = pd.DataFrame(data= np.c\_[ds['data'], ds['target']],  
 columns= ds['feature\_names'] + ['target'])  
iris\_df.describe()

sepal length (cm) sepal width (cm) petal length (cm) \  
count 150.000000 150.000000 150.000000   
mean 5.843333 3.057333 3.758000   
std 0.828066 0.435866 1.765298   
min 4.300000 2.000000 1.000000   
25% 5.100000 2.800000 1.600000   
50% 5.800000 3.000000 4.350000   
75% 6.400000 3.300000 5.100000   
max 7.900000 4.400000 6.900000   
  
 petal width (cm) target   
count 150.000000 150.000000   
mean 1.199333 1.000000   
std 0.762238 0.819232   
min 0.100000 0.000000   
25% 0.300000 0.000000   
50% 1.300000 1.000000   
75% 1.800000 2.000000   
max 2.500000 2.000000

## Деление выборки на обучающую и тестовую

X\_train, X\_test, y\_train, y\_test = train\_test\_split(  
 ds.data, ds.target, test\_size=0.5, random\_state=1)

# Размер обучающей модели  
X\_train.shape, y\_train.shape

((75, 4), (75,))

# Размер тестовой выборки  
X\_test.shape, y\_test.shape

((75, 4), (75,))

## Обучение моделей

### Бэггинг

# Обучим классификатор на 5 деревьях  
iris\_X = ds.data[:, :2]  
iris\_y = ds.target  
bc1 = BaggingClassifier(n\_estimators=5, oob\_score=True, random\_state=10)  
bc1.fit(iris\_X, iris\_y)

/usr/local/lib/python3.9/site-packages/sklearn/ensemble/\_bagging.py:789: UserWarning: Some inputs do not have OOB scores. This probably means too few estimators were used to compute any reliable oob estimates.  
 warn(  
/usr/local/lib/python3.9/site-packages/sklearn/ensemble/\_bagging.py:795: RuntimeWarning: invalid value encountered in divide  
 oob\_decision\_function = predictions / predictions.sum(axis=1)[:, np.newaxis]

BaggingClassifier(n\_estimators=5, oob\_score=True, random\_state=10)

Далее мы посмотрим какие объекты были использованы в обучающей выборке каждого дерева

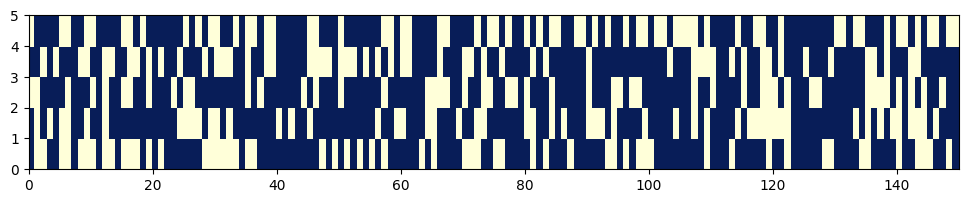
bc1.estimators\_samples\_

[array([137, 103, 142, 138, 26, 138, 50, 126, 67, 34, 24, 43, 149,  
 58, 112, 118, 104, 46, 104, 27, 74, 147, 37, 45, 132, 44,  
 142, 69, 74, 23, 108, 64, 0, 50, 78, 42, 112, 77, 50,  
 4, 114, 14, 56, 105, 43, 39, 43, 139, 80, 127, 116, 56,  
 54, 110, 138, 136, 4, 79, 62, 44, 60, 111, 74, 114, 125,  
 137, 102, 88, 14, 130, 107, 110, 118, 41, 62, 66, 37, 14,  
 52, 120, 117, 68, 73, 39, 104, 92, 44, 139, 22, 66, 107,  
 27, 85, 54, 40, 146, 95, 38, 92, 97, 61, 116, 73, 116,  
 68, 48, 20, 124, 82, 37, 58, 101, 7, 123, 141, 146, 38,  
 116, 105, 91, 7, 0, 131, 3, 22, 59, 133, 20, 106, 123,  
 11, 121, 66, 18, 46, 52, 147, 62, 89, 86, 37, 115, 132,  
 38, 90, 84, 86, 25, 48, 77]),  
 array([ 95, 46, 93, 129, 44, 136, 87, 149, 61, 68, 87, 102, 31,  
 34, 17, 58, 38, 79, 149, 88, 70, 22, 88, 112, 115, 52,  
 147, 19, 15, 49, 39, 41, 130, 14, 47, 127, 18, 13, 138,  
 13, 102, 15, 57, 111, 145, 62, 23, 128, 16, 67, 50, 115,  
 82, 43, 55, 7, 105, 4, 51, 102, 98, 35, 124, 52, 35,  
 58, 67, 148, 49, 103, 110, 16, 123, 13, 67, 125, 20, 33,  
 100, 0, 63, 20, 79, 4, 115, 100, 18, 131, 78, 7, 62,  
 141, 103, 7, 75, 101, 108, 21, 48, 28, 71, 4, 66, 85,  
 54, 46, 112, 74, 125, 126, 125, 97, 17, 10, 52, 53, 21,  
 2, 113, 95, 47, 89, 103, 54, 136, 43, 8, 19, 134, 36,  
 38, 106, 37, 83, 19, 16, 132, 115, 127, 100, 74, 39, 11,  
 76, 77, 127, 96, 115, 21, 28]),  
 array([ 54, 131, 76, 112, 99, 54, 92, 55, 38, 141, 40, 18, 29,  
 42, 61, 54, 21, 42, 82, 7, 28, 111, 72, 71, 51, 81,  
 124, 24, 39, 123, 129, 53, 27, 58, 125, 83, 106, 34, 139,  
 141, 17, 114, 113, 85, 124, 7, 7, 20, 2, 128, 149, 32,  
 56, 146, 5, 61, 30, 116, 105, 79, 54, 117, 49, 40, 8,  
 30, 121, 87, 48, 18, 103, 101, 58, 134, 146, 45, 33, 55,  
 43, 27, 27, 143, 85, 31, 13, 29, 102, 133, 141, 63, 146,  
 109, 99, 91, 148, 11, 145, 47, 3, 128, 108, 59, 129, 9,  
 79, 41, 22, 85, 21, 58, 148, 104, 86, 100, 60, 39, 68,  
 14, 13, 3, 121, 89, 148, 21, 132, 22, 128, 69, 145, 56,  
 33, 117, 52, 131, 58, 4, 11, 60, 62, 75, 49, 43, 88,  
 93, 96, 30, 7, 33, 36, 130]),  
 array([148, 104, 86, 55, 15, 120, 148, 97, 87, 85, 34, 67, 145,  
 42, 98, 114, 44, 88, 7, 127, 112, 18, 104, 120, 68, 20,  
 94, 14, 5, 101, 102, 130, 29, 53, 75, 111, 132, 22, 91,  
 26, 42, 5, 140, 65, 100, 55, 55, 94, 84, 139, 78, 37,  
 20, 94, 59, 82, 133, 149, 41, 104, 43, 95, 96, 69, 106,  
 99, 137, 111, 84, 64, 41, 68, 147, 27, 85, 63, 149, 105,  
 123, 122, 80, 62, 36, 1, 96, 134, 93, 68, 7, 6, 77,  
 25, 137, 23, 123, 146, 22, 98, 3, 145, 128, 57, 111, 91,  
 79, 0, 37, 72, 3, 93, 44, 15, 145, 105, 147, 65, 95,  
 87, 92, 132, 3, 130, 36, 122, 146, 72, 116, 105, 75, 131,  
 49, 49, 141, 126, 89, 130, 10, 40, 93, 33, 74, 15, 11,  
 72, 144, 15, 124, 117, 78, 111]),  
 array([144, 14, 119, 31, 54, 51, 32, 147, 139, 95, 37, 112, 103,  
 8, 24, 113, 68, 62, 139, 13, 11, 128, 112, 42, 126, 17,  
 63, 71, 17, 142, 139, 127, 56, 62, 94, 20, 73, 81, 90,  
 87, 62, 90, 132, 8, 40, 1, 8, 43, 4, 144, 44, 136,  
 144, 47, 24, 73, 116, 112, 78, 135, 86, 129, 81, 92, 120,  
 100, 53, 125, 49, 108, 97, 126, 111, 21, 123, 22, 21, 83,  
 23, 122, 95, 124, 136, 41, 103, 78, 12, 129, 3, 11, 94,  
 64, 87, 95, 100, 24, 70, 137, 3, 125, 69, 28, 49, 14,  
 7, 22, 32, 115, 37, 116, 70, 32, 34, 83, 97, 28, 4,  
 110, 79, 147, 19, 108, 86, 112, 79, 76, 124, 68, 126, 52,  
 44, 28, 2, 116, 55, 127, 26, 56, 111, 71, 65, 87, 32,  
 79, 19, 79, 59, 48, 1, 139])]

# Сконвертируем эти данные в двоичную матрицу,   
# 1 соответствует элементам, попавшим в обучающую выборку  
bin\_array = np.zeros((5, iris\_X.shape[0]))  
for i in range(5):  
 for j in bc1.estimators\_samples\_[i]:  
 bin\_array[i][j] = 1  
bin\_array

array([[1., 0., 0., 1., 1., 0., 0., 1., 0., 0., 0., 1., 0., 0., 1., 0.,  
 0., 0., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0.,  
 0., 0., 1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 1., 1., 0.,  
 1., 0., 1., 0., 1., 0., 1., 0., 1., 0., 1., 1., 1., 1., 1., 0.,  
 1., 0., 1., 1., 1., 1., 0., 0., 0., 1., 1., 0., 0., 1., 1., 1.,  
 1., 0., 1., 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 0., 0., 1.,  
 0., 1., 0., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1.,  
 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1., 1., 1., 1., 1.,  
 0., 0., 1., 1., 1., 1., 0., 0., 1., 1., 1., 1., 0., 1., 1., 0.,  
 0., 0., 1., 1., 0., 1.],  
 [1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 1., 0., 1., 1., 1.,  
 1., 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 0., 1.,  
 0., 1., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 1., 0., 1., 1.,  
 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1.,  
 0., 0., 1., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 1.,  
 0., 0., 1., 1., 0., 1., 0., 1., 1., 1., 0., 0., 0., 1., 0., 1.,  
 1., 1., 1., 0., 1., 1., 1., 1., 0., 1., 1., 0., 1., 0., 1., 1.,  
 1., 1., 0., 1., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 1., 1.,  
 1., 1., 1., 1., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 0., 0.,  
 0., 1., 0., 1., 1., 1.],  
 [0., 0., 1., 1., 1., 1., 0., 1., 1., 1., 0., 1., 0., 1., 1., 0.,  
 0., 1., 1., 0., 1., 1., 1., 0., 1., 0., 0., 1., 1., 1., 1., 1.,  
 1., 1., 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1.,  
 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1., 1.,  
 0., 0., 0., 0., 1., 1., 0., 1., 1., 0., 0., 1., 1., 0., 0., 1.,  
 0., 1., 1., 1., 0., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0., 0.,  
 1., 0., 0., 1., 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 0., 1.,  
 1., 1., 1., 0., 1., 1., 0., 0., 0., 1., 0., 1., 1., 1., 0., 0.,  
 1., 1., 1., 1., 1., 1., 1., 0., 0., 0., 0., 1., 0., 1., 0., 1.,  
 0., 1., 1., 0., 1., 1.],  
 [1., 1., 0., 1., 0., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1.,  
 0., 0., 1., 0., 1., 0., 1., 1., 0., 1., 1., 1., 0., 1., 0., 0.,  
 0., 1., 1., 0., 1., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 0.,  
 0., 1., 0., 0., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1.,  
 1., 1., 0., 1., 1., 1., 0., 0., 1., 0., 1., 1., 0., 1., 1., 1.,  
 1., 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 1., 1.,  
 1., 1., 1., 1., 1., 1., 1., 0., 1., 1., 1., 0., 0., 0., 0., 1.,  
 1., 0., 1., 0., 1., 1., 0., 0., 1., 0., 1., 1., 1., 0., 1., 1.,  
 1., 0., 1., 1., 1., 1., 1., 0., 0., 1., 0., 1., 1., 1., 0., 0.,  
 1., 1., 1., 1., 1., 1.],  
 [0., 1., 1., 1., 1., 0., 0., 1., 1., 0., 0., 1., 1., 1., 1., 0.,  
 0., 1., 0., 1., 1., 1., 1., 1., 1., 0., 1., 0., 1., 0., 0., 1.,  
 1., 0., 1., 0., 0., 1., 0., 0., 1., 1., 1., 1., 1., 0., 0., 1.,  
 1., 1., 0., 1., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 1., 1.,  
 1., 1., 0., 0., 1., 1., 1., 1., 0., 1., 0., 0., 1., 0., 1., 1.,  
 0., 1., 0., 1., 0., 0., 1., 1., 0., 0., 1., 0., 1., 0., 1., 1.,  
 0., 1., 0., 0., 1., 0., 0., 1., 0., 0., 0., 0., 1., 0., 1., 1.,  
 1., 1., 0., 1., 1., 0., 0., 1., 1., 0., 1., 1., 1., 1., 1., 1.,  
 1., 1., 0., 0., 1., 0., 0., 1., 1., 1., 0., 1., 0., 0., 1., 0.,  
 1., 0., 0., 1., 0., 0.]])

# И визуализируем (синим цветом показаны данные, которые попали в обучающую выборку)  
fig, ax = plt.subplots(figsize=(12,2))  
ax.pcolor(bin\_array, cmap='YlGnBu')  
plt.show()



# Оценим Out-of-bag error, теоретическое значение 37%  
for i in range(5):  
 cur\_data = bin\_array[i]  
 len\_cur\_data = len(cur\_data)  
 sum\_cur\_data = sum(cur\_data)  
 (len(bin\_array[0]) - sum(bin\_array[0])) / len(bin\_array[0])  
 oob\_i = (len\_cur\_data - sum\_cur\_data) / len\_cur\_data  
 print('Для модели № {} размер OOB составляет {}%'.format(i+1, round(oob\_i, 4)\*100.0))

Для модели № 1 размер OOB составляет 38.0%  
Для модели № 2 размер OOB составляет 35.33%  
Для модели № 3 размер OOB составляет 33.33%  
Для модели № 4 размер OOB составляет 36.67%  
Для модели № 5 размер OOB составляет 42.67%

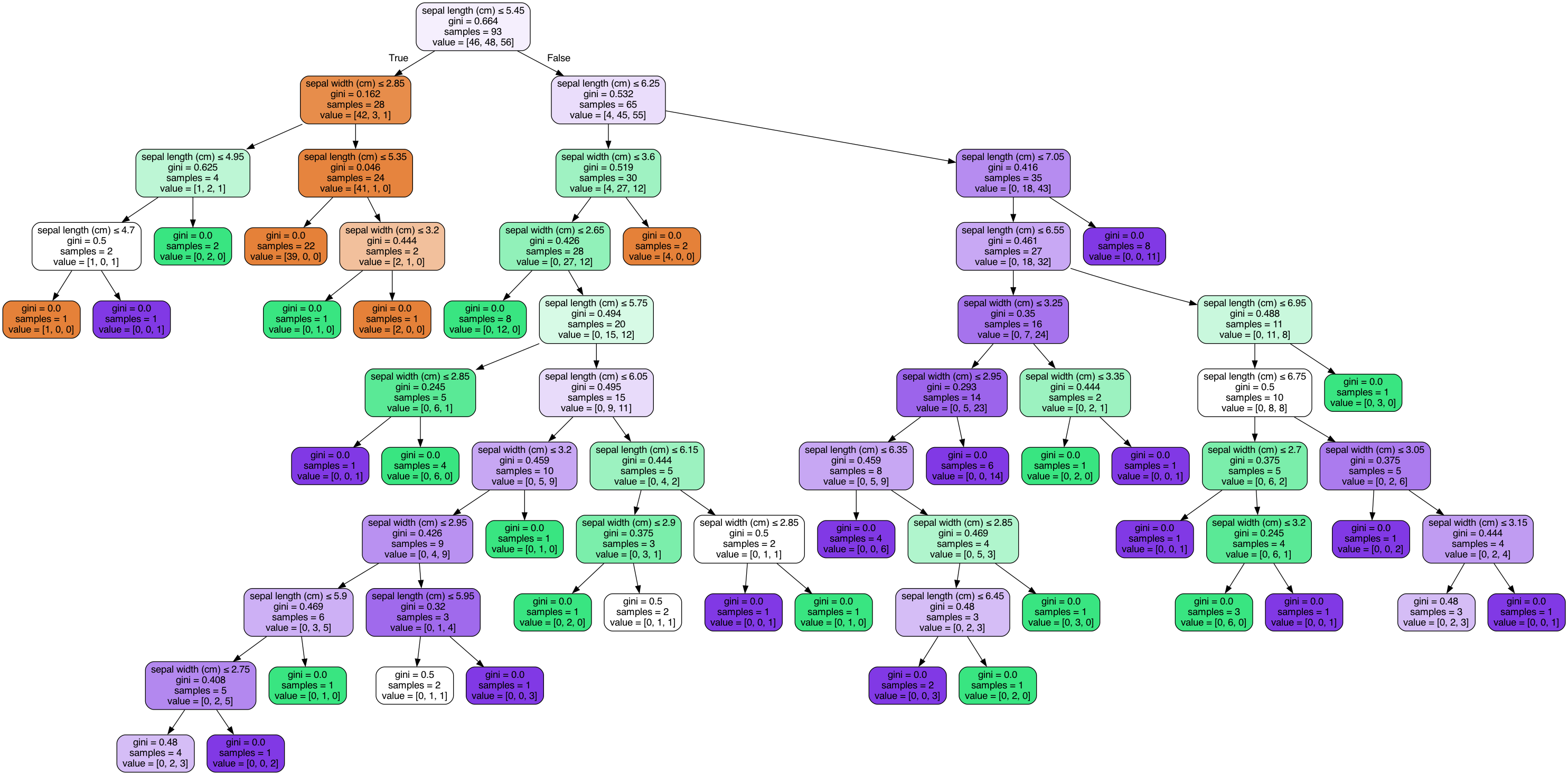
# Out-of-bag error, возвращаемый классификатором  
# Для классификации используется метрика accuracy  
bc1.oob\_score\_, 1-bc1.oob\_score\_

(0.6933333333333334, 0.30666666666666664)

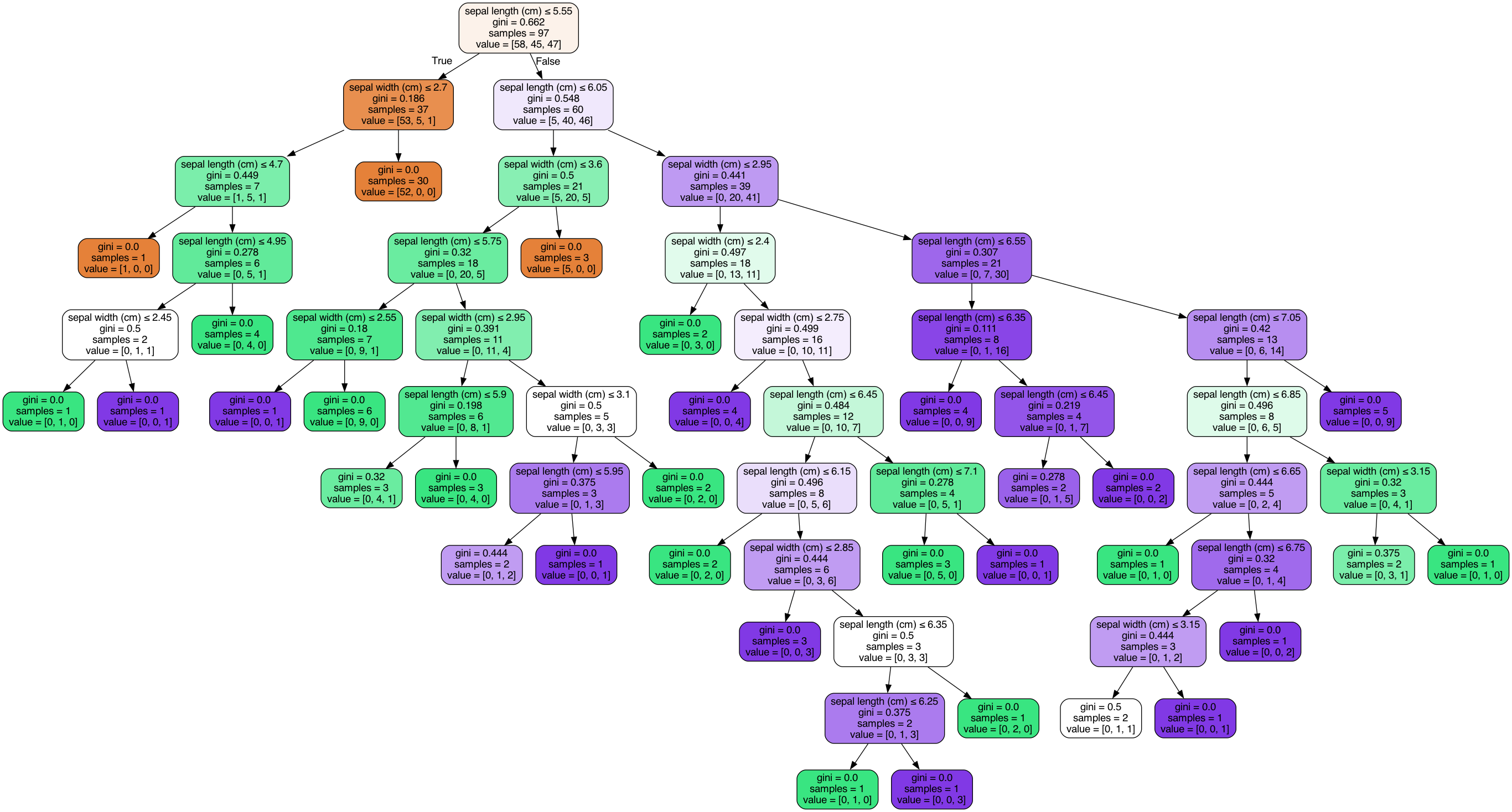
# Параметр oob\_decision\_function\_ возвращает вероятности   
# принадлежности объекта к классам на основе oob  
# В данном примере три класса,   
# значения nan могут возвращаться в случае маленькой выборки  
bc1.oob\_decision\_function\_[55:70]

array([[0. , 0. , 1. ],  
 [0. , 0. , 1. ],  
 [0. , 0.33333333, 0.66666667],  
 [0. , 0.5 , 0.5 ],  
 [0. , 1. , 0. ],  
 [0. , 1. , 0. ],  
 [0. , 0.5 , 0.5 ],  
 [ nan, nan, nan],  
 [0. , 0.5 , 0.5 ],  
 [0. , 1. , 0. ],  
 [0. , 0.83333333, 0.16666667],  
 [0. , 1. , 0. ],  
 [0. , 0.25 , 0.75 ],  
 [ nan, nan, nan],  
 [0. , 0. , 1. ]])

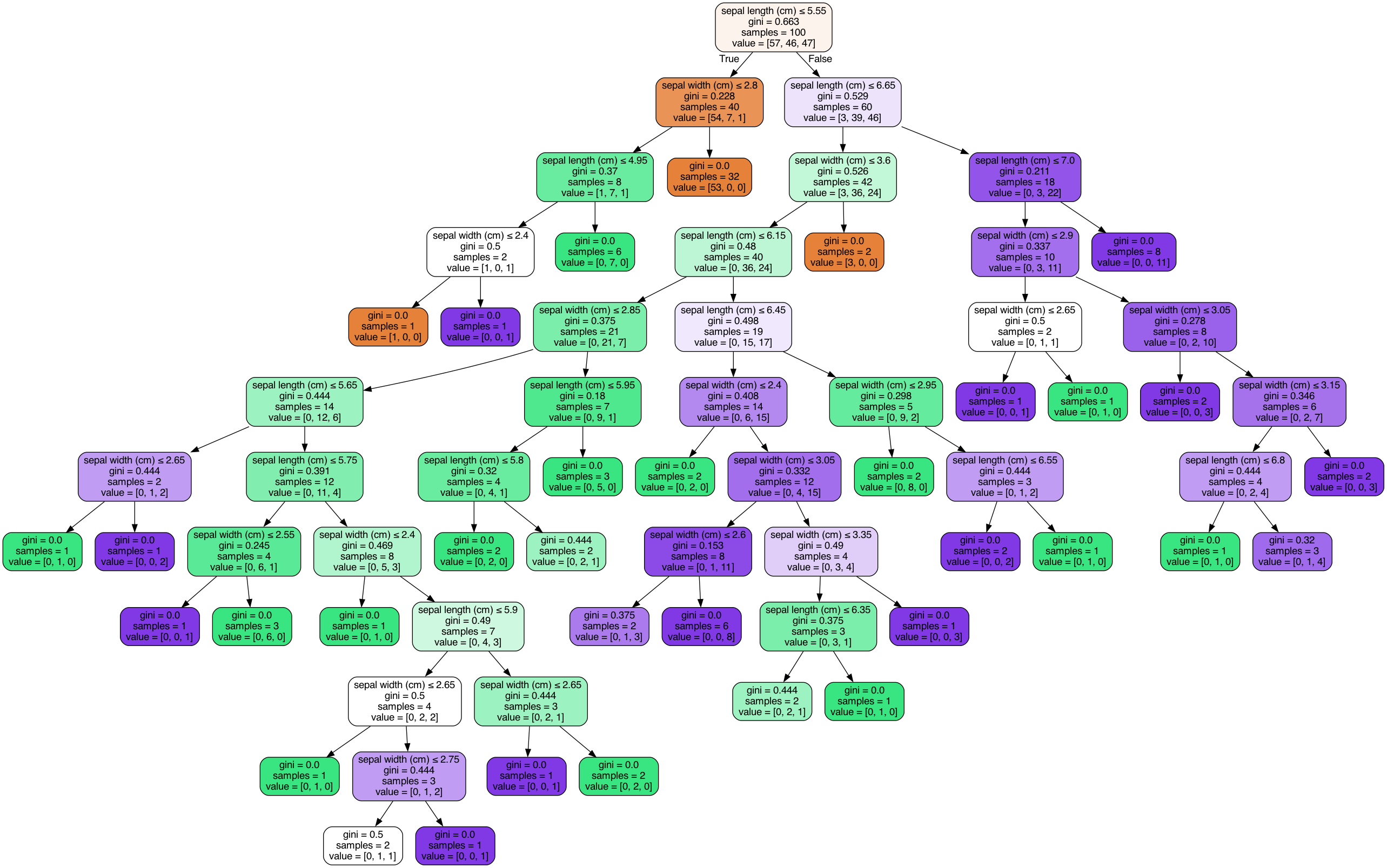
def get\_png\_tree(tree\_model\_param, feature\_names\_param):  
 dot\_data = StringIO()  
 export\_graphviz(tree\_model\_param, out\_file=dot\_data, feature\_names=feature\_names\_param,  
 filled=True, rounded=True, special\_characters=True)  
 graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())  
 return graph.create\_png()  
Image(get\_png\_tree(bc1.estimators\_[0], ds.feature\_names[:2]), width='80%')



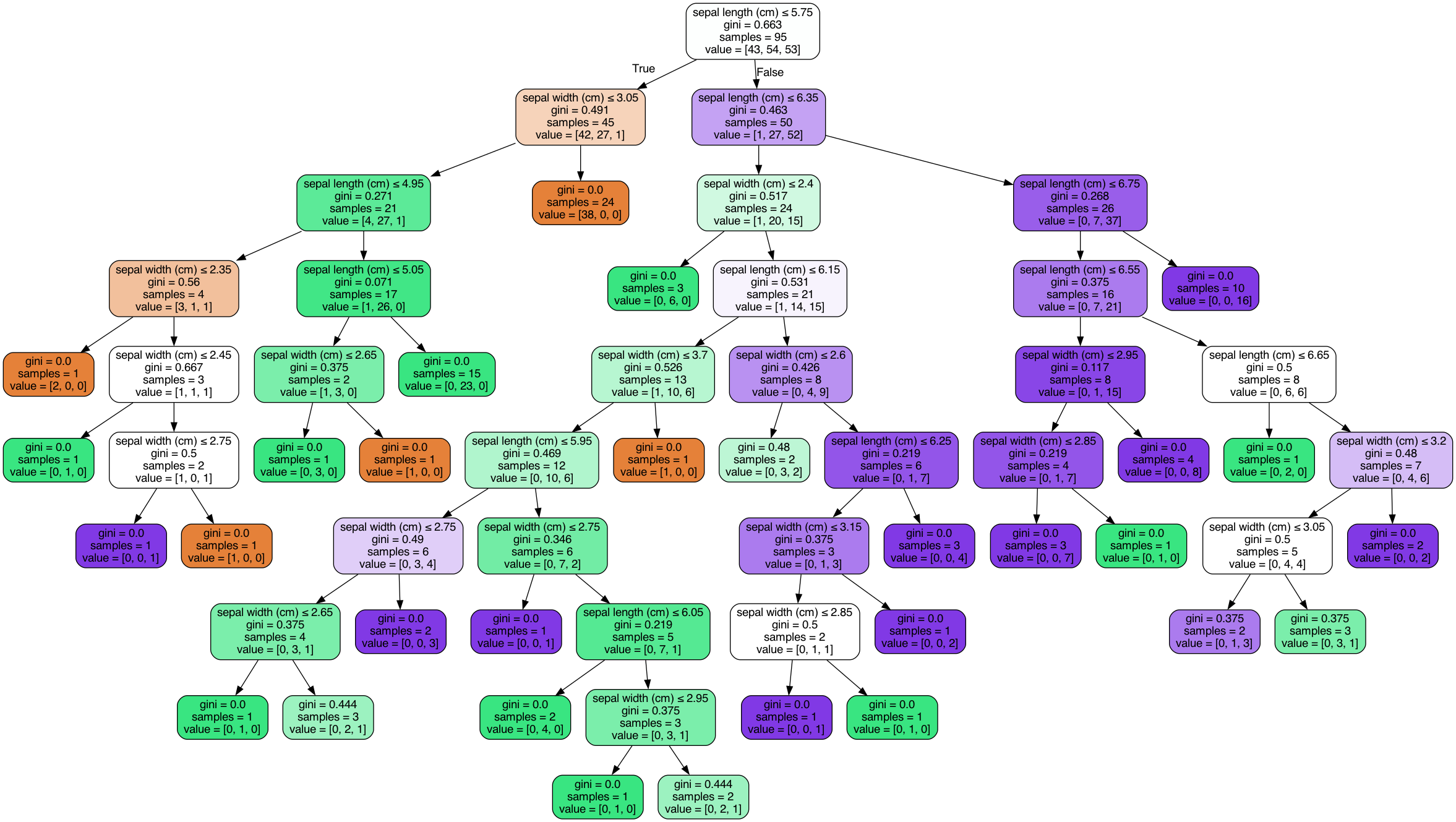
Image(get\_png\_tree(bc1.estimators\_[1], ds.feature\_names[:2]), width='80%')



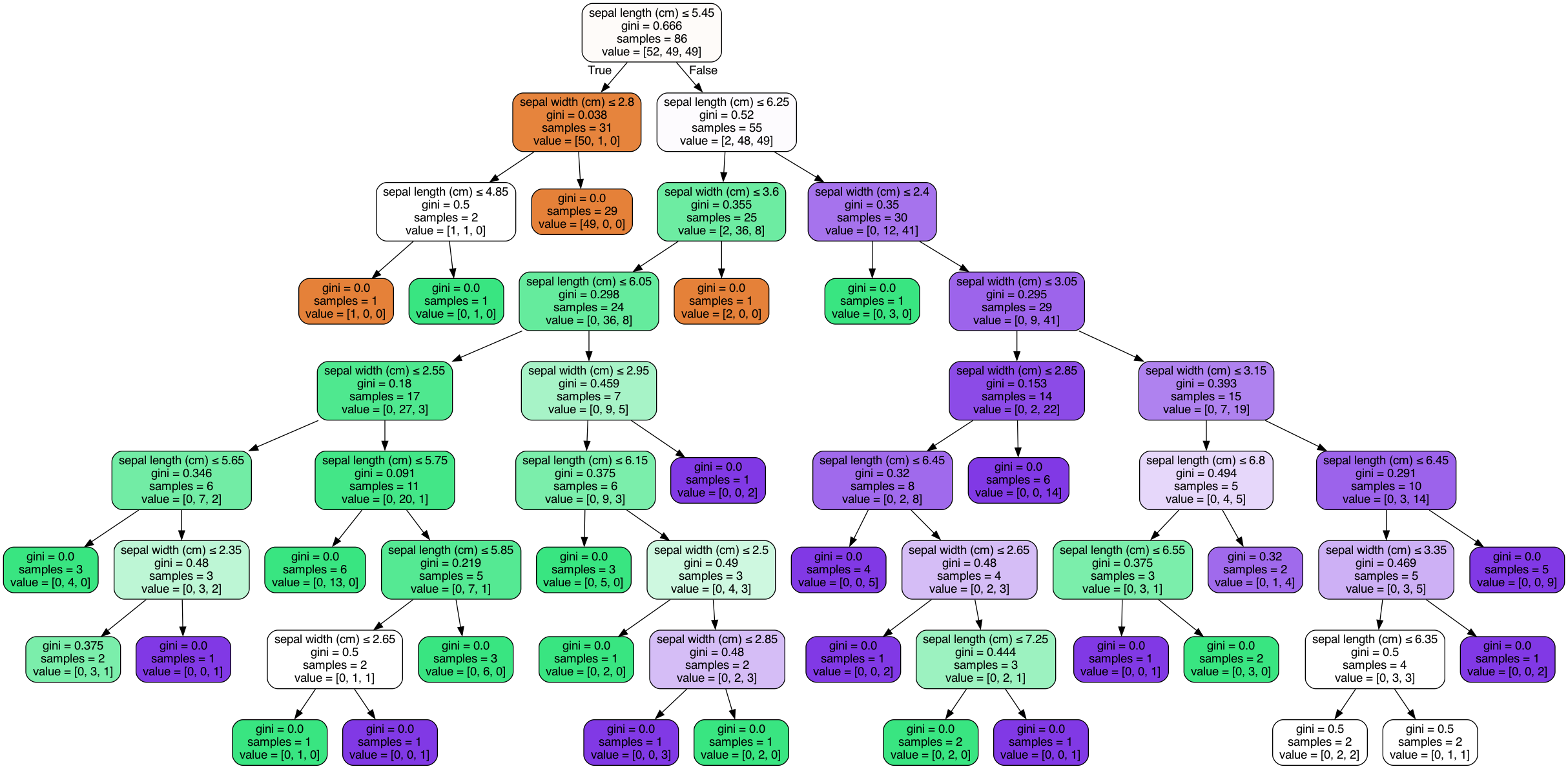
Image(get\_png\_tree(bc1.estimators\_[2], ds.feature\_names[:2]), width='80%')



Image(get\_png\_tree(bc1.estimators\_[3], ds.feature\_names[:2]), width='80%')



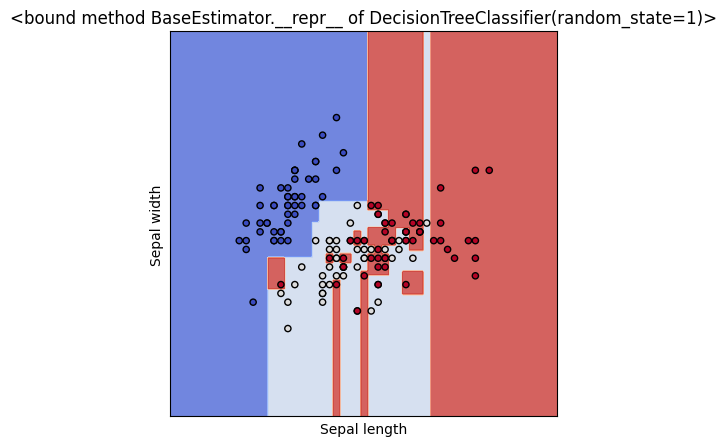
Image(get\_png\_tree(bc1.estimators\_[4], ds.feature\_names[:2]), width='80%')



Можно отметить, что деревья получаются различными. Таким образом, каждое дерево работает как "слабая модель".

### Визуализация результатов классификации

def make\_meshgrid(x, y, h=.02):  
 """Create a mesh of points to plot in  
  
 Parameters  
 ----------  
 x: data to base x-axis meshgrid on  
 y: data to base y-axis meshgrid on  
 h: stepsize for meshgrid, optional  
  
 Returns  
 -------  
 xx, yy : ndarray  
 """  
 x\_min, x\_max = x.min() - 1, x.max() + 1  
 y\_min, y\_max = y.min() - 1, y.max() + 1  
 xx, yy = np.meshgrid(np.arange(x\_min, x\_max, h),  
 np.arange(y\_min, y\_max, h))  
 return xx, yy  
def plot\_contours(ax, clf, xx, yy, \*\*params):  
 """Plot the decision boundaries for a classifier.  
  
 Parameters  
 ----------  
 ax: matplotlib axes object  
 clf: a classifier  
 xx: meshgrid ndarray  
 yy: meshgrid ndarray  
 params: dictionary of params to pass to contourf, optional  
 """  
 Z = clf.predict(np.c\_[xx.ravel(), yy.ravel()])  
 Z = Z.reshape(xx.shape)  
 #Можно проверить все ли метки классов предсказываются  
 #print(np.unique(Z))  
 out = ax.contourf(xx, yy, Z, \*\*params)  
 return out  
  
def plot\_cl(clf):  
 title = clf.\_\_repr\_\_  
 clf.fit(iris\_X, iris\_y)  
 fig, ax = plt.subplots(figsize=(5,5))  
 X0, X1 = iris\_X[:, 0], iris\_X[:, 1]  
 xx, yy = make\_meshgrid(X0, X1)  
 plot\_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)  
 ax.scatter(X0, X1, c=iris\_y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')  
 ax.set\_xlim(xx.min(), xx.max())  
 ax.set\_ylim(yy.min(), yy.max())  
 ax.set\_xlabel('Sepal length')  
 ax.set\_ylabel('Sepal width')  
 ax.set\_xticks(())  
 ax.set\_yticks(())  
 ax.set\_title(title)  
 plt.show()  
plot\_cl(DecisionTreeClassifier(random\_state=1))

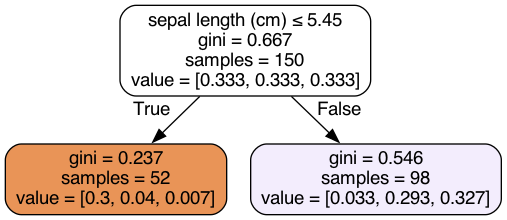


### Бустинг

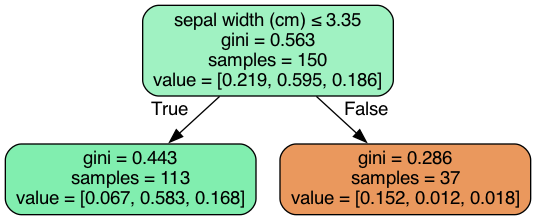
ab1 = AdaBoostClassifier(n\_estimators=5, algorithm='SAMME', random\_state=10)  
ab1.fit(iris\_X, iris\_y)

AdaBoostClassifier(algorithm='SAMME', n\_estimators=5, random\_state=10)

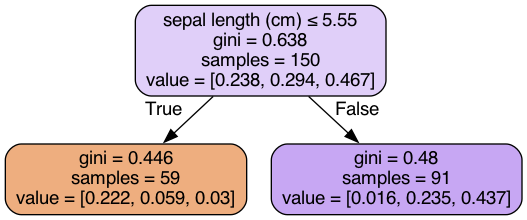
Image(get\_png\_tree(ab1.estimators\_[0], ds.feature\_names[:2]), width='40%')



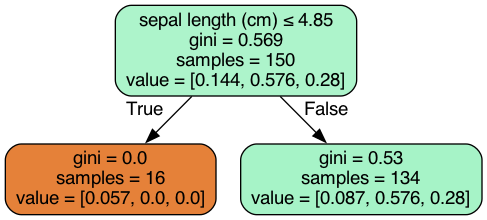
Image(get\_png\_tree(ab1.estimators\_[1], ds.feature\_names[:2]), width='40%')



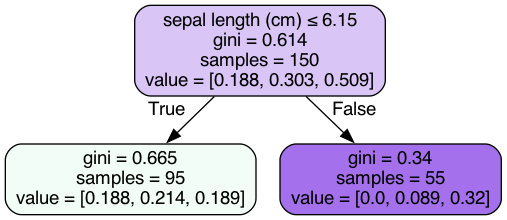
Image(get\_png\_tree(ab1.estimators\_[2], ds.feature\_names[:2]), width='40%')



Image(get\_png\_tree(ab1.estimators\_[3], ds.feature\_names[:2]), width='40%')



Image(get\_png\_tree(ab1.estimators\_[4], ds.feature\_names[:2]), width='40%')



ab1.estimator\_weights\_

array([1.21109027, 1.71357397, 1.35360531, 1.24138009, 0.83110679])

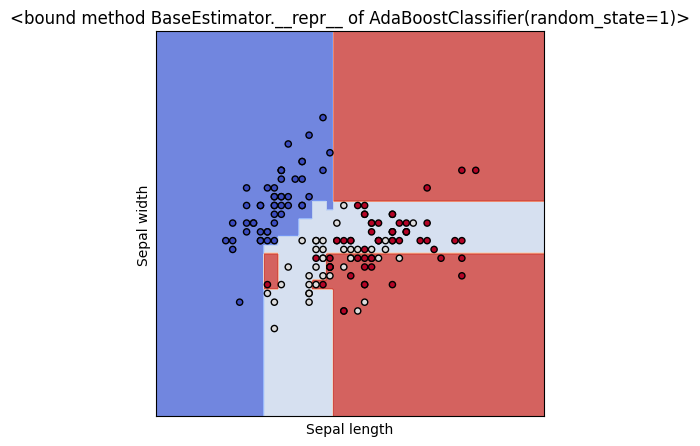
df1 = ab1.decision\_function(iris\_X)  
df1.shape

(150, 3)

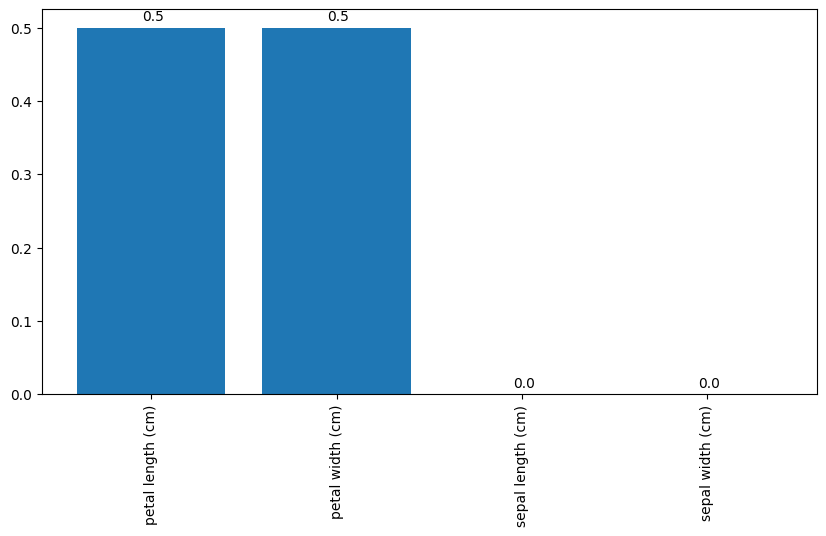
df1[:10]

array([[0.67366299, 0.32633701, 0. ],  
 [0.40384096, 0.59615904, 0. ],  
 [0.5993106 , 0.4006894 , 0. ],  
 [0.5993106 , 0.4006894 , 0. ],  
 [0.67366299, 0.32633701, 0. ],  
 [0.67366299, 0.32633701, 0. ],  
 [0.86913263, 0.13086737, 0. ],  
 [0.67366299, 0.32633701, 0. ],  
 [0.5993106 , 0.4006894 , 0. ],  
 [0.40384096, 0.59615904, 0. ]])

plot\_cl(AdaBoostClassifier(random\_state=1))



from operator import itemgetter  
  
def draw\_feature\_importances(tree\_model, X\_dataset, figsize=(10,5)):  
 """  
 Вывод важности признаков в виде графика  
 """  
 # Сортировка значений важности признаков по убыванию  
 list\_to\_sort = list(zip(X\_dataset.columns.values, tree\_model.feature\_importances\_))  
 sorted\_list = sorted(list\_to\_sort, key=itemgetter(1), reverse = True)  
 # Названия признаков  
 labels = [x for x,\_ in sorted\_list]  
 # Важности признаков  
 data = [x for \_,x in sorted\_list]  
 # Вывод графика  
 fig, ax = plt.subplots(figsize=figsize)  
 ind = np.arange(len(labels))  
 plt.bar(ind, data)  
 plt.xticks(ind, labels, rotation='vertical')  
 # Вывод значений  
 for a,b in zip(ind, data):  
 plt.text(a-0.05, b+0.01, str(round(b,3)))  
 plt.show()  
 return labels, data  
  
iris\_x\_ds = pd.DataFrame(data=ds['data'], columns=ds['feature\_names'])  
ab2 = AdaBoostClassifier(random\_state=1)  
ab2.fit(iris\_x\_ds, ds.target)  
\_,\_ = draw\_feature\_importances(ab2, iris\_x\_ds)



## Стекинг

def val\_mae(model):  
 model.fit(X\_train, y\_train)  
 y\_pred = model.predict(X\_test)  
 result = mean\_absolute\_error(y\_test, y\_pred)  
 print(model)  
 print('MAE={}'.format(result))

for model in [  
 LinearRegression(),  
 DecisionTreeRegressor(),  
 RandomForestRegressor(n\_estimators=50)  
]:  
 val\_mae(model)  
 print('==========================')  
 print()

LinearRegression()  
MAE=0.20171778209678323  
==========================  
  
DecisionTreeRegressor()  
MAE=0.08  
==========================  
  
RandomForestRegressor(n\_estimators=50)  
MAE=0.06080000000000001  
==========================

dataset = Dataset(X\_train, y\_train, X\_test)

model\_tree = Regressor(dataset=dataset, estimator=DecisionTreeRegressor, name='tree')  
model\_lr = Regressor(dataset=dataset, estimator=LinearRegression,name='lr')  
model\_rf = Regressor(dataset=dataset, estimator=RandomForestRegressor, parameters={'n\_estimators': 50},name='rf')

pipeline = ModelsPipeline(model\_tree, model\_lr)  
stack\_ds = pipeline.stack(k=10, seed=1)  
# модель второго уровня  
stacker = Regressor(dataset=stack\_ds, estimator=LinearRegression)  
results = stacker.validate(k=10,scorer=mean\_absolute\_error)

Metric: mean\_absolute\_error  
Folds accuracy: [0.17081515612463896, 0.14143284991104105, 0.10609119233791114, 0.18004801783734437, 0.07436735224932983, 0.15990310908830654, 0.1591113062757908, 0.1266532535111755, 0.11043404201598409, 0.17355974255116716]  
Mean accuracy: 0.14024160219026896  
Standard Deviation: 0.03309418713701339  
Variance: 0.0010952252222596627

stack\_ds = pipeline.stack(k=10, seed=1)  
stacker = Regressor(dataset=stack\_ds, estimator=RandomForestRegressor)  
results = stacker.validate(k=10,scorer=mean\_absolute\_error)

Metric: mean\_absolute\_error  
Folds accuracy: [0.027499999999999997, 0.10500000000000001, 0.03375, 0.12125, 0.0, 0.06571428571428571, 0.0057142857142857195, 0.031428571428571424, 0.12285714285714285, 0.14142857142857143]  
Mean accuracy: 0.06546428571428572  
Standard Deviation: 0.05020044007450004  
Variance: 0.0025200841836734694

# Эксперимент 3  
# Первый уровень - три модели: дерево, линейная регрессия и случайный лес  
# Второй уровень: линейная регрессия  
pipeline = ModelsPipeline(model\_tree, model\_lr, model\_rf)  
stack\_ds3 = pipeline.stack(k=10, seed=1)  
# модель второго уровня  
stacker = Regressor(dataset=stack\_ds3, estimator=LinearRegression)  
results = stacker.validate(k=10,scorer=mean\_absolute\_error)

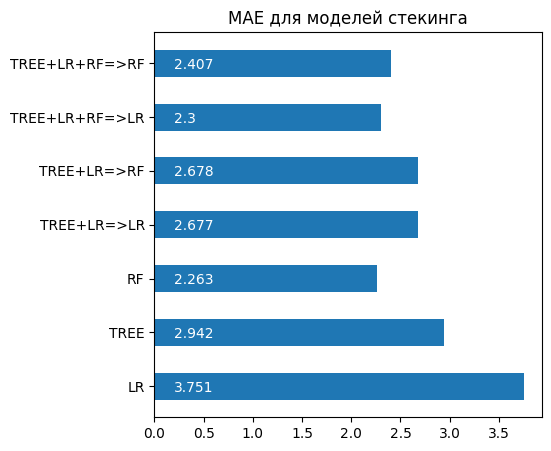
Metric: mean\_absolute\_error  
Folds accuracy: [0.0820751424773805, 0.10750820345145327, 0.11199217536353251, 0.1311720650148756, 0.0458560959310263, 0.140276979280042, 0.10823276738312201, 0.0794651161162457, 0.1561550119020172, 0.14026346863254738]  
Mean accuracy: 0.11029970255522425  
Standard Deviation: 0.03201583109487226  
Variance: 0.0010250134406953892

# Эксперимент 4  
# Первый уровень - три модели: дерево, линейная регрессия и случайный лес  
# Второй уровень: случайный лес  
# Результат хуже чем в эксперименте 3  
stacker = Regressor(dataset=stack\_ds3, estimator=RandomForestRegressor)  
results = stacker.validate(k=10,scorer=mean\_absolute\_error)

Metric: mean\_absolute\_error  
Folds accuracy: [0.012500000000000011, 0.03125, 0.08875, 0.11499999999999999, 0.0, 0.06857142857142857, 0.028571428571428564, 0.05714285714285713, 0.08, 0.08142857142857143]  
Mean accuracy: 0.056321428571428564  
Standard Deviation: 0.035110721369718975  
Variance: 0.0012327627551020407

# Результаты  
array\_labels = ['LR','TREE', 'RF', 'TREE+LR=>LR',   
 'TREE+LR=>RF', 'TREE+LR+RF=>LR', 'TREE+LR+RF=>RF']  
array\_mae = [3.7507121808389168, 2.942156862745098, 2.263039215686275,   
 2.6766504031924305, 2.6775473780487804, 2.2998386142710823,   
 2.406510426829268]

def vis\_models\_quality(array\_metric, array\_labels, str\_header, figsize=(5, 5)):  
 fig, ax1 = plt.subplots(figsize=figsize)  
 pos = np.arange(len(array\_metric))  
 rects = ax1.barh(pos, array\_metric,  
 align='center',  
 height=0.5,   
 tick\_label=array\_labels)  
 ax1.set\_title(str\_header)  
 for a,b in zip(pos, array\_metric):  
 plt.text(0.2, a-0.1, str(round(b,3)), color='white')  
 plt.show()  
vis\_models\_quality(array\_mae, array\_labels, 'MAE для моделей стекинга')



## Метрики

lab = preprocessing.LabelEncoder()  
cl1\_1 = KNeighborsClassifier(n\_neighbors=10)  
y\_transformed = lab.fit\_transform(y\_train)  
cl1\_1.fit(X\_train, y\_transformed)  
target1\_2 = cl1\_1.predict(X\_test)  
len(target1\_2), target1\_2

(75,  
 array([0, 1, 1, 0, 2, 1, 2, 0, 0, 2, 1, 0, 2, 1, 1, 0, 1, 1, 0, 0, 1, 1,  
 2, 0, 2, 1, 0, 0, 1, 2, 1, 2, 1, 2, 2, 0, 1, 0, 1, 2, 2, 0, 1, 2,  
 1, 2, 0, 0, 0, 1, 0, 0, 2, 2, 2, 2, 2, 1, 2, 1, 0, 2, 2, 0, 0, 2,  
 0, 2, 2, 1, 1, 2, 2, 0, 1]))

# 10 ближайших соседей  
accuracy\_score(y\_test, target1\_2)

0.9733333333333334