|  |  |
| --- | --- |
| Изображение выглядит как эмблема, герб, нашивка, символ  Автоматически созданное описание | **Министерство науки и высшего образования Российской Федерации**  **Федеральное государственное автономное образовательное учреждение**  **высшего образования**  **«Московский государственный технический университет**  **имени Н.Э. Баумана**  **(национальный исследовательский университет)»**  **(МГТУ им. Н.Э. Баумана)** |

|  |  |
| --- | --- |
| ФАКУЛЬТЕТ | ГОЛОВНОЙ УЧЕБНО-ИССЛЕДОВАТЕЛЬСКИЙ И МЕТОДИЧЕСКИЙ ЦЕНТР |
| ПРОФЕССИОНАЛЬНОЙ РЕАБИЛИТАЦИИ ЛИЦ С ОГРАНИЧЕННЫМИ |
| ВОЗМОЖНОСТЯМИ ЗДОРОВЬЯ |
| КАФЕДРА | СИСТЕМЫ ОБРАБОТКИ ИНФОРМАЦИИ И УПРАВЛЕНИЯ |

Отчёт по лабораторной работе №5 по курсу «Технологии машинного обучения».

«Ансамбли моделей машинного обучения».

Выполнил: Проверил:

Новиков С. А. Гапанюк Ю.Е.

студент группы ИУ5-62Б

Подпись и дата: Подпись и дата:

*2025 г.*

# Задание лабораторной работы

* Выбрать набор данных (датасет) для решения задачи классификации или регресии.
* В случае необходимости произвести удаление или заполнение пропусков и кодирование категориальных признаков.
* С использованием метода train\_test\_split разделить выборку на обучающую и тестовую.
* Обучить следующие ансамблевые модели: одну из моделей группы бэггинга (бэггинг или случайный лес или сверхслучайные деревья); одну из моделей группы бустинга; одну из моделей группы стекинга.
* Дополнительно к указанным моделям обучить еще две модели: модель многослойного персептрона; модель МГУА.
* Оценить качество моделей с помощью одной из подходящих для задачи метрик. Сравнить качество полученных моделей.

# Ячейки Jupyter-ноутбука

## Выбор и загрузка данных

В качестве датасета будем использовать набор данных, содержащий данные о различных стёклах. Данный набор доступен по адресу: [https://www.kaggle.com/datasets/uciml/glass](http://www.kaggle.com/datasets/uciml/glass)

Набор данных имеет следующие атрибуты:

* RI - Refractive Index - коэффициент преломления
* Na - Sodium - Содержание натрия (массовый процент в соответствующем оксиде)
* Mg - Magnesium - Содержание магния
* Al - Aluminum - Содержание алюминия
* Si - Silicon - Содержание кремния
* K - Potassium - Содержание калия
* Ca - Calcium - Содержание кальция
* Ba - Barium - Содеражние бария
* Fe - Iron - Содержание железа
* Type - Type of glass - тип стекла (1, 2 - стекла для зданий, 3, 4 - стекла для автомобилей, 5 - стеклотара, 6 - бытовые стекла, 7 - стекла для ламп; 4 отсутствует в данном наборе данных)

[1]:

### Импорт библиотек

Импортируем библиотеки с помощью команды import:

**import numpy as np import pandas as pd import seaborn as sns**

**import matplotlib.pyplot as plt**

%**matplotlib** inline sns.set(style=DticksD)

Уберем предупреждения:

[2]:

**import warnings**

warnings.filterwarnings('ignore')

### Загрузка данных

Загрузим набор данных:

[3]:

data = pd.read\_csv('glass.csv')

[4]:

## Первичный анализ и обработка данных

Выведем первые 5 строк датасета:

data.head()

|  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [4]: |  | RI | Na | Mg | Al | Si | K | Ca | Ba | Fe | Type |
|  | 0 | 1.52101 | 13.64 | 4.49 | 1.10 | 71.78 | 0.06 | 8.75 | 0.0 | 0.0 | 1 |
|  | 1 | 1.51761 | 13.89 | 3.60 | 1.36 | 72.73 | 0.48 | 7.83 | 0.0 | 0.0 | 1 |
|  | 2 | 1.51618 | 13.53 | 3.55 | 1.54 | 72.99 | 0.39 | 7.78 | 0.0 | 0.0 | 1 |
|  | 3 | 1.51766 | 13.21 | 3.69 | 1.29 | 72.61 | 0.57 | 8.22 | 0.0 | 0.0 | 1 |
|  | 4 | 1.51742 | 13.27 | 3.62 | 1.24 | 73.08 | 0.55 | 8.07 | 0.0 | 0.0 | 1 |

Определим размер датасета:

[5]:

data.shape

[5]: (214, 10)

[6]:

### Обработка данных

Проверим наличие пропусков:

data.isnull().sum()

|  |  |  |
| --- | --- | --- |
| [6]: | RI | 0 |
|  | Na | 0 |
|  | Mg | 0 |
|  | Al | 0 |
|  | Si | 0 |
|  | K | 0 |
|  | Ca | 0 |
|  | Ba | 0 |
|  | Fe | 0 |
|  | Type | 0 |
|  | dtype: | int64 |

В датасете не наблюдаются пропуски. Определим типы данных:

[7]:

data.dtypes

1. : RI float64

Na float64

Mg float64

Al float64

Si float64

K float64

Ca float64

Ba float64

Fe float64 Type int64 dtype: object

Кодирование категориальных признаков не потребуется.

1. :

### Разделение данных

Разделим данные на столбец с целевым признаком и данные с другими столбцами:

X = data.drop(DTypeD, axis=1) y = data[DTypeD]

1. :

print(X.head(), D**\n**D) print(y.head())

RI Na Mg Al Si K Ca Ba Fe

0 1.52101 13.64 4.49 1.10 71.78 0.06 8.75 0.0 0.0

1 1.51761 13.89 3.60 1.36 72.73 0.48 7.83 0.0 0.0

2 1.51618 13.53 3.55 1.54 72.99 0.39 7.78 0.0 0.0

3 1.51766 13.21 3.69 1.29 72.61 0.57 8.22 0.0 0.0

4 1.51742 13.27 3.62 1.24 73.08 0.55 8.07 0.0 0.0

|  |  |
| --- | --- |
| 0 | 1 |
| 1 | 1 |
| 2 | 1 |
| 3 | 1 |
| 4 | 1 |

Name: Type, dtype: int64

1. :

print(X.shape) print(y.shape)

(214, 9)

(214,)

1. :

### Разделение выборки на обучающую и тестовую

Будем решать задачу регрессии - отображения новых предсказанных записей. Для этого разделим выборку с помощью функции train\_test\_split:

**from sklearn.model\_selection import** train\_test\_split

1. :

X\_train, X\_test, y\_train, y\_test = train\_test\_split(X, y, random\_state=1)

1. :

Размеры обучающей выборки и тестовой выборки:

X\_train.shape, y\_train.shape, X\_test.shape, y\_test.shape

1. : ((160, 9), (160,), (54, 9), (54,))

## Обучение ансамблевых моделей

### Модель бэггинга

**from sklearn.ensemble import** BaggingRegressor

1. :
2. :

Обучим модель на 5 деревьях:

bagging\_model = BaggingRegressor(n\_estimators=5, oob\_score=**True**, random\_state=10) bagging\_model.fit(X\_train, y\_train)

1. : BaggingRegressor(n\_estimators=5, oob\_score=True, random\_state=10)

Сконвертируем объекты, которые были использованы в обучающей выборке каждого дере- ва, в двоичную матрицу (1 соответствует элементам, попавшим в обучающую выборку):

1. :

bin\_array = np.zeros((5, X\_train.shape[0]))

**for** i **in** range(5):

**for** j **in** bagging\_model.estimators\_samples\_[i]: bin\_array[i][j] = 1

bin\_array

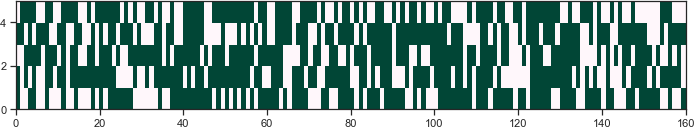
|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| [16]: | array([[1., | 0., | 0., | 1., | 1., | 0., | 0., | 1., | 0., | 0., | 0., | 1., | 0., | 0., | 1., | 0., |
|  | 0., | 0., | 1., | 0., | 1., | 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., | 1., | 1., | 1., | 0., | 0., | 1., | 0., | 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., | 1., | 0., | 1., | 0., | 1., | 1., | 1., | 1., | 1., | 0.], |
|  | [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., | 0., | 1., | 0., | 1., | 0.], |
|  | [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., | 1., | 0., | 0., | 0., | 1., | 0., | 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., | 1., | 1., | 1., | 1., | 1., | 1., | 1., | 0., | 1., | 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., | 1., | 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., | 0., | 0., | 0., | 0., | 1., | 1., | 1., | 0., | 0., | 0.]]) |

Визуализируем эти данные:

1. :

fig, ax = plt.subplots(figsize=(12,2)) ax.pcolor(bin\_array, cmap='PuBuGn') plt.show()



1. :



**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))

Оценим Out-of-bag error (теоретическое значение = 37%) - несмещенную оценку ошибки набора тестов:

1. :

Для модели 1 размер OOB составляет 38.12% Для модели 2 размер OOB составляет 35.0% Для модели 3 размер OOB составляет 34.38% Для модели 4 размер OOB составляет 36.88% Для модели 5 размер OOB составляет 42.5%

Визуализируем обученные деревья:

**from io import** StringIO

**from IPython.display import** Image

**import graphviz import pydotplus**

**from sklearn.tree import** export\_graphviz

1. :



**def** get\_png\_tree(tree\_model\_param, feature\_names\_param): dot\_data = StringIO() export\_graphviz(tree\_model\_param, out\_file=dot\_data,

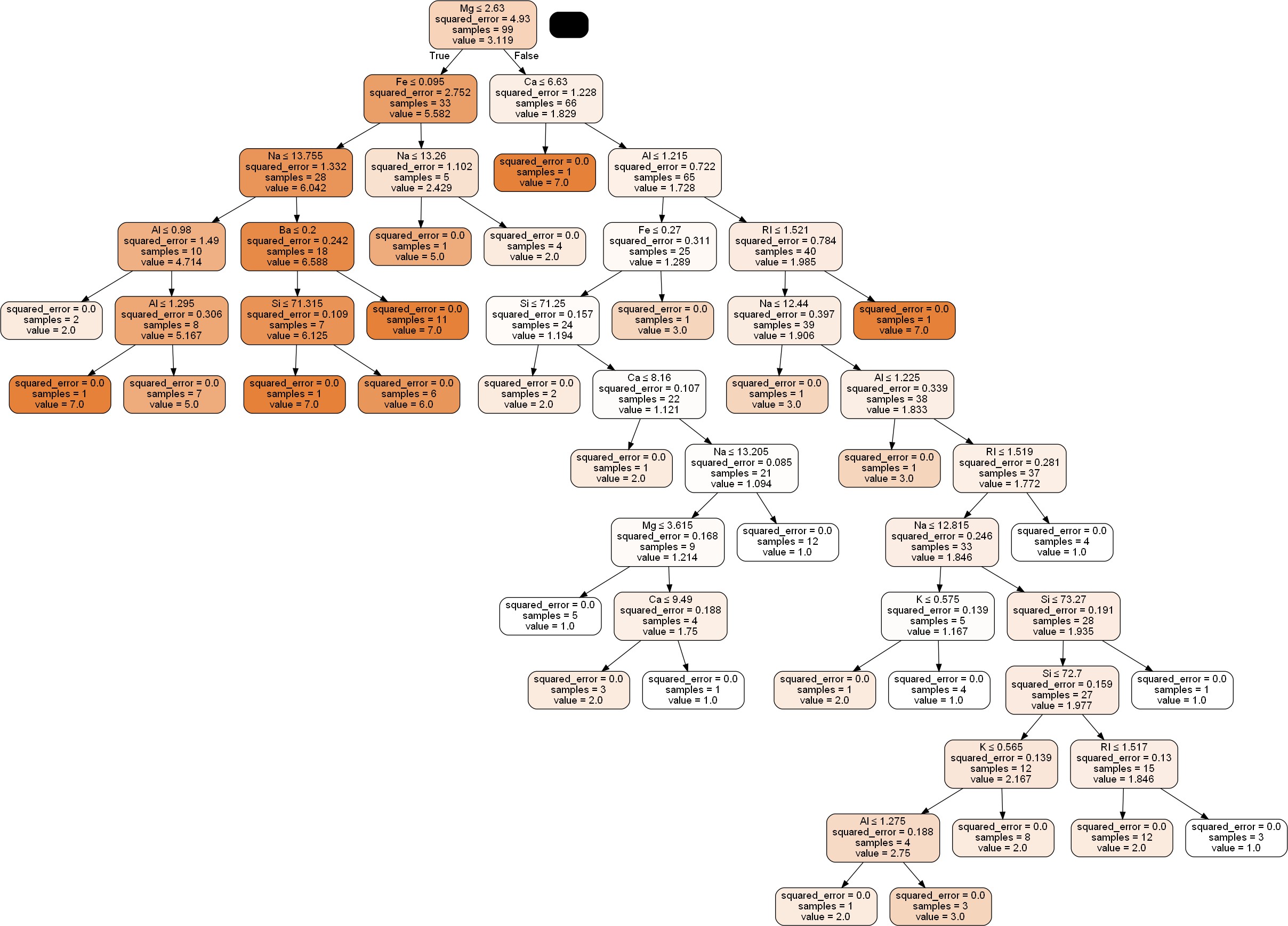
*‹→*feature\_names=feature\_names\_param,

filled=**True**, rounded=**True**, special\_characters=**True**) graph = pydotplus.graph\_from\_dot\_data(dot\_data.getvalue())

**return** graph.create\_png()

1. :

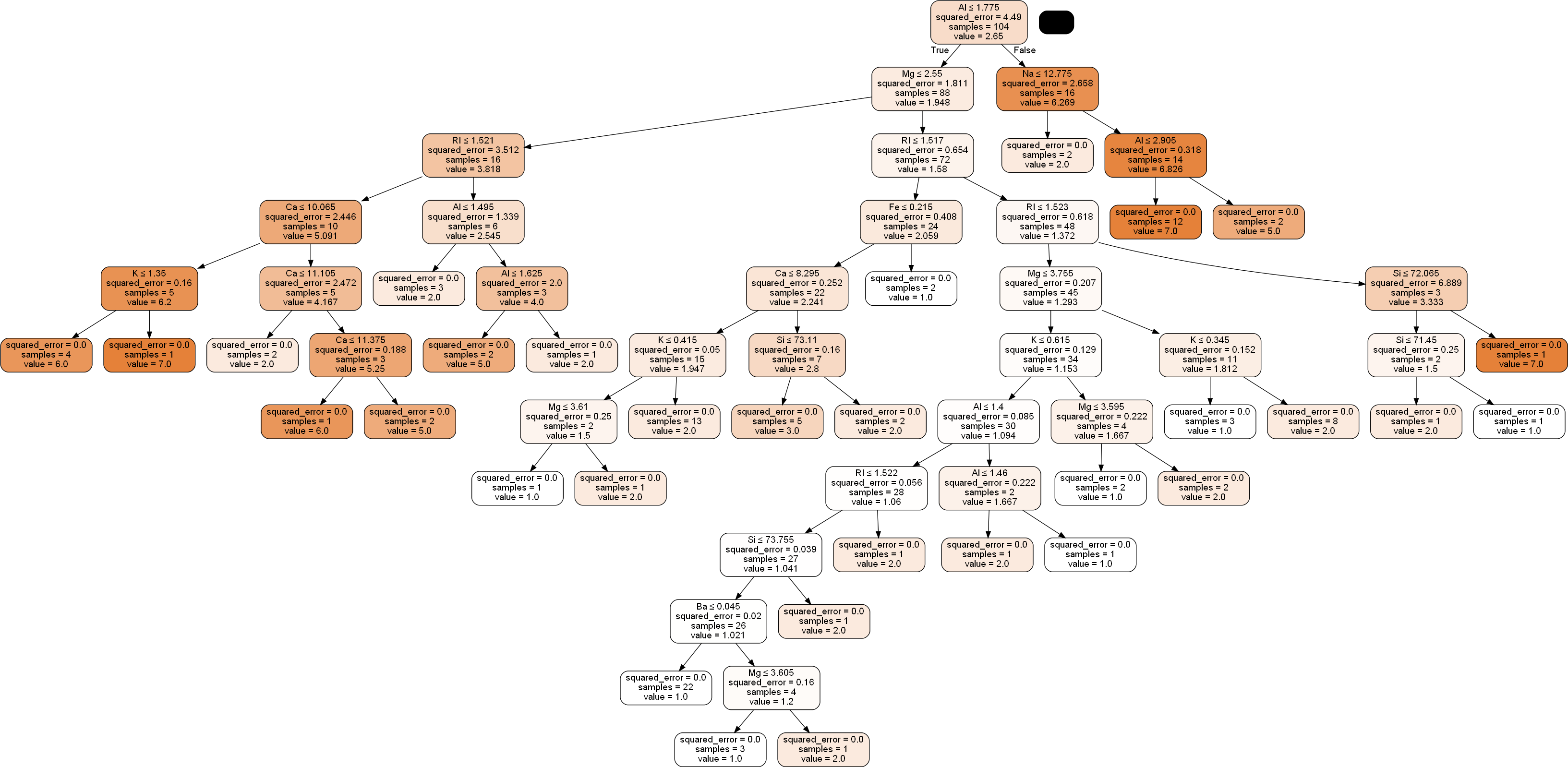
Image(get\_png\_tree(bagging\_model.estimators\_[0], X.columns))

[21]:

1. :

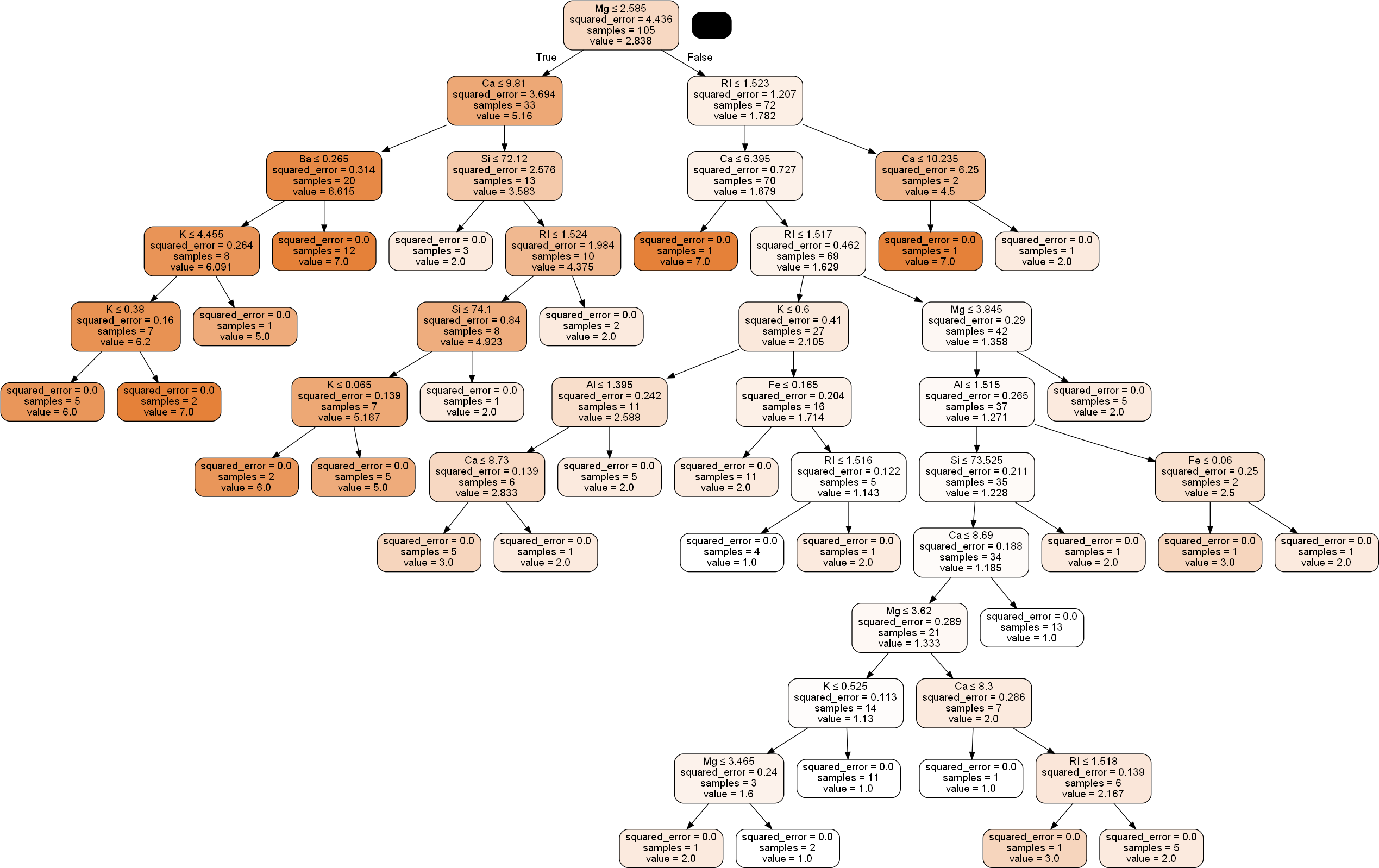
Image(get\_png\_tree(bagging\_model.estimators\_[1], X.columns))

[22]:



1. :

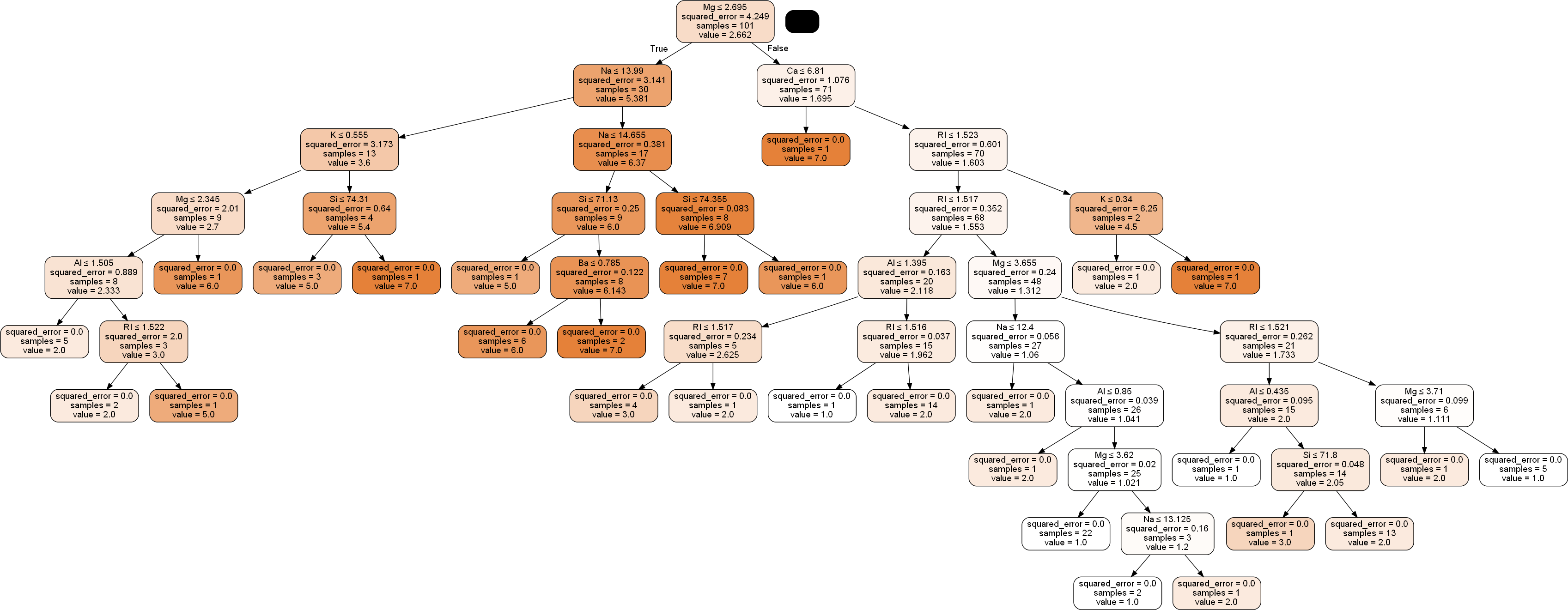
Image(get\_png\_tree(bagging\_model.estimators\_[2], X.columns))

[23]:

1. :

Image(get\_png\_tree(bagging\_model.estimators\_[3], X.columns))

[24]:

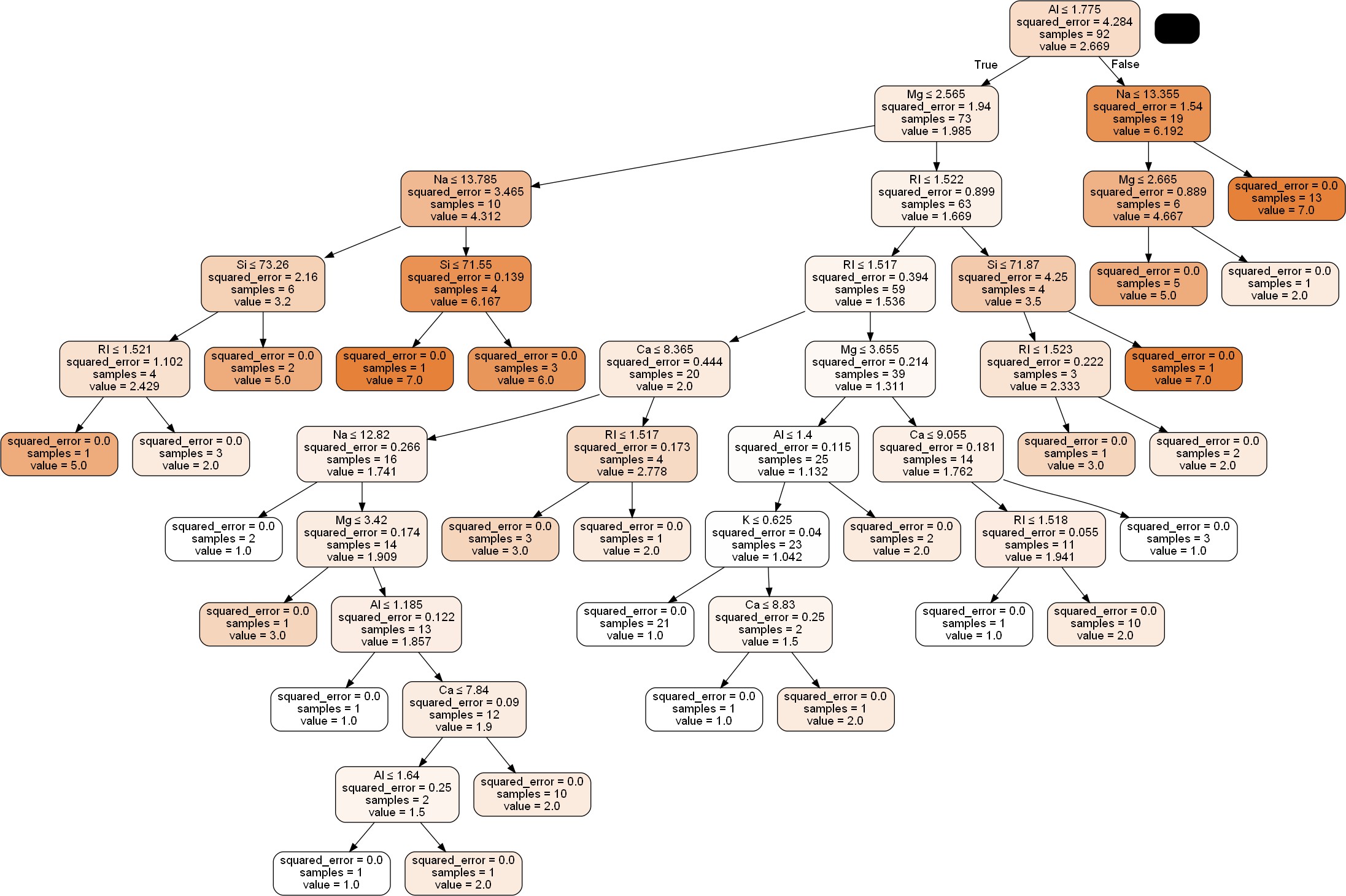


1. :

Image(get\_png\_tree(bagging\_model.estimators\_[4], X.columns))

[25]:

1. :



Заметно, что деревья различны. Визуализируем результаты регрессии:

**from sklearn.tree import** DecisionTreeRegressor

1. :

**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(X2, y)

fig, ax = plt.subplots(figsize=(5,5)) X0, X1 = X2[:, 0], X2[:, 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=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('Na') ax.set\_ylabel('Si') ax.set\_xticks(()) ax.set\_yticks(()) ax.set\_title(title)

plt.show()

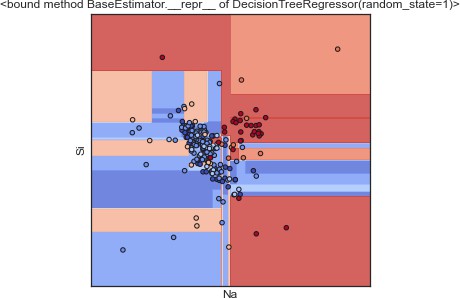
Оставим только два признака - Na и Si:

1. :

X2 = X[['Na', 'Si']].to\_numpy()

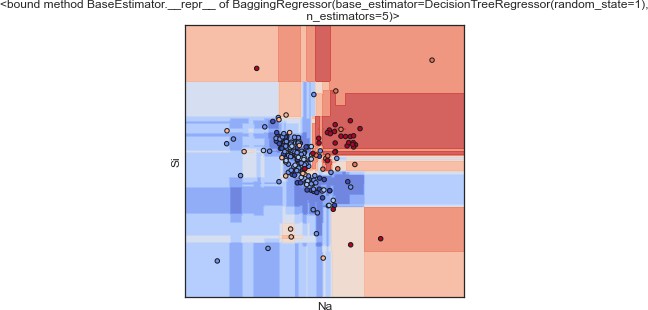
1. :

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



1. :

plot\_cl(BaggingRegressor(DecisionTreeRegressor(random\_state=1), n\_estimators=5))



### Модель градиентного бустинга

1. :

**from sklearn.ensemble import** GradientBoostingRegressor

Обучим модель на 5 деревьях:

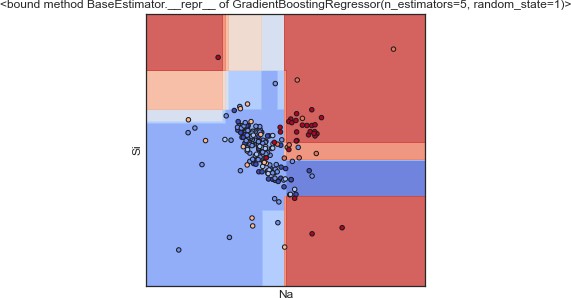
1. :

gradient\_model = GradientBoostingRegressor(n\_estimators=5) gradient\_model.fit(X\_train, y\_train)

1. : GradientBoostingRegressor(n\_estimators=5)
2. :

plot\_cl(GradientBoostingRegressor(random\_state=1, n\_estimators=5))

Для визуализации регрессии будем использовать функцию plot\_cl из визуализации регрессии модели бэггинга:



### Модель стекинга

1. :

**from sklearn.ensemble import** RandomForestRegressor

**from sklearn.linear\_model import** LinearRegression

Реализуем модель стекинга через библиотеку heamy:

1. :

**from heamy.estimator import** Regressor **from heamy.pipeline import** ModelsPipeline **from heamy.dataset import** Dataset

1. :

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

Построим модели дерева, линейную модель и случайного леса для задачи регрессии:

1. :



model\_tree = Regressor(dataset=dataset, estimator=DecisionTreeRegressor,

*‹→*name='tree')

model\_lr = Regressor(dataset=dataset, estimator=LinearRegression,

*‹→*parameters={'normalize': **True**}, name='lr')



model\_rf = Regressor(dataset=dataset, estimator=RandomForestRegressor,

*‹→*parameters={'n\_estimators': 5}, name='rf')

Определим их качество:

1. :

**from sklearn.metrics import** mean\_absolute\_error

1. :

**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(DMAE = **{}**D.format(result))

1. :

**for** model **in** [ LinearRegression(), DecisionTreeRegressor(),

RandomForestRegressor(n\_estimators=5)

]:

val\_mae(model) print()

LinearRegression()

MAE = 0.7674795585625986

DecisionTreeRegressor() MAE = 0.5740740740740741

RandomForestRegressor(n\_estimators=5) MAE = 0.7000000000000003

1. :

Сделаем несколько экспериментов для определения лучшего расположения моделей. 1: на первом уровне - дерево и линейная регрессия, а на втором - линейная

регрессия:

pipeline = ModelsPipeline(model\_tree, model\_lr) stack\_ds = pipeline.stack(k=10, seed=1)

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

Metric: mean\_absolute\_error

Folds accuracy: [0.9343038538699631, 0.6741240299143151, 1.148855880577542,

0.804952423447935, 0.7433758171159082, 0.5933363445393178, 0.841554871284919,

0.9027277159278726, 0.9792465696505968, 0.7520786269630576]

Mean accuracy: 0.8374556133291428

Standard Deviation: 0.1530358629440931

Variance: 0.02341997534704325

2: на первом уровне - дерево и стохастический градиентный спуск, а на втором - случайный лес:

1. :

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

Metric: mean\_absolute\_error

Folds accuracy: [0.7756250000000001, 0.529375, 1.2375, 0.9025000000000001,

0.80625, 0.850625, 0.8162499999999999, 0.9293750000000001, 0.77125, 0.64375]

Mean accuracy: 0.8262500000000002

Standard Deviation: 0.17711534024471173

Variance: 0.03136984375

3: на первом уровне - дерево, линейная регрессия и случайный лес, а на втором - линейная регрессия:

1. :

pipeline = ModelsPipeline(model\_tree, model\_lr, model\_rf) stack\_ds3 = pipeline.stack(k=10, seed=1)

stacker = Regressor(dataset=stack\_ds3, estimator=LinearRegression) results = stacker.validate(k=10, scorer=mean\_absolute\_error)

Metric: mean\_absolute\_error

Folds accuracy: [0.822566076331519, 0.5952680586138698, 1.1577994260154325,

0.7950018519448512, 0.648576912753356, 0.5167249218149097, 0.7253297444314115,

0.8648446064521539, 0.8336842495096648, 0.6534908584095975]

Mean accuracy: 0.7613286706276765

Standard Deviation: 0.17060739660716165

Variance: 0.02910688377707335

4: на первом уровне - дерево, линейная регрессия и случайный лес, а на втором - случайный лес:

1. :

stacker = Regressor(dataset=stack\_ds3, estimator=RandomForestRegressor) results = stacker.validate(k=10, scorer=mean\_absolute\_error)

Metric: mean\_absolute\_error

Folds accuracy: [0.865, 0.5437500000000001, 1.053125, 0.825625,

0.9287500000000001, 0.505625, 0.69875, 0.880625, 0.7306249999999999, 0.546875]

Mean accuracy: 0.757875

Standard Deviation: 0.17485511859822692

Variance: 0.0305743125

Выведем результаты:

1. :

array\_labels = ['LR','TREE', 'RF', 'TREE+LR=>LR',

'TREE+LR=>RF', 'TREE+LR+RF=>LR', 'TREE+LR+RF=>RF'] array\_mae = [0.7674795585625986, 0.5740740740740741, 0.7000000000000003,

0.8374556133291428, 0.8262500000000002, 0.7613286706276765,

0.757875]

1. :

**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)

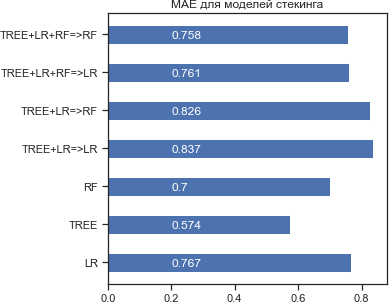
**for** a,b **in** zip(pos, array\_metric):

plt.text(0.2, a-0.1, str(round(b,3)), color='white')

plt.show()

1. :

vis\_models\_quality(array\_mae, array\_labels, 'MAE для моделей стекинга')



1. :

Чем ближе значение MAE к нулю, тем лучше качество регрессии.

Лучший результат у исходных моделей - у модели дерева, а у моделей стекинга - у эксперимента 4, где на первом уровне располагаются дерево, линейная регрессия и случайный лес, а на втором - случайный лес.

Эту модель и будем использовать для дальнейшей оценки качества:

stacking\_model = Regressor(dataset=stack\_ds3, estimator=RandomForestRegressor)

1. :

## Обучение дополнительных моделей

### Модель многослойного персептрона

Обучим модель многослойного персептрона:

**from sklearn.neural\_network import** MLPRegressor

1. :

perceptron\_model = MLPRegressor(solver='lbfgs', alpha=1e-5, hidden\_layer\_sizes=(5, 2), random\_state=1)

perceptron\_model.fit(X\_train, y\_train)

1. : MLPRegressor(alpha=1e-05, hidden\_layer\_sizes=(5, 2), random\_state=1, solver='lbfgs')
2. :

### Модель МГУА

Также обучим модель МГУА с помощью библиотеки GmdhPy:

**from gmdhpy.gmdh import** Regressor

1. :

multilayered\_model = Regressor() multilayered\_model.fit(X\_train.values, y\_train.values)

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| train | layer0 | in | 0.03 | sec |
| train | layer1 | in | 0.11 | sec |
| train | layer2 | in | 0.10 | sec |
| train | layer3 | in | 0.17 | sec |
| train | layer4 | in | 0.13 | sec |
| train | layer5 | in | 0.12 | sec |
| train | layer6 | in | 0.12 | sec |
| train | layer7 | in | 0.11 | sec |
| train | layer8 | in | 0.09 | sec |
| train | layer9 | in | 0.11 | sec |
| train | layer10 | in | 0.10 | sec |
| train | layer11 | in | 0.10 | sec |
| train | layer12 | in | 0.11 | sec |
| train | layer13 | in | 0.10 | sec |
| train | layer14 | in | 0.10 | sec |
| train | layer15 | in | 0.10 | sec |

1. : <gmdhpy.gmdh.Regressor at 0x1f19be363a0>
2. :

## Оценка качества полученных моделей

Для оценки качества полученных моделей будем использовать метрику ``Средняя абсолютная ошибка'' (mean\_absolute\_error).

Чем ближе её значение к нулю, тем лучше качество регрессии. Посчитаем метрику для всех моделей:

mean\_absolute\_error(y\_test, bagging\_model.predict(X\_test))

[53]: 0.6074074074074075

1. :

mean\_absolute\_error(y\_test, gradient\_model.predict(X\_test))

[54]: 1.251807776235007

1. :

results = stacking\_model.validate(k=10, scorer=mean\_absolute\_error)

Metric: mean\_absolute\_error

Folds accuracy: [0.865, 0.5437500000000001, 1.053125, 0.825625,

0.9287500000000001, 0.505625, 0.69875, 0.880625, 0.7306249999999999, 0.546875]

Mean accuracy: 0.757875

Standard Deviation: 0.17485511859822692

Variance: 0.0305743125

1. :

mean\_absolute\_error(y\_test, perceptron\_model.predict(X\_test))

[56]: 1.6462962962963001

1. :

mean\_absolute\_error(y\_test, multilayered\_model.predict(X\_test.values))

[57]: 0.7043107407355916

1. :

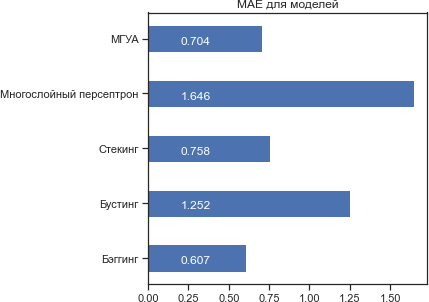
labels = ['Бэггинг','Бустинг', 'Стекинг', 'Многослойный персептрон', 'МГУА']

mae = [0.6074074074074075, 1.251807776235007, 0.757875,

1.6462962962963001, 0.7043107407355916]

1. :

vis\_models\_quality(mae, labels, 'MAE для моделей')



Самое лучшее качество регресии наблюдается у модели бэггинга (минимальное число - 0.607), а самое худшее качество - у модели многослойного персептрона (1.646).

Результаты качества регресии у моделей МГУА и стекинга сравнимы (0.704 и 0.758).