In [1]:

```
{
  "authors": [
    {
      "name": "Алексеев Андрей Сергеевич"
    }
  ],
  "group": "ИУ5-62Б",
  "kernelspec": {
    "name": "python3",
    "display_name": "Python 3 (ipykernel)",
    "language": "python"
  },
  "language_info": {
    "name": "python",
    "version": "3.9.7",
    "mimetype": "text/x-python",
    "codemirror_mode": {
      "name": "ipython",
      "version": 3
    },
    "pygments_lexer": "ipython3",
    "nbconvert_exporter": "python",
    "file_extension": ".py"
  },
  "title": "Ансамбли моделей машинного обучения"
}
```

Out[1]:

```
{'authors': [{'name': 'Алексеев Андрей Сергеевич'}],
 'group': 'ИУ5-62Б',
 'kernelspec': {'name': 'python3',
  'display_name': 'Python 3 (ipykernel)',
  'language': 'python'},
 'language_info': {'name': 'python',
  'version': '3.9.7',
  'mimetype': 'text/x-python',
  'codemirror_mode': {'name': 'ipython', 'version': 3},
  'pygments_lexer': 'ipython3',
  'nbconvert_exporter': 'python',
  'file_extension': '.py'},
 'title': 'Ансамбли моделей машинного обучения'}
```

In [1]:

```
import sys
sys.path.append(r"C:\Users\Admин")
sys.path.append(r"c:\users\admин\appdata\local\packages\pythonsoftwarefo
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from IPython.display import Image
from io import StringIO
from IPython.display import Image
import pydotplus
from sklearn.datasets import load_iris, load_boston
from sklearn.linear_model import LinearRegression
from sklearn.model_selection import cross_val_score
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.model_selection import GridSearchCV, RandomizedSearchCV
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, cla
```

```
from sklearn.metrics import confusion_matrix
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, 
from sklearn.ensemble import RandomForestClassifier, RandomForestRegress
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostin
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import AdaBoostClassifier
from sklearn.metrics import mean_absolute_error, mean_squared_error, mea
from sklearn.metrics import roc_curve, roc_auc_score
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

In [2]:

```
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_
    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
```

In [3]:

```
# Визуализация дерева
def get_png_tree(tree_model_param, feature_names_param):
    dot_data = StringIO()
    export_graphviz(tree_model_param, out_file=dot_data, feature_names=f
                    filled=True, rounded=True, special_characters=True)
    graph = pydotplus.graph_from_dot_data(dot_data.getvalue())
    return graph.create_png()
```

In [4]:

```
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
```

```
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res

def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))
```

In [5]:

```
# Сформируем DataFrame
data = pd.read_csv('WIKI_PRICES3.csv', sep=",")
def regr_to_class(close: float) -> str:
    if close<43:
        result = 1
    elif close<87:
        result = 2
    else:
        result = 3
    return result
```

In [6]:

```
# формирование второго целевого признака для классификации
data['y_clas'] = \
data.apply(lambda row: regr_to_class(row['close']),axis=1)
data1 = {'a': [], 'b': []}
data2 = {'c': []}
df = pd.DataFrame(data1)
df1 = pd.DataFrame(data2)
iris = load_iris()
df['a'] = data['high']
df['b'] = data['low']
df1['c'] = data['y_clas']
df = df.astype({'a': float, 'b': float})
df1 = df1.astype({'c': np.int8})
X = df.to_numpy()
y = df1.iloc[:, 0].tolist()
data
```

Out[6]:

	ticker	date	open	high	low	close	volume	ex-dividend	split_ratio	adj_open

0	A	1999-11-18	45.50	50.00	40.00	44.00	44739900.0	0.0	1.0	31.041951
1	A	1999-11-19	42.94	43.00	39.81	40.38	10897100.0	0.0	1.0	29.295415
2	A	1999-11-22	41.31	44.00	40.06	44.00	4705200.0	0.0	1.0	28.183363
3	A	1999-11-23	42.50	43.63	40.25	40.25	4274400.0	0.0	1.0	28.995229
4	A	1999-11-24	40.13	41.94	40.00	41.06	3464400.0	0.0	1.0	27.378319
...	...	...	...	...	...	...	...	...	...	...
9994	AAN	1986-11-20	14.00	14.50	14.00	14.50	66193.0	0.0	1.0	1.928298
9995	AAN	1986-11-21	14.50	14.50	14.00	14.00	50993.0	0.0	1.0	1.997166
9996	AAN	1986-11-24	14.25	15.00	14.25	14.87	73793.0	0.0	1.0	1.962732
9997	AAN	1986-11-25	14.87	15.25	14.50	14.50	511600.0	0.0	1.0	2.048128
9998	AAN	1986-11-26	14.50	15.25	14.50	14.75	30593.0	0.0	1.0	1.997166

9999 rows × 15 columns

In [7]:

```
# Разделение выборки на обучающую и тестовую
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    train_size=0.5,
                                                    random_state=1)
```

In [8]:

```
print(sys.path)
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(X_train, y_train)
    fig, ax = plt.subplots(figsize=(5,5))
    X0, X1 = X_train[:, 0], X_train[:, 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_train, cmap=plt.cm.coolwarm, s=20, edgecolors
    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()
```

```
['C:\\Users\\Admин', 'C:\\Users\\Admин\\AppData\\Local\\Programs\\Python
\\Python39\\python39.zip', 'C:\\Users\\Admин\\AppData\\Local\\Programs\\P
ython\\Python39\\DLLs', 'C:\\Users\\Admин\\AppData\\Local\\Programs\\Pyth
on\\Python39\\lib', 'C:\\Users\\Admин\\AppData\\Local\\Programs\\Python
\\Python39', '', 'C:\\Users\\Admин\\AppData\\Roaming\\Python\\Python39\\s
ite-packages', 'C:\\Users\\Admин\\AppData\\Local\\Programs\\Python\\Pytho
n39\\lib\\site-packages', 'C:\\Users\\Admин\\AppData\\Local\\Programs\\Py
thon\\Python39\\lib\\site-packages\\win32', 'C:\\Users\\Admин\\AppData\\L
ocal\\Programs\\Python\\Python39\\lib\\site-packages\\win32\\lib', 'C:\\U
sers\\Admин\\AppData\\Local\\Programs\\Python\\Python39\\lib\\site-packag
es\\Pythonwin', 'C:\\Users\\Admин\\AppData\\Local\\Programs\\Python\\Pyth
on39\\lib\\site-packages\\IPython\\extensions', 'C:\\Users\\Admин\\.ipyth
on', 'C:\\Users\\Admин', 'c:\\users\\admин\\appdata\\local\\packages\\pyt
honsoftwarefoundation.python.3.9_qbz5n2kfra8p0\\localcache\\local-package
s\\python39\\site-packages\\graphviz']
```

In [9]:

```
# Обучим классификатор на 5 деревьях
bc1 = BaggingClassifier(n_estimators=5, oob_score=True, random_state=10)
bc1.fit(X, y)
```

```
C:\Users\Admин\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_bagging.py:706: UserWarning: Some inputs do not have OOB
scores. This probably means too few estimators were used to compute any r
eliable oob estimates.
  warn(
C:\Users\Admин\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_bagging.py:712: RuntimeWarning: invalid value encountere
d in true_divide
  oob_decision_function = predictions / predictions.sum(axis=1)[:, np.new
axis]
```

Out[9]:

```
BaggingClassifier(n_estimators=5, oob_score=True, random_state=10)
```

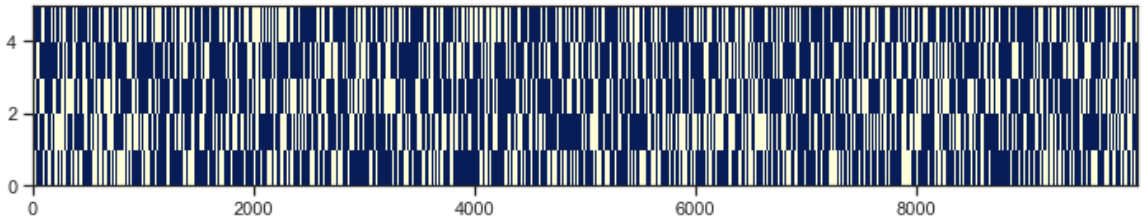
In [10]:
```
# Какие объекты были использованы в обучающей выборке каждого дерева
bc1.estimators_samples_
```

Out[10]:
```
[array([1445, 7817, 3508, ..., 5130, 6658, 8787]),
 array([7775, 8151,  814, ..., 3129,  644, 4293]),
 array([9625, 2043,  643, ...,  622, 9776, 5808]),
 array([5224, 5086, 8463, ..., 6812, 3037, 3489]),
 array([4752, 1806, 6065, ...,  306, 6699, 7347])]
```

In [11]:
```
# Сконвертируем эти данные в двоичную матрицу,
# 1 соответствует элементам, попавшим в обучающую выборку
bin_array = np.zeros((5, X.shape[0]))
for i in range(5):
    for j in bc1.estimators_samples_[i]:
        bin_array[i][j] = 1
bin_array
```

Out[11]:
```
array([[1., 0., 0., ..., 1., 1., 1.],
       [0., 1., 0., ..., 0., 1., 1.],
       [1., 1., 1., ..., 1., 0., 1.],
       [1., 1., 1., ..., 0., 1., 1.],
       [1., 1., 1., ..., 1., 1., 0.]])
```

In [12]:
```
# И визуализируем (синим цветом показаны данные, которые попали в обучающ
fig, ax = plt.subplots(figsize=(12,2))
ax.pcolor(bin_array, cmap='YlGnBu')
plt.show()
```



In [13]:
```
# 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(
```

```
Для модели № 1 размер OOB составляет 36.6%
Для модели № 2 размер OOB составляет 36.370000000000005%
Для модели № 3 размер OOB составляет 37.13%
Для модели № 4 размер OOB составляет 36.59%
Для модели № 5 размер OOB составляет 36.33%
```

In [14]:
```
# Out-of-bag error, возвращаемый классификатором
# Для классификации используется метрика accuracy
bc1.oob_score_, 1-bc1.oob_score_
```

Out[14]:
```
(0.9625962596259626, 0.037403740374037375)
```

In [15]:
```
# 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]
```

Out[15]:
```
array([[0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 1., 0.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.],
       [0., 0., 1.]])
```

In [16]:
```
# Обучим классификатор на 5 деревьях
tree1 = RandomForestClassifier(n_estimators=5, oob_score=True, random_st
tree1.fit(X_train, y_train)
```

```
C:\Users\Admин\AppData\Local\Programs\Python\Python39\lib\site-packages\s
klearn\ensemble\_forest.py:560: UserWarning: Some inputs do not have OOB
scores. This probably means too few trees were used to compute any reliab
le OOB estimates.
  warn(
```

Out[16]:
```
RandomForestClassifier(n_estimators=5, oob_score=True, random_state=10)
```

In [17]:
```
# Обучим классификатор на 5 деревьях
ab1 = AdaBoostClassifier(n_estimators=5, algorithm='SAMME', random_state
ab1.fit(X, y)
```

Out[17]:
```
AdaBoostClassifier(algorithm='SAMME', n_estimators=5, random_state=10)
```

In [18]:
```
ab1.estimator_weights_
```

Out[18]:
```
array([3.08223757, 2.68679548, 2.86171071, 2.61275355, 2.80674742])
```

In [19]:
```
df1 = ab1.decision_function(X)
df1.shape
```

Out[19]:
```
(9999, 3)
```

In [20]:
```
df1[:10]
```

Out[20]:
```
array([[0.6103652 , 0.3896348 , 0.        ],
       [0.79632307, 0.20367693, 0.        ],
       [0.6103652 , 0.3896348 , 0.        ],
       [0.6103652 , 0.3896348 , 0.        ],
       [0.79632307, 0.20367693, 0.        ],
       [0.79632307, 0.20367693, 0.        ],
```

```
       [0.79632307, 0.20367693, 0.        ],
       [0.79632307, 0.20367693, 0.        ],
       [0.6103652 , 0.3896348 , 0.        ],
       [0.39099268, 0.60900732, 0.        ]])
```

In [21]:

```
def vis_models_quality(array_metric, array_labels, str_header, figsize=(
    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()
```

In [22]:

```
sys.path.append(r"c:\users\admин\appdata\local\packages\pythonsoftwarefo
from heamy.estimator import Regressor, Classifier
from heamy.pipeline import ModelsPipeline
from heamy.dataset import Dataset
```

In [23]:

```
# Качество отдельных моделей
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))
```

In [24]:

```
# Точность на отдельных моделях
for model in [
    LinearRegression(),
    DecisionTreeRegressor(),
    RandomForestRegressor(n_estimators=50)
]:
    val_mae(model)
    print('==========================')
    print()
```

```
LinearRegression()
MAE=0.4246609360197598
==========================

DecisionTreeRegressor()
MAE=0.0086
==========================

RandomForestRegressor(n_estimators=50)
MAE=0.009647999999999999
==========================
```

In [25]:

```
# Используем библиотеку heamy
# набор данных
dataset = Dataset(X_train, y_train, X_test)

# модели первого уровня
```

```
model_tree = Regressor(dataset=dataset, estimator=DecisionTreeRegressor,
model_lr = Regressor(dataset=dataset, estimator=LinearRegression, parame
model_rf = Regressor(dataset=dataset, estimator=RandomForestRegressor, pa
```

In [26]:

```
# Эксперимент 1
# Первый уровень - две модели: дерево и линейная регрессия
# Второй уровень: линейная регрессия

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: [0.010576403829468952, 0.01555720767808864, 0.02265507090
643779, 0.012570534388173168, 0.022874672145763435, 0.018009852568257885,
0.019810098845817484, 0.017995870900274654, 0.010502575996007277, 0.02360
2318269823192]
Mean accuracy: 0.01741546055281125
Standard Deviation: 0.004726999787675129
Variance: 2.2344526992680717e-05
```

In [27]:

```
# Эксперимент 2
# Первый уровень - две модели: дерево и линейная регрессия
# Второй уровень: случайный лес

stacker = Regressor(dataset=stack_ds, estimator=RandomForestRegressor)
results = stacker.validate(k=10,scorer=mean_absolute_error)
```

```
Metric: mean_absolute_error
Folds accuracy: [0.013459999999999998, 0.010139999999999998, 0.02068, 0.0
1226, 0.023759999999999996, 0.01286, 0.0119, 0.01728, 0.009559999999999999
9, 0.018156312625250498]
Mean accuracy: 0.015005631262525049
Standard Deviation: 0.004488233419477417
Variance: 2.014423922771394e-05
```

In [28]:

```
# Эксперимент 3
# Первый уровень - три модели: дерево, линейная регрессия и случайный ле
# 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: [0.009433002874279733, 0.009531921644314214, 0.0168113401
6660672, 0.007113589643901484, 0.019280611674909267, 0.00851901993010258,
0.014493334929616247, 0.01355565562800903, 0.0055874916354488185, 0.01428
1302538161626]
Mean accuracy: 0.011860727066534972
Standard Deviation: 0.004237910249030783
Variance: 1.7959883278840157e-05
```

In [29]:

```
# Эксперимент 4
# Первый уровень - три модели: дерево, линейная регрессия и случайный ле
# 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)
```

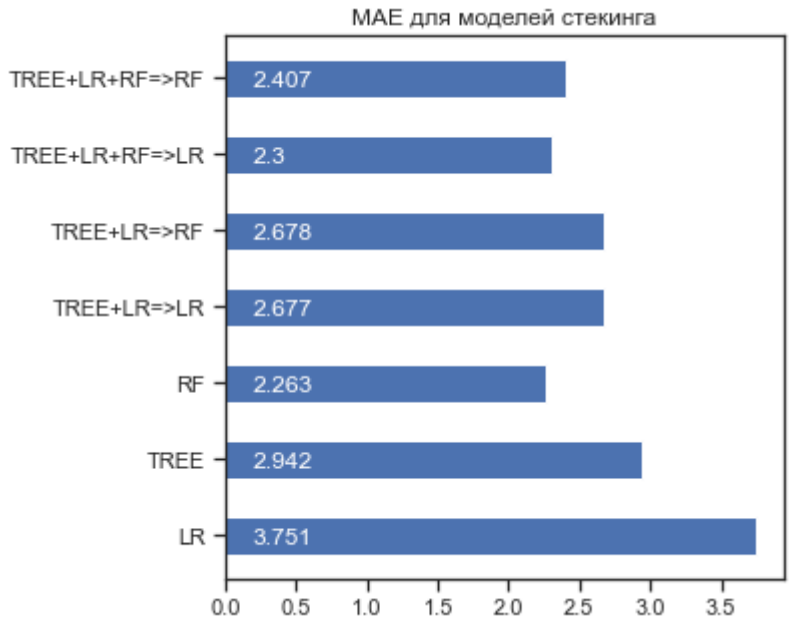
```
Metric: mean_absolute_error
Folds accuracy: [0.009880000000000002, 0.00992, 0.017799999999999996, 0.0
0536, 0.02014, 0.00796, 0.0121, 0.01642, 0.005560000000000001, 0.01747494
98997996]
Mean accuracy: 0.01226149498997996
Standard Deviation: 0.005093553573954725
Variance: 2.5944288010746946e-05
```

In [30]:

```
# 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]
```

In [31]:

```
# Визуализация результатов
vis_models_quality(array_mae, array_labels, 'MAE для моделей стекинга')
```



In [ ]: