

```
import numpy as np
import pandas as pd
import scipy.stats as sps
from tqdm.notebook import tqdm

import matplotlib.pyplot as plt
import seaborn as sns

from sklearn.ensemble import RandomForestRegressor
from sklearn.ensemble import BaggingRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.linear_model import Ridge, LinearRegression
from sklearn.model_selection import RandomizedSearchCV
from sklearn.model_selection import train_test_split
from sklearn.model_selection import cross_val_score
from sklearn.datasets import fetch_california_housing
from sklearn.metrics import mean_squared_error as mse
from sklearn.utils import shuffle

sns.set(font_scale=1.5)
```

▼ Задача 3.3

В этой задаче вам предлагается исследовать зависимость качества предсказаний модели случайного леса в зависимости от различных гиперпараметров на примере задаче регрессии. Будем использовать класс `RandomForestRegressor` библиотеки `sklearn`.

В качестве данных возьмём датасет `california_housing` из библиотеки `sklearn` о стоимости недвижимости в различных округах Калифорнии. Этот датасет состоит из 20640 записей и содержит следующие признаки для каждого округа: `MedInc`, `HouseAge`, `AveRooms`, `AveBedrms`, `Population`, `AveOccup`, `Latitude`, `Longitude`. `HouseAge` и `Population` — целочисленные признаки. Остальные признаки — вещественные.

Совет. При отладке кода используйте небольшую часть данных. Финальные вычисления проведите на полных данных. Для оценки времени работы используйте `tqdm` в циклах.

```
housing = fetch_california_housing()
X, y = housing.data, housing.target
```

Разбейте данные на обучающую выборку и на валидацию, выделив на валидацию 25% данных.

```
X_train, X_test, y_train, y_test = train_test_split(
    X, y, test_size=0.25, random_state=17
)
```

Посмотрите, как изменяется качество леса в зависимости от выбранных параметров. Для этого постройте графики зависимости MSE на тестовой выборке от количества деревьев (от 1 до 100) и от максимальной глубины дерева (от 3 до 25). Когда варьируете один из параметров, в качестве другого берите значение по умолчанию.

```
def plot_dependence_test(param_grid, test_values, param_label,
                        metrics_label, title):
    ...

    Функция для построения графиков зависимости целевой метрики
    от некоторого параметра модели на валидационной выборке.

    Параметры.
    1) param_grid – значения исследуемого параметра,
    2) test_values – значения метрики на валидационной выборке,
    3) param_label – названия параметра,
    4) metrics_label – название метрики,
    5) title – заголовок для графика.
    ...

    plt.figure(figsize=(12, 6))
    plt.plot(param_grid, test_values, label='test', linewidth=3)

    plt.xlabel(param_label)
    plt.ylabel(metrics_label)
    plt.legend()
    plt.title(title)
    plt.show()
```

Для построения зависимости от количества деревьев можно было бы построить модели для каждого количества деревьев, посчитать метрику и построить график. Однако лес — набор этих деревьев. Построив один лес, мы можем посчитать предсказания для каждого дерева в отдельности. Затем, усредняя, получаем предсказания для произвольного количества деревьев. Такой трюк позволяет экономить время проведения эксперимента.

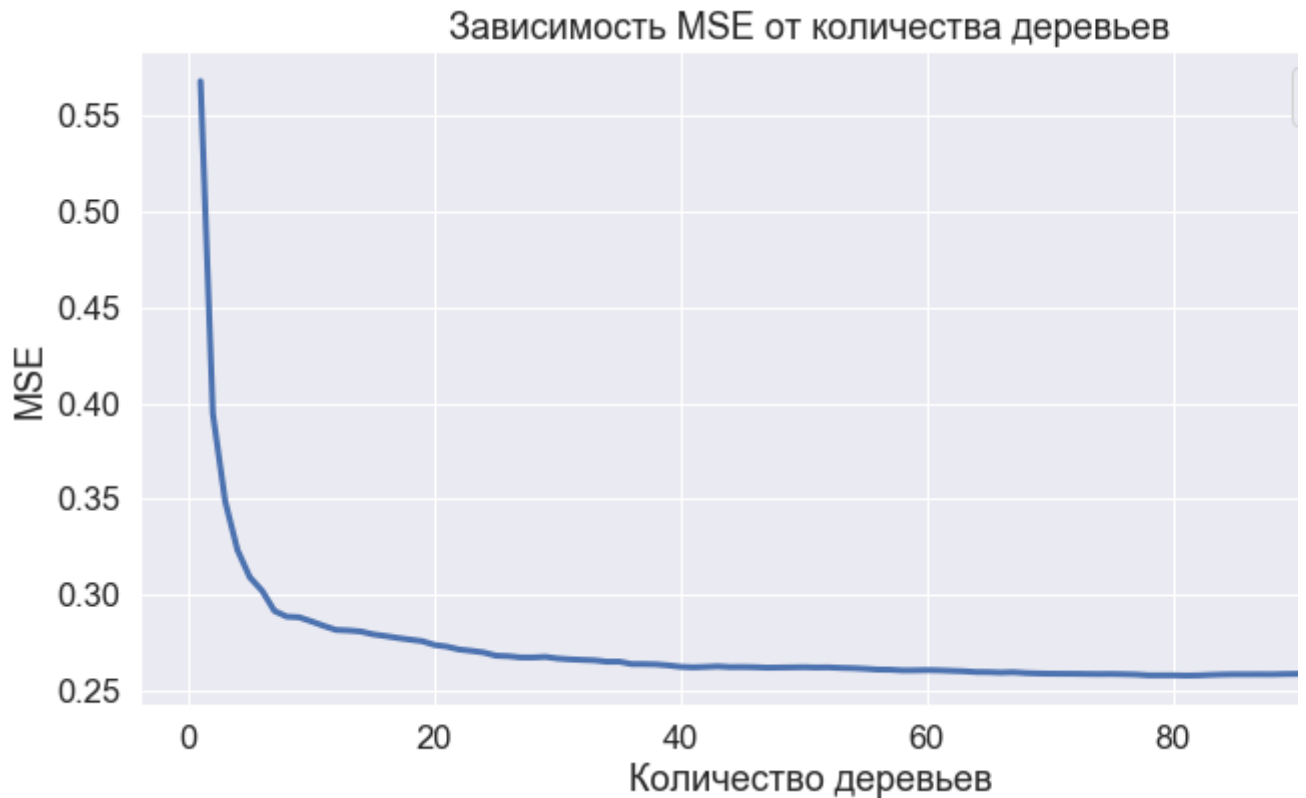
```
regressor = RandomForestRegressor(n_estimators=100, random_state=42)
regressor.fit(X_train, y_train)

predictions_by_tree = np.array(
    [tree.predict(X_test) for tree in regressor.estimators_]
)

n_estimators_grid = np.arange(1, 101)
predictions = np.cumsum(predictions_by_tree, axis=0) \
    / n_estimators_grid[:, np.newaxis]
mse_values = [mse(p, y_test) for p in predictions]

plot_dependence_test(n_estimators_grid, mse_values,
```

```
'Количество деревьев', 'MSE',
'Зависимость MSE от количества деревьев')
```



```
mse_values = []
```

```
for max_depth in tqdm(range(3, 25)):
    regressor = RandomForestRegressor(max_depth=max_depth,
                                     n_estimators=50, random_state=42)
    regressor.fit(X_train, y_train)
    predictions = regressor.predict(X_test)
    mse_values.append(mse(predictions, y_test))

    HBox(children=(FloatProgress(value=0.0, max=22.0), HTML(value='')))
```

```
plot_dependence_test(np.arange(3, 25), mse_values,
                     'Максимальная глубина дерева', 'MSE',
                     'Зависимость MSE от максимальной глубины')
```





Основываясь на полученных графиках, ответьте на следующие вопросы.

1. Какие закономерности можно увидеть на построенных графиках? Почему графики получились такими?
2. Как изменяется качество предсказаний с увеличением исследуемых параметров, когда эти параметры уже достаточно большие.
3. В предыдущем задании вы на практике убедились, что решающее дерево начинает переобучаться при достаточно больших значениях максимальной глубины. Справедливо ли это утверждение для случайного леса? Поясните свой ответ, опираясь на своё знание статистики.

Вывод.

По первому графику можно сделать вывод, что с возрастанием числа использованных деревьев MSE снижается. Но при достаточно больших значениях `n_estimators` значение MSE практически перестаёт меняться. С параметром `max_depth` ситуация аналогична. Случайный лес, в отличие от решающего дерева, гораздо менее склонен к переобучению при больших `max_depth`. Ведь использование большого количества деревьев снижает дисперсию предсказаний, а использование больших значений `max_depth` или вообще отсутствие ограничения на максимальную глубину позволяет строить модели, близкие к несмещенным, предсказания которых имеют большую дисперсию. В итоге случайный лес как комбинация большого количества глубоких деревьев имеет маленькое смещение (bias) и маленькую дисперсию.

Обучите случайный лес с параметрами по умолчанию и выведите mse на тестовой выборке. Проведите эксперимент 3 раза. Почему результаты отличаются?

```
for iteration in tqdm(range(3)):
    regressor = RandomForestRegressor(n_estimators=100)
    regressor.fit(X_train, y_train)
    predictions = regressor.predict(X_test)
    print('MSE = {:.4f}'.format(mse(predictions, y_test)))
```

```

HBox(children=(FloatProgress(value=0.0, max=3.0), HTML(value='')))
MSE = 0.2580
MSE = 0.2593
MSE = 0.2574

```

Ответ. Потому что лес случайный. Он основан на `n_estimators` случайных деревьях, которые могут очень сильно отличаться друг от друга.

Было бы неплохо определиться с тем, какое количество деревьев нужно использовать и какой максимальной глубины они будут. Подберите оптимальные значения `max_depth` и `n_estimators` с помощью кросс-валидации.

```

rf_gridsearch = RandomizedSearchCV(
    estimator=RandomForestRegressor(random_state=42),
    param_distributions={
        'max_depth': [None] + np.arange(3, 31).tolist(),
        'n_estimators': np.arange(25, 401)
    },
    scoring='neg_mean_squared_error', # оптимизируемый скор
    cv=5, # разбиение выборки на 5 фолдов
    verbose=10, # насколько часто печатать сообщения
    n_jobs=2, # кол-во параллельных процессов
    n_iter=100, # кол-во итераций случайного выбора гиперпараметров
    random_state=0 # seed для фиксации генератора случайных чисел
)

```

```
rf_gridsearch.fit(X_train, y_train)
```

```

Fitting 5 folds for each of 100 candidates, totalling 500 fits
[Parallel(n_jobs=2)]: Using backend LokyBackend with 2 concurrent workers.
[Parallel(n_jobs=2)]: Done 1 tasks | elapsed: 9.4s
[Parallel(n_jobs=2)]: Done 4 tasks | elapsed: 18.2s
[Parallel(n_jobs=2)]: Done 9 tasks | elapsed: 1.6min
[Parallel(n_jobs=2)]: Done 14 tasks | elapsed: 2.2min
[Parallel(n_jobs=2)]: Done 21 tasks | elapsed: 3.6min
[Parallel(n_jobs=2)]: Done 28 tasks | elapsed: 5.3min
[Parallel(n_jobs=2)]: Done 37 tasks | elapsed: 8.1min
[Parallel(n_jobs=2)]: Done 46 tasks | elapsed: 10.2min
[Parallel(n_jobs=2)]: Done 57 tasks | elapsed: 11.9min
[Parallel(n_jobs=2)]: Done 68 tasks | elapsed: 13.7min
[Parallel(n_jobs=2)]: Done 81 tasks | elapsed: 16.6min
[Parallel(n_jobs=2)]: Done 94 tasks | elapsed: 18.4min
[Parallel(n_jobs=2)]: Done 109 tasks | elapsed: 20.7min
[Parallel(n_jobs=2)]: Done 124 tasks | elapsed: 23.2min
[Parallel(n_jobs=2)]: Done 141 tasks | elapsed: 25.7min
[Parallel(n_jobs=2)]: Done 158 tasks | elapsed: 26.8min
[Parallel(n_jobs=2)]: Done 177 tasks | elapsed: 27.5min
[Parallel(n_jobs=2)]: Done 196 tasks | elapsed: 31.1min
[Parallel(n_jobs=2)]: Done 217 tasks | elapsed: 33.6min
[Parallel(n_jobs=2)]: Done 238 tasks | elapsed: 34.8min
[Parallel(n_jobs=2)]: Done 261 tasks | elapsed: 38.7min
[Parallel(n_jobs=2)]: Done 284 tasks | elapsed: 41.3min
[Parallel(n_jobs=2)]: Done 309 tasks | elapsed: 44.8min
[Parallel(n_jobs=2)]: Done 334 tasks | elapsed: 46.4min
[Parallel(n_jobs=2)]: Done 361 tasks | elapsed: 51.5min

```

```

[Parallel(n_jobs=2)]: Done 388 tasks      | elapsed: 56.9min
[Parallel(n_jobs=2)]: Done 417 tasks      | elapsed: 61.3min
[Parallel(n_jobs=2)]: Done 446 tasks      | elapsed: 65.2min
[Parallel(n_jobs=2)]: Done 477 tasks      | elapsed: 69.3min
[Parallel(n_jobs=2)]: Done 500 out of 500 | elapsed: 71.3min finished
RandomizedSearchCV(cv=5, error_score=nan,
                    estimator=RandomForestRegressor(bootstrap=True,
                                                    ccp_alpha=0.0,
                                                    criterion='mse',
                                                    max_depth=None,
                                                    max_features='auto',
                                                    max_leaf_nodes=None,
                                                    max_samples=None,
                                                    min_impurity_decrease=0.0,
                                                    min_impurity_split=None,
                                                    min_samples_leaf=1,
                                                    min_samples_split=2,
                                                    min_weight_fraction_leaf=0.0,
                                                    n_estimators=100,
                                                    n_jobs=None, oob_score=Fals...
                    337, 338, 339, 340, 341, 342, 343, 344, 345, 346, 347, 348, 349,
                    350, 351, 352, 353, 354, 355, 356, 357, 358, 359, 360, 361, 362,
                    363, 364, 365, 366, 367, 368, 369, 370, 371, 372, 373, 374, 375,
                    376, 377, 378, 379, 380, 381, 382, 383, 384, 385, 386, 387, 388,
                    389, 390, 391, 392, 393, 394, 395, 396, 397, 398, 399, 400])),
                    pre_dispatch='2*n_jobs', random_state=0, refit=True,
                    return_train_score=False, scoring='neg_mean_squared_error',
                    verbose=10)

```

Выведите найденные оптимальные параметры.

```

print(rf_gridsearch.best_params_)

{'n_estimators': 325, 'max_depth': 25}

```

Зафиксируем эти оптимальные значения параметров и в дальнейшем будем их использовать.

```

max_depth = rf_gridsearch.best_params_['max_depth']
n_estimators = rf_gridsearch.best_params_['n_estimators']

```

Оценим качество предсказаний обученного решающего леса.

```

predictions = rf_gridsearch.best_estimator_.predict(X_test)
print('{:.4f}'.format(mse(predictions, y_test)))

0.2568

```

Исследуйте зависимость метрики `mse` от количества признаков, по которым происходит разбиение в вершине дерева. Поскольку количество признаков в датасете не очень

большое (их 8), то можно перебрать все возможные варианты количества признаков, использующихся при разбиении вершин.

```
mse_train_values = []
mse_test_values = []

for n_features in tqdm(range(1, 9)):
    rf_regressor = RandomForestRegressor(
        max_depth=max_depth, n_estimators=n_estimators,
        max_features=n_features, random_state=42
    )
    rf_regressor.fit(X_train, y_train)

    current_train_mse = mse(rf_regressor.predict(X_train), y_train)
    current_test_mse = mse(rf_regressor.predict(X_test), y_test)
    print('n_features: {}, train_mse: {:.4f}, test_mse: {:.4f}'.format(
        n_features, current_train_mse, current_test_mse
    ))

    mse_train_values.append(current_train_mse)
    mse_test_values.append(current_test_mse)

HBox(children=(FloatProgress(value=0.0, max=8.0), HTML(value='')))
n_features: 1, train_mse: 0.0385, test_mse: 0.2890
n_features: 2, train_mse: 0.0325, test_mse: 0.2525
n_features: 3, train_mse: 0.0321, test_mse: 0.2492
n_features: 4, train_mse: 0.0327, test_mse: 0.2511
n_features: 5, train_mse: 0.0334, test_mse: 0.2559
n_features: 6, train_mse: 0.0334, test_mse: 0.2565
n_features: 7, train_mse: 0.0336, test_mse: 0.2557
n_features: 8, train_mse: 0.0338, test_mse: 0.2568
```

Постройте график зависимости метрики mse на test и train в зависимости от числа признаков, использующихся при разбиении в каждой вершине.

```
def plot_dependence(param_grid, train_values, test_values,
                    param_label='', metrics_label='', title='',
                    train_label='train (RandomForest)',
                    test_label='test (RandomForest)',
                    create_figure=True):
    ...
```

Функция для построения графиков зависимости целевой метрики от некоторого параметра модели на обучающей и на валидационной выборке.

Параметры.

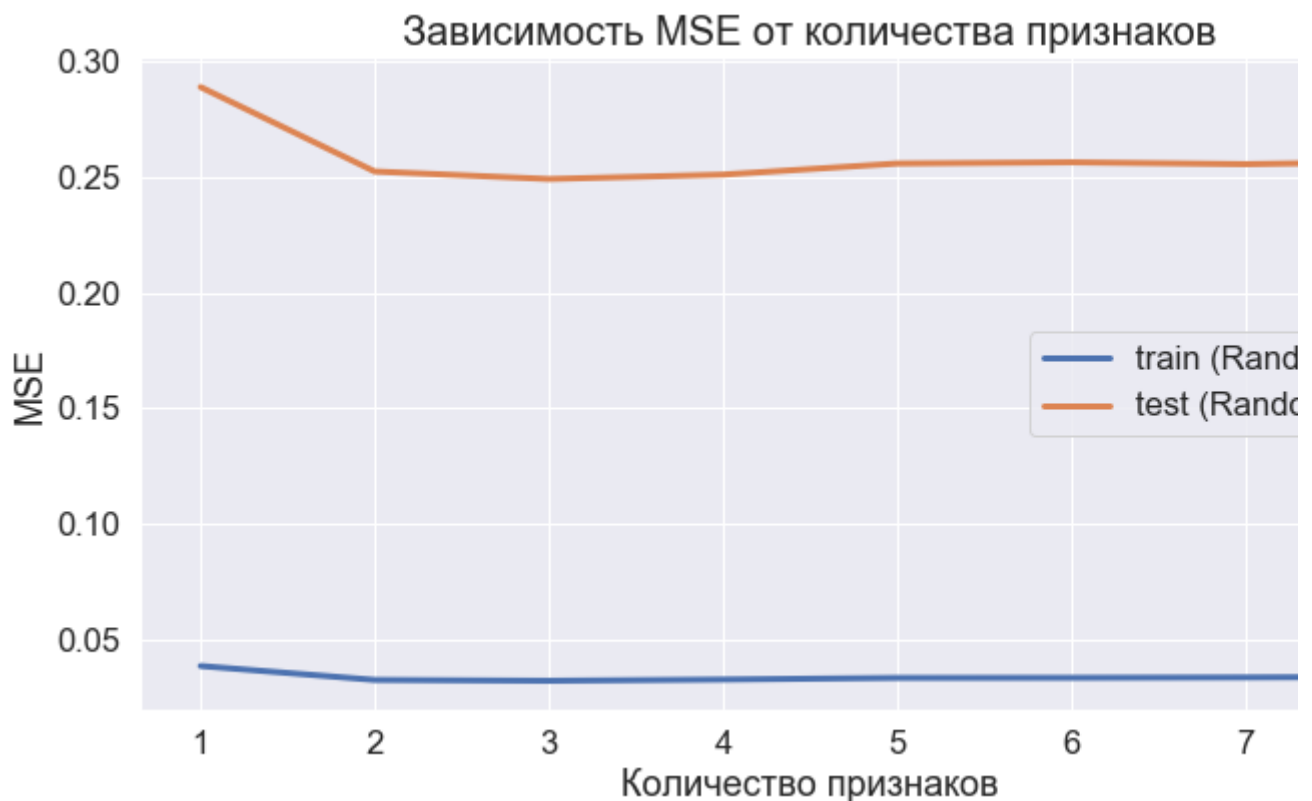
- 1) param_grid – значения исследуемого параметра,
- 2) train_values – значения метрики на обучающей выборке,
- 3) test_values – значения метрики на валидационной выборке,
- 4) param_label – названия параметра,
- 5) metrics_label – название метрики,
- 6) title – заголовок для графика,

7) `create_figure` – флаг, устанавливающий нужно ли создавать новую фигуру для графика.
 ...

```
if create_figure:
    plt.figure(figsize=(12, 6))
plt.plot(param_grid, train_values, label=train_label, linewidth=3)
plt.plot(param_grid, test_values, label=test_label, linewidth=3)

plt.legend()
if create_figure:
    plt.xlabel(param_label)
    plt.ylabel(metrics_label)
    plt.title(title, fontsize=20)
```

```
plot_dependence(range(1, 9), mse_train_values, mse_test_values,
                 'Количество признаков', 'MSE',
                 'Зависимость MSE от количества признаков')
```



Почему график получился таким? Как зависит разнообразие деревьев от величины `n_features`?

Вывод.

Как можно заметить по графику, оптимальное число признаков для тестовой выборки равно 3, а для обучающей выборки ошибка практически не меняется с увеличением количества признаков. Значит, действительно не всегда стоит выбирать при каждом разбиении вершины из всех признаков, поскольку это может привести к переобучению.

При этом, при увеличении значения `n_features` деревья в случайном лесу становятся всё более похожими друг на друга и их попарная корреляция увеличивается. При этом, если взять значение `n_features` равным количеству признаков в датасете, то в каждой вершине дерева будут оптимизировать значения одинакового функционала, выбирая признак из одинакового множества признаков, и останется единственный фактор случайности — бутстре핑ованная выборка, которая может меняться у разных деревьев.

Проведите эксперимент, в котором выясните, как изменится качество регрессии, если набор признаков, по которым происходит разбиение в каждой вершине определяется не заново в каждой вершине, а задан заранее для всего дерева (то есть перед каждым построением дерева выбирается новый набор признаков, который будет рассматриваться при построении всех разбиений в этом дереве).

Для реализации данного эксперимента используйте класс беггинг-модели `sklearn.ensemble.BaggingRegressor`, у которого используйте следующие поля:

- `base_estimator` — базовая модель, используйте `sklearn.tree.DecisionTreeRegressor`;
- `max_features` — количество признаков для каждой базовой модели;
- `n_estimators` — количество базовых моделей.

Постройте графики `mse` на обучающей и на валидационной выборке в зависимости от числа рассматриваемых признаков в дереве.

```
stupid_mse_train_values = []
stupid_mse_test_values = []

for n_features in tqdm(range(1, 9)):
    rf_regressor = BaggingRegressor(
        base_estimator=DecisionTreeRegressor(
            max_depth=max_depth
        ),
        random_state=42,
        max_features=n_features,
        n_estimators=100
    )

    rf_regressor.fit(X_train, y_train)
    current_train_mse = mse(rf_regressor.predict(X_train), y_train)
    current_test_mse = mse(rf_regressor.predict(X_test), y_test)
    print('n_features: {}, train_mse: {:.4f}, test_mse: {:.4f}'.format(
        n_features, current_train_mse, current_test_mse))

    stupid_mse_train_values.append(current_train_mse)
    stupid_mse_test_values.append(current_test_mse)
```

```

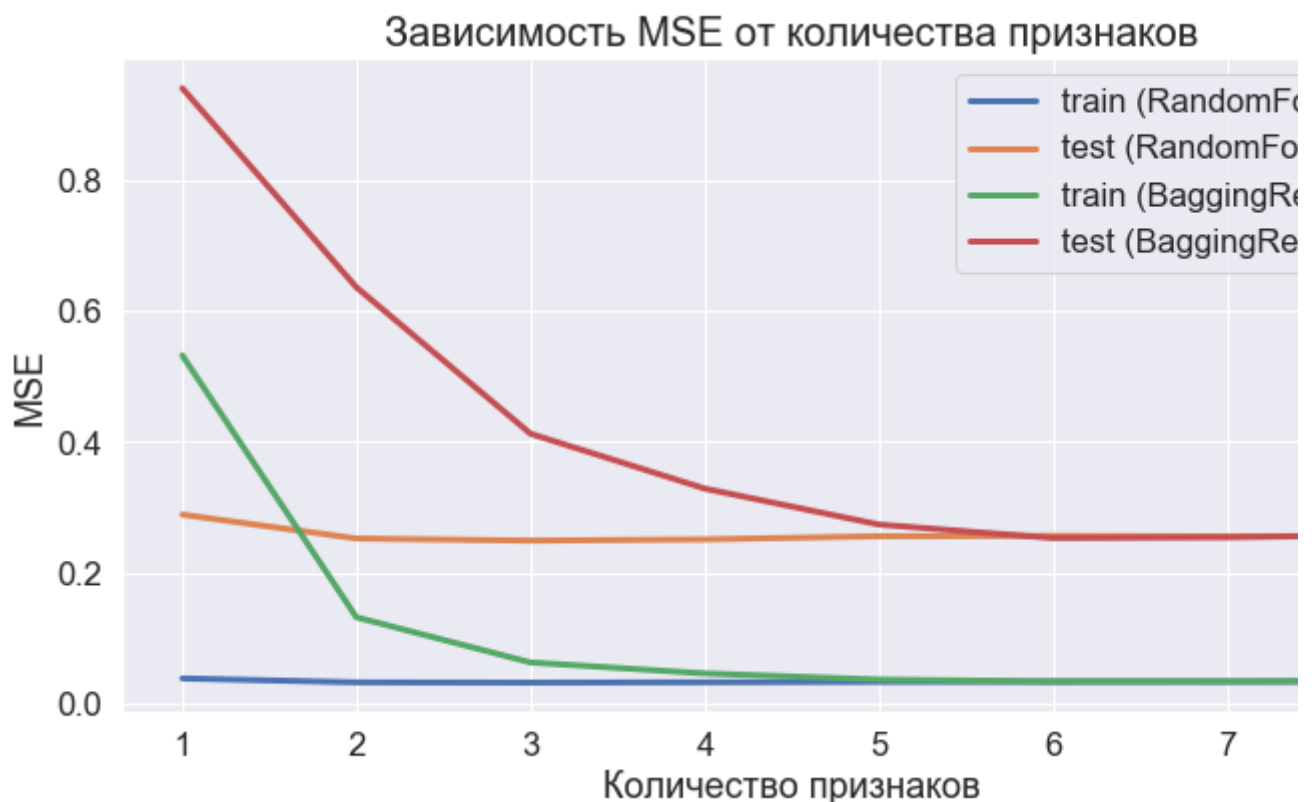
HBox(children=(FloatProgress(value=0.0, max=8.0), HTML(value='')))
n_features: 1, train_mse: 0.5326, test_mse: 0.9411
n_features: 2, train_mse: 0.1320, test_mse: 0.6366
n_features: 3, train_mse: 0.0629, test_mse: 0.4126
n_features: 4, train_mse: 0.0462, test_mse: 0.3288
n_features: 5, train_mse: 0.0367, test_mse: 0.2738
n_features: 6, train_mse: 0.0340, test_mse: 0.2532

```

```

plot_dependence(range(1, 9), mse_train_values, mse_test_values,
                 'Количество признаков', 'MSE',
                 'Зависимость MSE от количества признаков')
plot_dependence(range(1, 9), stupid_mse_train_values,
                 stupid_mse_test_values,
                 train_label='train (BaggingRegressor)',
                 test_label='test (BaggingRegressor)',
                 create_figure=False)

```



Сравните результаты обычного случайного леса с только что построенным лесом.

Сделайте выводы. Объясните, чем плох такой подход построения случайного леса. Какое преимущество мы получаем, когда выбираем случайное подмножество признаков в каждой вершине в обычном случайном лесу?

Вывод.

Во всех экспериментах при фиксации набора признаков значение MSE стало больше, чем без фиксации признаков. Из этого можно сделать вывод, что фиксировать признаки заранее — плохая идея. Действительно, если мы строим дерево таким образом, то мы, по сути, просто отказываемся от некоторых признаков и надеемся, что без них регрессия

будет давать хороший результат. Из-за этого множество различных деревьев сужается.

▼ Задача 3.4

На лекции получена формула bias-variance разложения для беггинга. Проведите эксперимент, в котором выясните, насколько уменьшается разброс (variance-компонента) беггинг-модели на 100 базовых моделях по отношению к одной базовой модели. Используйте данные из предыдущей задачи. Рассмотрите беггинг на следующих базовых моделях:

- решающие деревья, можно использовать вариант случайного леса,
- ридж-регрессия.

1. Оцените корреляции предсказаний базовых моделей, входящих в состав беггинг-модели, для тестовой выборки. Постройте матрицу корреляций предсказаний базовых моделей и гистограмму по ним. Базовые модели можно получить с помощью поля `estimators_` у обученной беггинг-модели. Какую оценку корреляций вы используете и почему?
2. Используя среднее значение полученных корреляции как оценку корреляции моделей, посчитайте разброс беггинг-модели. Мы хотим оценить во сколько раз уменьшится дисперсия композиции по сравнению с одной базовой моделью, поэтому можете предположить, что дисперсия всех базовых моделей равна 1 (это не повлияет на отношение двух величин).

Обучим случайный лес и бэггинг над Ridge-регрессией.

```
ridge_bagging = BaggingRegressor(
    base_estimator=Ridge(), n_estimators=100, random_state=42
).fit(X_train, y_train)
rf_regressor = RandomForestRegressor(
    n_estimators=100, random_state=42
).fit(X_train, y_train)
```

В качестве оценки коэффициента корреляции будем использовать коэффициент Пирсона. Он оценивает степень линейной зависимости между выборками. Поскольку в variance-компоненте содержится теоретическая ковариация, тоже характеризующая степень линейной зависимости, использование коэффициента Пирсона логично в этой задаче.

```
for model in [rf_regressor, ridge_bagging]:
    print(model)
```

```
predictions = [
    estimator.predict(X_test) for estimator in model.estimators_
]
estimators_count = len(model.estimators_)

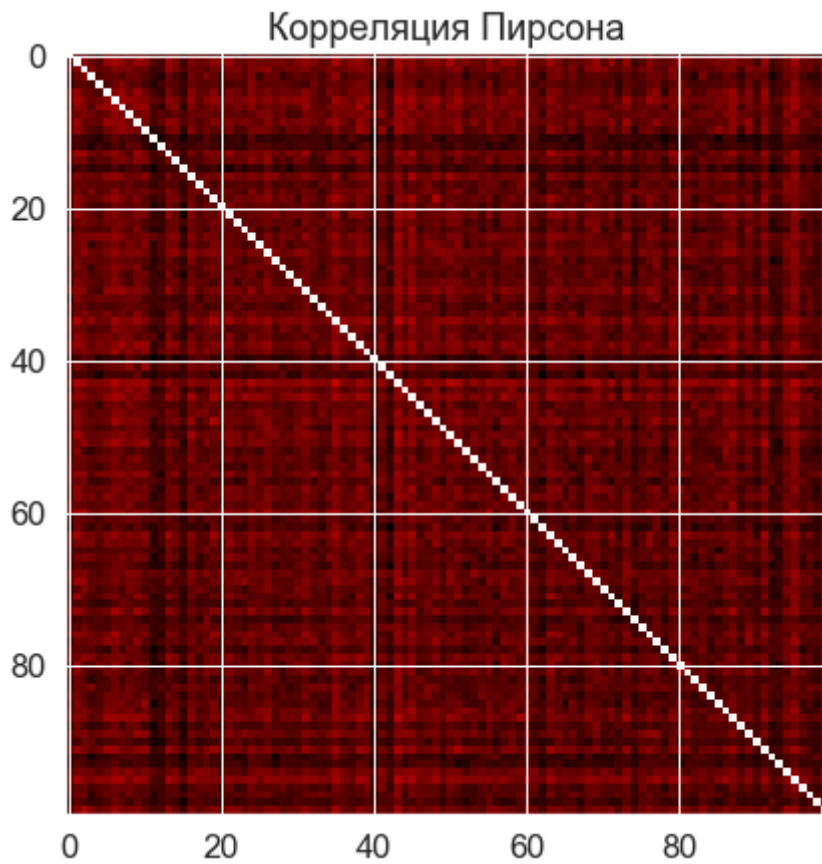
pearson_matrix = np.zeros((estimators_count, estimators_count))
for i in range(estimators_count):
    for j in range(i, estimators_count):
        pearson_matrix[i, j] = sps.pearsonr(predictions[i],
                                             predictions[j])[0]
        pearson_matrix[j, i] = pearson_matrix[i, j]

# визуализируем полученные матрицы ковариаций
plt.figure(figsize=(15, 7))
plt.subplot(121)
plt.imshow(pearson_matrix, interpolation='none', cmap='hot')
plt.title('Корреляция Пирсона')
plt.show()

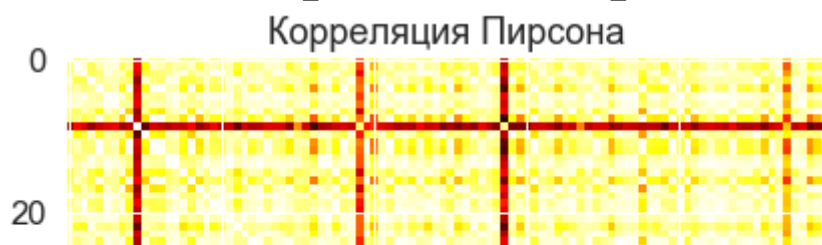
plt.figure(figsize=(15, 5))

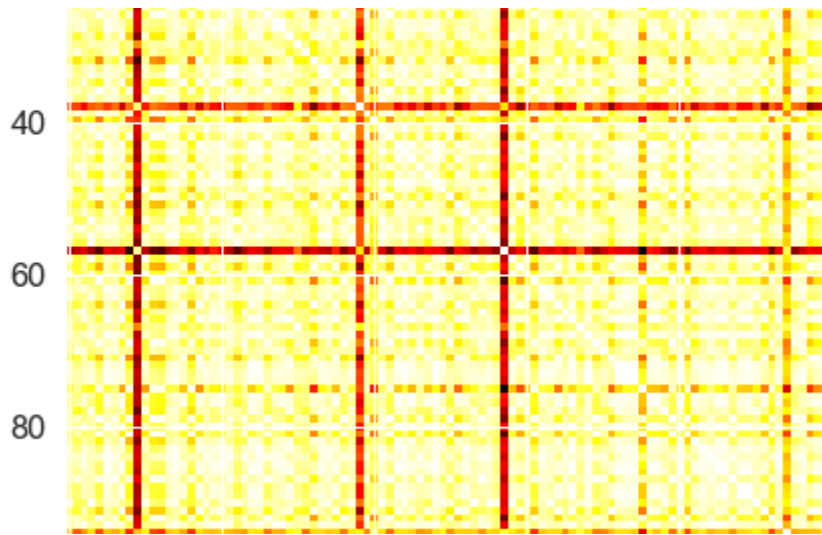
# построим гистограммы ковариаций
plt.subplot(111)
ravel_corr = pearson_matrix[np.tril_indices(100, k=-1)]
plt.hist(ravel_corr, bins=30)
plt.title('Пирсон: {:.3f} +- {:.3f}'.format(ravel_corr.mean(),
                                             ravel_corr.std()))
plt.show()
```

```
RandomForestRegressor(bootstrap=True, ccp_alpha=0.0, criterion='mse',
                      max_depth=None, max_features='auto', max_leaf_nodes=None,
                      max_samples=None, min_impurity_decrease=0.0,
                      min_impurity_split=None, min_samples_leaf=1,
                      min_samples_split=2, min_weight_fraction_leaf=0.0,
                      n_estimators=100, n_jobs=None, oob_score=False,
                      random_state=42, verbose=0, warm_start=False)
```



```
BaggingRegressor(base_estimator=Ridge(alpha=1.0, copy_X=True,
                                     fit_intercept=True, max_iter=None,
                                     normalize=False, random_state=None,
                                     solver='auto', tol=0.001),
                 bootstrap=True, bootstrap_features=False, max_features=1.0,
                 max_samples=1.0, n_estimators=100, n_jobs=None,
                 oob_score=False, random_state=42, verbose=0, warm_start=False)
```





Посчитаем разброс бэггинга над Ridge-регрессией.

```
ridge_var = 1/100 + 99/100 * 0.998
print(ridge_var)
```

0.99802



Посчитаем разброс случайного леса.

400

```
rf_var = 1/100 + 99/100 * 0.76
print(rf_var)
```

0.7624



Оценим, во сколько раз бэггинг уменьшил разброс предсказаний базовых моделей.

Для леса:

```
1 / rf_var
```

1.311647429171039

Для Ridge-регрессии:

```
1 / ridge_var
```

1.001983928177792

Вывод.

Как мы видим, в обоих случаях разброс предсказаний уменьшился.

В случайном лесе базовые модели оказались менее коррелированными, чем в бэггинге над Ridge-регрессией, поэтому случайный лес сильнее уменьшает разброс. Это связано

с тем, что в лесе деревья могут быть достаточно разнообразными в отличии от линейной регрессии.

