**Московский государственный технический университет им. Н.Э. Баумана**

Факультет «Информатика и системы управления»
Кафедра ИУ5 «Системы обработки информации и управления»

Курс «Технологии машинного обучения».

# Отчет по лабораторной работе №5
# «Ансамбли моделей машинного обучения.»

Выполнил:
студент группы ИУ5-61Б
Головацкий А.Д.

Проверил:
Гапанюк Ю. Е.

Москва, 2022 г.

## ИУ5-61Б Головацкий Андрей

## Ансамбли моделей машинного обучения.

1. Выберите набор данных (датасет) для решения задачи классификации или регресии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода train_test_split разделите выборку на обучающую и тестовую.
4. Обучите 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.
5. (+1 балл на экзамене) Дополнительно к указанным моделям обучите еще две модели:
   - Модель многослойного персептрона. По желанию, вместо библиотеки scikit-learn возможно использование библиотек TensorFlow, PyTorch или других аналогичных библиотек.
   - Модель МГУА с использованием библиотеки - https://github.com/kvoyager/GmdhPy (или аналогичных библиотек). Найдите такие параметры запуска модели, при которых она будет по крайней мере не хуже, чем одна из предыдущих ансамблевых моделей.
6. Оцените качество моделей с помощью одной из подходящих для задачи метрик. Сравните качество полученных моделей.

```
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
import numpy as np
from sklearn.preprocessing import PolynomialFeatures, MinMaxScaler,
StandardScaler
from sklearn.linear_model import LinearRegression
from sklearn.metrics import r2_score, mean_squared_error,
mean_absolute_error
from sklearn.model_selection import train_test_split, GridSearchCV
from sklearn.ensemble import RandomForestRegressor,
GradientBoostingRegressor
from heamy.estimator import Regressor
```

```
from heamy.pipeline import ModelsPipeline
from heamy.dataset import Dataset
from sklearn.neural_network import MLPRegressor
from warnings import simplefilter

sns.set(style="ticks")
import warnings
warnings.filterwarnings('ignore')
simplefilter('ignore')

data = pd.read_csv('datasets/real_estate_data.csv', sep='\t')
data.head()
```

```
   total_images  last_price  total_area first_day_exposition  rooms  \
0            20  13000000.0       108.0  2019-03-07T00:00:00      3
1             7   3350000.0        40.4  2018-12-04T00:00:00      1
2            10   5196000.0        56.0  2015-08-20T00:00:00      2
3             0  64900000.0       159.0  2015-07-24T00:00:00      3
4             2  10000000.0       100.0  2018-06-19T00:00:00      2

   ceiling_height  floors_total  living_area  floor is_apartment  ...
\
0            2.70          16.0         51.0      8          NaN  ...

1             NaN          11.0         18.6      1          NaN  ...

2             NaN           5.0         34.3      4          NaN  ...

3             NaN          14.0          NaN      9          NaN  ...

4            3.03          14.0         32.0     13          NaN  ...


   kitchen_area  balcony    locality_name  airports_nearest  \
0          25.0      NaN  Санкт-Петербург           18863.0
1          11.0      2.0   посёлок Шушары           12817.0
2           8.3      0.0  Санкт-Петербург           21741.0
3           NaN      0.0  Санкт-Петербург           28098.0
4          41.0      NaN  Санкт-Петербург           31856.0

  cityCenters_nearest  parks_around3000  parks_nearest
ponds_around3000  \
0             16028.0               1.0          482.0
2.0
1             18603.0               0.0            NaN
0.0
2             13933.0               1.0           90.0
2.0
3              6800.0               2.0           84.0
3.0
```

```
4              8098.0               2.0          112.0
1.0

   ponds_nearest  days_exposition
0          755.0              NaN
1            NaN             81.0
2          574.0            558.0
3          234.0            424.0
4           48.0            121.0

[5 rows x 22 columns]
```

```
# Список колонок с типами данных
data.dtypes
```

```
total_images              int64
last_price              float64
total_area              float64
first_day_exposition     object
rooms                     int64
ceiling_height          float64
floors_total            float64
living_area             float64
floor                     int64
is_apartment             object
studio                     bool
open_plan                  bool
kitchen_area            float64
balcony                 float64
locality_name            object
airports_nearest        float64
cityCenters_nearest     float64
parks_around3000        float64
parks_nearest           float64
ponds_around3000        float64
ponds_nearest           float64
days_exposition         float64
dtype: object
```

```
# Проверим наличие пустых значений
data.isnull().sum()
```

```
total_images                0
last_price                  0
total_area                  0
first_day_exposition        0
rooms                       0
ceiling_height           9195
floors_total               86
living_area              1903
floor                       0
```

```
is_apartment            20924
studio                      0
open_plan                   0
kitchen_area             2278
balcony                 11519
locality_name              49
airports_nearest         5542
cityCenters_nearest      5519
parks_around3000         5518
parks_nearest           15620
ponds_around3000         5518
ponds_nearest           14589
days_exposition          3181
dtype: int64
```

## Обработка данных

### Пропущенные данные (к оглавлению)

Выведем список параметров датасета и для каждого из них найдём процент null значений.

```
proportion_nans = []
for i in data.columns:
    print('{} : {:.2%}\n'.format(i,
data[i].isna().sum()/data.shape[0]))
    proportion_nans.append([i,data[i].isna().sum()/data.shape[0] *
100])
```

```
total_images : 0.00%

last_price : 0.00%

total_area : 0.00%

first_day_exposition : 0.00%

rooms : 0.00%

ceiling_height : 38.80%

floors_total : 0.36%

living_area : 8.03%

floor : 0.00%

is_apartment : 88.29%
```

```
studio : 0.00%

open_plan : 0.00%

kitchen_area : 9.61%

balcony : 48.61%

locality_name : 0.21%

airports_nearest : 23.38%

cityCenters_nearest : 23.29%

parks_around3000 : 23.28%

parks_nearest : 65.91%

ponds_around3000 : 23.28%

ponds_nearest : 61.56%

days_exposition : 13.42%
```

Столбцы, у которых пропущено **>40%** данных, удалим, т.к. в данном датасете они не имеют большого веса, однако иногда такие удаления могут навредить, например, может вырасти количество дубликатов, порой в разы.

Столбцы, у которых пропущено **от 0.1% до 40%** данных, заполним средним по квантилям

```
columns_to_del = []
columns_tofill_mean = []
columns_tofill_median = []
for i in proportion_nans:
    if(i[1] > 40):
        columns_to_del.append(i[0])
        print('{:19} ({:5.2f}%) => columns_to_del'.format(i[0],i[1]))
    elif (i[1] > 0):
        columns_tofill_median.append(i[0])
        print('{:19} ({:5.2f}%) =>
columns_tofill_median'.format(i[0],i[1]))
```

```
ceiling_height      (38.80%) => columns_tofill_median
floors_total        ( 0.36%) => columns_tofill_median
living_area         ( 8.03%) => columns_tofill_median
```

```
is_apartment        (88.29%) => columns_to_del
kitchen_area        ( 9.61%) => columns_tofill_median
balcony             (48.61%) => columns_to_del
locality_name       ( 0.21%) => columns_tofill_median
airports_nearest    (23.38%) => columns_tofill_median
cityCenters_nearest (23.29%) => columns_tofill_median
parks_around3000    (23.28%) => columns_tofill_median
parks_nearest       (65.91%) => columns_to_del
ponds_around3000    (23.28%) => columns_tofill_median
ponds_nearest       (61.56%) => columns_to_del
days_exposition     (13.42%) => columns_tofill_median
```

### Проблемы конкретных столбцов (к оглавлению)

Проверим данные на наличие дубликатов. Для начала проверим параметр `locality_name`.

```
print(f"Уникальных значений параметра 'locality_name':
{data['locality_name'].unique().size}.")
print(f"Количество записей в датасете: {data.shape[0]}.")
```

```
Уникальных значений параметра 'locality_name': 365.
Количество записей в датасете: 23699.
```

Видна проблема дублирования данных. Например посёлок Бугры и поселок Бугры это одно и то же, но в таблице отмечены как разные значения. Необходимо лемматизировать данные в этом столбце и убрать тип населенного пункта в значениях.

### Обработка пропущенных значений (к оглавлению)

```
for i in columns_tofill_median:
    print("{:19} - {}".format(i, data[i].isna().sum()))
```

```
ceiling_height      - 9195
floors_total        - 86
living_area         - 1903
kitchen_area        - 2278
locality_name       - 49
airports_nearest    - 5542
cityCenters_nearest - 5519
parks_around3000    - 5518
ponds_around3000    - 5518
days_exposition     - 3181
```

Столбец **locality_name** имеет строковый тип данных и заполнить его медианой не получится

Так как данные пропущены всего лишь в 49 строках можно просто их удалить

```
data.dropna(subset=['locality_name'], inplace=True)
data.reset_index(inplace=True, drop=True)
data['locality_name'].isna().sum()
```

```
0
```

Обработаем созданный ранее список **columns_tofill_median**, пропуски в котором заполним значениями квантилей и медианой(1/3 заполняется первым квантилем, 1/3 заполняется третим квартилем, 1/3 заполняется медианой). Заменять просто на медиану некоторые параметры - это очень сильно усреднять показатели. Если высота потолков спокойно выдержит такую процедуру, т.к. это не настолько уникальный параметр, то жилую площадь, площадь кухни лучше заменять в зависимости от какого-нибудь коэффициента, или в зависимости от категорий.

Можно выдвинуть гипотезу, почему такие пропуски образовались. Например, days_exposition пропуск может свидетельствовать, что или объявление еще не снято, или было снято в течении суток после выставления.

```
for i in columns_tofill_median:
    if(data[i].dtype != object):
        Q1 = data[i].quantile(0.25)
        Q3 = data[i].quantile(0.75)
        med = data[i].median()
        c = int(data[i].isna().sum() * 0.33)
        data[i].fillna(Q1,limit = c ,inplace=True)
        data[i].fillna(Q3,limit = c ,inplace=True)
        data[i].fillna(med,limit = c ,inplace=True)
        data.dropna(subset=[i], inplace=True)

data.reset_index(inplace=True)
```

```
for i in columns_tofill_median:
    print("{:19} - {}".format(i, data[i].isna().sum()))
```

```
ceiling_height      - 0
floors_total        - 0
living_area         - 0
kitchen_area        - 0
locality_name       - 0
airports_nearest    - 0
cityCenters_nearest - 0
parks_around3000    - 0
ponds_around3000    - 0
days_exposition     - 0
```

Видим, что пропущенные данные теперь отсутствуют. Обработаем созданный ранее список **columns_to_del**, столбцы входящие в этот список надо удалить.

```
print(columns_to_del)
```

```
['is_apartment', 'balcony', 'parks_nearest', 'ponds_nearest']
```

```
data = data.drop(columns=columns_to_del)
data.columns
```

```
Index(['index', 'total_images', 'last_price', 'total_area',
       'first_day_exposition', 'rooms', 'ceiling_height',
'floors_total',
       'living_area', 'floor', 'studio', 'open_plan', 'kitchen_area',
       'locality_name', 'airports_nearest', 'cityCenters_nearest',
       'parks_around3000', 'ponds_around3000', 'days_exposition'],
      dtype='object')
```

### Изменение типов данных столбцов (к оглавлению)

```
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23252 entries, 0 to 23251
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype
---  ------                --------------  -----
 0   index                 23252 non-null  int64
 1   total_images          23252 non-null  int64
 2   last_price            23252 non-null  float64
 3   total_area            23252 non-null  float64
 4   first_day_exposition  23252 non-null  object
 5   rooms                 23252 non-null  int64
 6   ceiling_height        23252 non-null  float64
 7   floors_total          23252 non-null  float64
 8   living_area           23252 non-null  float64
 9   floor                 23252 non-null  int64
 10  studio                23252 non-null  bool
 11  open_plan             23252 non-null  bool
 12  kitchen_area          23252 non-null  float64
 13  locality_name         23252 non-null  object
 14  airports_nearest      23252 non-null  float64
 15  cityCenters_nearest   23252 non-null  float64
 16  parks_around3000      23252 non-null  float64
 17  ponds_around3000      23252 non-null  float64
 18  days_exposition       23252 non-null  float64
dtypes: bool(2), float64(11), int64(4), object(2)
memory usage: 3.1+ MB
```

**Изменим типы данных таким образом**

- total_images ✓
- last_price float64 ➡ int64 (Стоимость в рублях)
- total_area float64 ➡ int64 (Площадь в квадратных метрах)
- first_day_exposition ✓
- rooms ✓
- ceiling_height ✓
- floors_total float64 ➡ int64 (Кол-во этажей целочислено)
- living_area float64 ➡ int64 (Жилая площадь в квадратных метрах)
- floor ✓
- studio ✓
- open_plan ✓
- kitchen_area float64 ➡ int64 (Площадь в квадратных метрах)
- locality_name ✓
- airports_nearest float64 ➡ int64 (Расстояние в метрах)
- cityCenters_nearest float64 ➡ int64 (Расстояние в метрах)
- parks_around3000 float64 ➡ int64 (Кол-во парков целочислено)
- ponds_around3000 float64 ➡ int64 (Кол-во прудов целочислено)
- days_exposition float64 ➡ int64 (Кол-во дней целочислено)

```
data['last_price'] = data['last_price'].astype(int)
data['total_area'] = data['total_area'].astype(int)
data['floors_total'] = data['floors_total'].astype(int)
data['living_area'] = data['living_area'].astype(int)
data['kitchen_area'] = data['kitchen_area'].astype(int)
data['airports_nearest'] = data['airports_nearest'].astype(int)
data['cityCenters_nearest'] = data['cityCenters_nearest'].astype(int)
data['parks_around3000'] = data['parks_around3000'].astype(int)
data['ponds_around3000'] = data['ponds_around3000'].astype(int)
data['days_exposition'] = data['days_exposition'].astype(int)
data.info()
```

```
<class 'pandas.core.frame.DataFrame'>
RangeIndex: 23252 entries, 0 to 23251
Data columns (total 19 columns):
 #   Column                Non-Null Count  Dtype  
---  ------                --------------  -----  
 0   index                 23252 non-null  int64  
 1   total_images          23252 non-null  int64  
 2   last_price            23252 non-null  int64  
 3   total_area            23252 non-null  int64  
 4   first_day_exposition  23252 non-null  object 
 5   rooms                 23252 non-null  int64  
 6   ceiling_height        23252 non-null  float64
 7   floors_total          23252 non-null  int64  
 8   living_area           23252 non-null  int64  
```

```
 9   floor                 23252 non-null  int64
 10  studio                23252 non-null  bool
 11  open_plan             23252 non-null  bool
 12  kitchen_area          23252 non-null  int64
 13  locality_name         23252 non-null  object
 14  airports_nearest      23252 non-null  int64
 15  cityCenters_nearest   23252 non-null  int64
 16  parks_around3000      23252 non-null  int64
 17  ponds_around3000      23252 non-null  int64
 18  days_exposition       23252 non-null  int64
dtypes: bool(2), float64(1), int64(14), object(2)
memory usage: 3.1+ MB
```

```
data = data.head(1000)
```

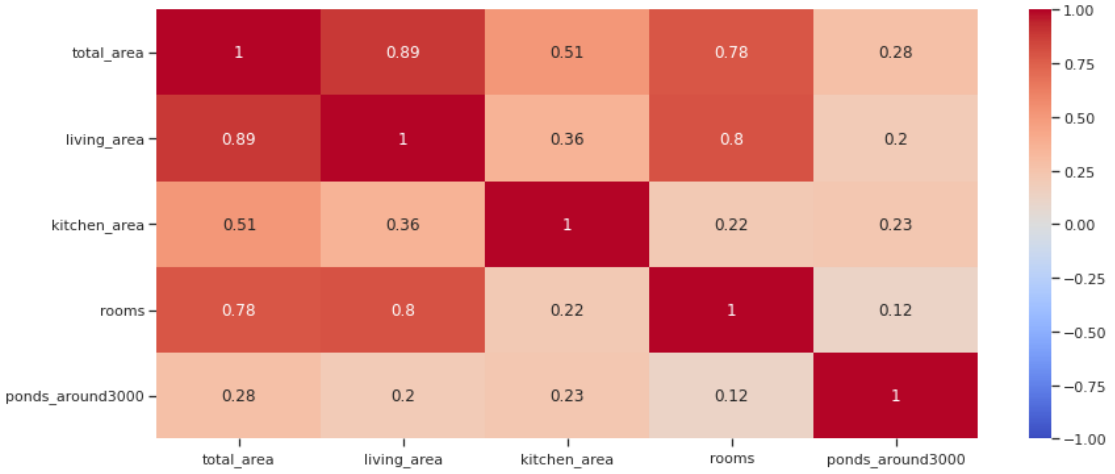
## Корреляционный анализ

```
print('Признаки, имеющие максимальную по модулю корреляцию с ценой квартиры')
best_params = data.corr()
['last_price'].map(abs).sort_values(ascending=False)[1:]
best_params = best_params[best_params.values > 0.35]
best_params
```

```
Признаки, имеющие максимальную по модулю корреляцию с ценой квартиры

total_area          0.773440
living_area         0.636271
kitchen_area        0.557340
rooms               0.445844
ponds_around3000    0.354214
Name: last_price, dtype: float64
```

```
plt.figure(figsize=(14, 6))
sns.heatmap(data[best_params.index].corr(), vmin=-1, vmax=1,
cmap='coolwarm', annot=True)
plt.show()
```



```
plt.figure(figsize=(6, 3))
sns.heatmap(pd.DataFrame(data[np.append(best_params.index.values,
'last_price')].corr()['last_price'].sort_values(ascending=False)[1:]),
vmin=-1, vmax=1, cmap='coolwarm', annot=True)
plt.show()
```


## Разделение выборки на обучающую и тестовую

```
y = data['last_price']
X = data[best_params.index]
x_train, x_test, y_train, y_test = train_test_split(X, y,
test_size=0.3, random_state=3)
```

## Масштабирование данных

```
scaler = StandardScaler().fit(x_train)
x_train_scaled = pd.DataFrame(scaler.transform(x_train),
columns=x_train.columns)
x_test_scaled = pd.DataFrame(scaler.transform(x_test),
columns=x_train.columns)
```

## Метрики

```
def print_metrics(y_test, y_pred):
    print(f"R^2: {r2_score(y_test, y_pred)}")
    print(f"MSE: {mean_squared_error(y_test, y_pred)}")
    print(f"MAE: {mean_absolute_error(y_test, y_pred)}")
```

## Модель №1: Случайный лес

```
print_metrics(y_test,
RandomForestRegressor(random_state=17).fit(x_train,
y_train).predict(x_test))
```

```
R^2: 0.6060466993184502
MSE: 52092031583290.516
MAE: 2503470.347848148
```

### Подбор гиперпараметров

```
rf = RandomForestRegressor(random_state=17)
params = {'n_estimators': [1, 5, 10, 50, 100], 'criterion':
['squared_error', 'absolute_error', 'poisson'],
          'max_features': ['auto', 'sqrt'], 'min_samples_leaf': [1, 3,
5]}
grid_cv = GridSearchCV(estimator=rf, cv=5, param_grid=params, n_jobs=-
1, scoring='neg_mean_absolute_error')
grid_cv.fit(x_train, y_train)
print(grid_cv.best_params_)
```

```
{'criterion': 'absolute_error', 'max_features': 'auto',
'min_samples_leaf': 3, 'n_estimators': 10}
```

```
best_rf = grid_cv.best_estimator_
best_rf.fit(x_train, y_train)
y_pred_rf = best_rf.predict(x_test)
print_metrics(y_test, y_pred_rf)
```

```
R^2: 0.571588219654472
MSE: 56648440192795.0
MAE: 2425045.487833333
```

## Модель №2: Градиентный бустинг

```
print_metrics(y_test,
GradientBoostingRegressor(random_state=17).fit(x_train,
y_train).predict(x_test))
```

```
R^2: 0.4724745284477533
MSE: 69754139583420.016
MAE: 2595754.1778003774
```

### Подбор гиперпараметров

```
gb = GradientBoostingRegressor(random_state=17)
params = {'loss': ['squared_error', 'absolute_error', 'huber'],
'n_estimators': [1, 10, 100],
          'criterion': ['squared_error'], 'min_samples_leaf': [1, 3,
5]}
grid_cv = GridSearchCV(estimator=gb, cv=5, param_grid=params, n_jobs=-
1, scoring='r2')
grid_cv.fit(x_train, y_train)
print(grid_cv.best_params_)
```

```
{'criterion': 'squared_error', 'loss': 'huber', 'min_samples_leaf': 5,
'n_estimators': 100}
```

```
best_gb = grid_cv.best_estimator_
best_gb.fit(x_train, y_train)
y_pred_gb = best_gb.predict(x_test)
print_metrics(y_test, y_pred_gb)
```

```
R^2: 0.573547968080554
MSE: 56389304714731.62
MAE: 2386928.1786402967
```

### Модель №3: Стекинг

```
dataset = Dataset(x_train, y_train, x_test)

model_lr = Regressor(dataset=dataset, estimator=LinearRegression,
name='lr')
model_rf = Regressor(dataset=dataset, estimator=RandomForestRegressor,

                     parameters={'criterion': 'absolute_error',
'n_estimators': 100, 'random_state': 17}, name='rf')
model_gb = Regressor(dataset=dataset,
estimator=GradientBoostingRegressor,
                     parameters={'loss': 'huber', 'random_state': 17},
name='rf')

pipeline = ModelsPipeline(model_lr, model_rf)
stack_ds = pipeline.stack(k=10, seed=1)
stacker = Regressor(dataset=stack_ds,
estimator=GradientBoostingRegressor)
results = stacker.validate(k=10, scorer=mean_absolute_error)
```

```
Metric: mean_absolute_error
Folds accuracy: [2287815.499443283, 1779067.3609490092,
2102662.19077094, 2988468.1277450477, 1537381.131933341,
2647001.1933625233, 2345522.1892407974, 1998635.6350704292,
3421770.9501033765, 1647101.6298290598]
Mean accuracy: 2275542.590844781
Standard Deviation: 570789.3362953931
Variance: 325800466428.5353
```

```
y_pred_stack = stacker.predict()
print_metrics(y_test, y_pred_stack)
```

```
R^2: 0.6630212586234335
MSE: 44558345388442.88
MAE: 2366080.3002260188
```

### Модель №4: Многослойный персептрон

```
print_metrics(y_test, MLPRegressor(random_state=17).fit(x_train,
y_train).predict(x_test))
```

```
R^2: -0.4085569010542929
MSE: 186252001061146.12
MAE: 7357758.667997818
```

#### Подбор гиперпараметров

```
best_mlp = grid_cv.best_estimator_
best_mlp.fit(x_train, y_train)
```

```
y_pred_mlp = best_mlp.predict(x_test)
print_metrics(y_test, y_pred_mlp)
```

```
R^2: 0.49624820930798264
MSE: 66610570708429.91
MAE: 2880865.367704884
```

## Модель №5: Метод группового учёта аргументов

## Сравнение моделей

```
print("Случайный лес")
print_metrics(y_test, y_pred_rf)

print("\nГрадиентный бустинг")
print_metrics(y_test, y_pred_gb)

print("\nСтекинг")
print_metrics(y_test, y_pred_stack)

print("\nМногослойный персептрон")
print_metrics(y_test, y_pred_mlp)

print("\nМетод группового учёта аргументов")
print_metrics(y_test, y_pred_gm)
```

```
Случайный лес
R^2: 0.571588219654472
MSE: 56648440192795.0
MAE: 2425045.487833333

Градиентный бустинг
R^2: 0.573547968080554
MSE: 56389304714731.62
MAE: 2386928.1786402967

Стекинг
R^2: 0.6630212586234335
MSE: 44558345388442.88
MAE: 2366080.3002260188

Многослойный персептрон
R^2: 0.49624820930798264
MSE: 66610570708429.91
MAE: 2880865.367704884

Метод группового учёта аргументов

---------------------------------------------------------------------------
-----
NameError                                 Traceback (most recent call
```

```
last)
Input In [190], in <cell line: 14>()
     11 print_metrics(y_test, y_pred_mlp)
     13 print("\nМетод группового учёта аргументов")
---> 14 print_metrics(y_test, y_pred_gm)

NameError: name 'y_pred_gm' is not defined
```