Министерство науки и высшего образования Российской Федерации
Федеральное государственное бюджетное образовательное учреждение высшего образования
«Московский государственный технический университет имени Н.Э. Баумана (национальный исследовательский университет)»
(МГТУ им. Н.Э. Баумана)

Факультет «Информатика и системы управления»

Кафедра «Системы обработки информации и управления»

ОТЧЁТ ПО

Лабораторной работе №6

Выполнил:	Проверил:
студент группы ИУ5-65Б	преподаватель каф. ИУ5
Кулешова И. А.	Гапанюк Ю. Е.
Подпись и дата:	Подпись и дата:

Москва

2024

In [3]:
```
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LinearRegression
%matplotlib inline
sns.set(style="ticks")
```

Загрузка данных и первичный анализ

In [4]:
```
df = pd.read_csv('medical_insurance.csv', sep=",")
```

In [5]:
```
df
```

Out[5]:

	age	sex	bmi	children	smoker	region	charges
0	19	female	27.900	0	yes	southwest	16884.92400
1	18	male	33.770	1	no	southeast	1725.55230
2	28	male	33.000	3	no	southeast	4449.46200
3	33	male	22.705	0	no	northwest	21984.47061
4	32	male	28.880	0	no	northwest	3866.85520
...	...	...	...	...	...	...	...
2767	47	female	45.320	1	no	southeast	8569.86180
2768	21	female	34.600	0	no	southwest	2020.17700
2769	19	male	26.030	1	yes	northwest	16450.89470
2770	23	male	18.715	0	no	northwest	21595.38229
2771	54	male	31.600	0	no	southwest	9850.43200

2772 rows × 7 columns

In [4]:
```
df.shape
```

Out[4]:
```
(2772, 7)
```

In [5]:
```
# ищем пропуски
df.isna().sum()
```

Out[5]:
```
age         0
sex         0
bmi         0
children    0
smoker      0
region      0
charges     0
dtype: int64
```

In [6]:
```
df.dtypes
```

Out[6]:
```
age           int64
sex          object
bmi         float64
children      int64
smoker       object
region       object
charges     float64
dtype: object
```

In [6]:
```
categorical_cols=df.select_dtypes(include=object).columns.to_list()
categorical_cols
```

Out[6]:
```
['sex', 'smoker', 'region']
```

In [7]:
```
for cat in categorical_cols:
    print(f"column -- {cat}: {df[cat].unique()}")
```

```
column -- sex: ['female' 'male']
column -- smoker: ['yes' 'no']
column -- region: ['southwest' 'southeast' 'northwest' 'northeast']
```

In [8]:
```
from sklearn.preprocessing import LabelEncoder
```

In [9]:
```
for cat in categorical_cols:
    le = LabelEncoder()
    df[cat] = le.fit_transform(df[cat])
```

In [11]:
```
df
```

Out[11]:

	age	sex	bmi	children	smoker	region	charges
0	19	0	27.900	0	1	3	16884.92400
1	18	1	33.770	1	0	2	1725.55230
2	28	1	33.000	3	0	2	4449.46200
3	33	1	22.705	0	0	1	21984.47061
4	32	1	28.880	0	0	1	3866.85520
...	...	...	...	...	...	...	...
2767	47	0	45.320	1	0	2	8569.86180
2768	21	0	34.600	0	0	3	2020.17700
2769	19	1	26.030	1	1	1	16450.89470
2770	23	1	18.715	0	0	1	21595.38229
2771	54	1	31.600	0	0	3	9850.43200

2772 rows × 7 columns

In [12]:
```
df.dtypes
```

```
Out[12]: age           int64
         sex           int64
         bmi         float64
         children      int64
         smoker        int64
         region        int64
         charges     float64
         dtype: object
```

```
In [13]: for cat in categorical_cols:
             print(f"column -- {cat}: {df[cat].unique()}")
```

```
column -- sex: [0 1]
column -- smoker: [1 0]
column -- region: [3 2 1 0]
```

```
In [10]: X = df.drop('charges', axis=1)  # Замените 'целевая_переменная' на название ваше
         y = df['charges']
```

```
In [11]: # Формирование обучающей и тестовой выборки
         X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.2, randc
```

```
In [16]: #Построим корреляционную матрицу
         fig, ax = plt.subplots(figsize=(15,7))
         sns.heatmap(df.corr(method='pearson'), ax=ax, annot=True, fmt='.2f')
```

```
Out[16]: <Axes: >
```


```
In [12]: from sklearn.preprocessing import StandardScaler
         from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score
```

```
In [13]: # Scaling the features
         scaler = StandardScaler()
         X_train_scale = scaler.fit_transform(X_train)
         X_test_scale = scaler.transform(X_test)

         # Converting the scaled arrays into DataFrames
         X_train = pd.DataFrame(X_train_scale, columns=X_train.columns)
         X_test = pd.DataFrame(X_test_scale, columns=X_test.columns)
```

In [14]:
```
X_train.shape
```
Out[14]: (2217, 6)

In [15]:
```
X_test.shape
```
Out[15]: (555, 6)

In [25]:
```
def create_df(data,models, cols):
    index = []
    for model in models:
        model_name = type(model).__name__
        if model_name in index:
            model_name = str(type(model).__name__) + '_hyp'
        index.append(model_name)

    df = pd.DataFrame(data = data,
                      index = index)
    df.rename(columns=dict(zip(df.columns, cols)), inplace=True)
    return df

def training(models, X=X_train, y=y_train):
    metric = {}
    train_model = []
    mses =[]
    maes =[]
    r2s =[]
    index =[]
    for model in models:
        #score = []  # Initialize score for each model
        model.fit(X, y)
        train_model.append(model)
        y_pred = model.predict(X)
        mse = mean_squared_error(y, y_pred)
        mses.append(mse)
        mae = mean_absolute_error(y, y_pred)
        maes.append(mae)
        r2 = r2_score(y, y_pred)*100
        r2s.append(r2)
        #score.extend([mse, mae, r2])  # Use extend to add multiple elements to

    cols=['train_mse', 'train_mae', 'train_r2']
    metric['mse'] = mses
    metric['mae'] = maes
    metric['r2'] = r2s
    metric_df = create_df(data=metric,models= train_model, cols = cols)
    return metric_df, train_model

def testing(models,X = X_test, y = y_test):
    mses =[]
    maes =[]
    r2s =[]
    index =[]
    metric = {}
    for model in models:
        #score = []  # Initialize score for each model
        y_pred = model.predict(X)
        mse = mean_squared_error(y, y_pred)
        mses.append(mse)
```

```
        mae = mean_absolute_error(y, y_pred)
        maes.append(mae)
        r2 = r2_score(y, y_pred)*100
        r2s.append(r2)
        #score.extend([mse, mae, r2])  # Use extend to add multiple elements to
    metric['mse'] = mses
    metric['mae'] = maes
    metric['re'] = r2s
    cols=['test_mse', 'test_mae', 'test_r2']
    metric_df = create_df(data=metric,models= models, cols=cols)
    return metric_df
```

In [17]:
```
from sklearn.ensemble import RandomForestRegressor
from sklearn.tree import export_graphviz
from sklearn.ensemble import ExtraTreesRegressor
from sklearn.tree import DecisionTreeRegressor
from sklearn.ensemble import AdaBoostRegressor
from sklearn.ensemble import GradientBoostingRegressor
base_model = DecisionTreeRegressor(max_depth=3)
rf = RandomForestRegressor(n_estimators=5, oob_score=True, random_state=42)
etr = ExtraTreesRegressor(n_estimators=5, oob_score=True, random_state=42, boots
abr = AdaBoostRegressor(base_estimator=base_model, n_estimators=5, random_state=
gbr = GradientBoostingRegressor(n_estimators=200, learning_rate=0.1, random_stat
```

In [23]:
```
pd.options.display.float_format = '{:.3f}'.format
models = [rf, etr, abr, gbr]
training_df, train_models = training(models)

training_df
```

```
/home/sofi_flin/Documents/ml_spring_23/.venv/lib/python3.10/site-packages/sklear
n/ensemble/_forest.py:583: UserWarning: Some inputs do not have OOB scores. This
probably means too few trees were used to compute any reliable OOB estimates.
  warn(
/home/sofi_flin/Documents/ml_spring_23/.venv/lib/python3.10/site-packages/sklear
n/ensemble/_forest.py:583: UserWarning: Some inputs do not have OOB scores. This
probably means too few trees were used to compute any reliable OOB estimates.
  warn(
/home/sofi_flin/Documents/ml_spring_23/.venv/lib/python3.10/site-packages/sklear
n/ensemble/_base.py:166: FutureWarning: `base_estimator` was renamed to `estimato
r` in version 1.2 and will be removed in 1.4.
  warnings.warn(
```

Out[23]:

	train_mse	train_mae	train_r2
RandomForestRegressor	2179569.176	498.927	98.508
ExtraTreesRegressor	2118513.707	493.815	98.550
AdaBoostRegressor	22643419.217	3679.647	84.500
GradientBoostingRegressor	10957994.903	1769.074	92.499

In [24]:
```
test_df = testing(train_models)

test_df
```

Out[24]:

	test_mse	test_mae	test_r2
RandomForestRegressor	9444619.196	1343.661	93.846
ExtraTreesRegressor	10098510.710	1416.470	93.420
AdaBoostRegressor	25702298.929	3785.518	83.254
GradientBoostingRegressor	16573386.339	2119.189	89.202

In [ ]:

```
from sklearn.metrics import mean_absolute_error, r2_score

def val_metrics(model, train_x, train_y, test_x, test_y):
    model.fit(train_x, train_y)
    y_pred = model.predict(test_x)
    mae = mean_absolute_error(test_y, y_pred)
    r2 = r2_score(test_y, y_pred)
    print(model)
    print('MAE={}'.format(mae))
    print('R2 Score={}'.format(r2))
    print('==========================')

# Точность на отдельных моделях
for model in [
    LinearRegression(),
    DecisionTreeRegressor(random_state=42),
    RandomForestRegressor(n_estimators=105, random_state=42)
]:
    val_metrics(model, X_train, y_train, X_test, y_test)
```

In [21]:

```
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()
```

Стекинг

In [1]:

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

In [19]:

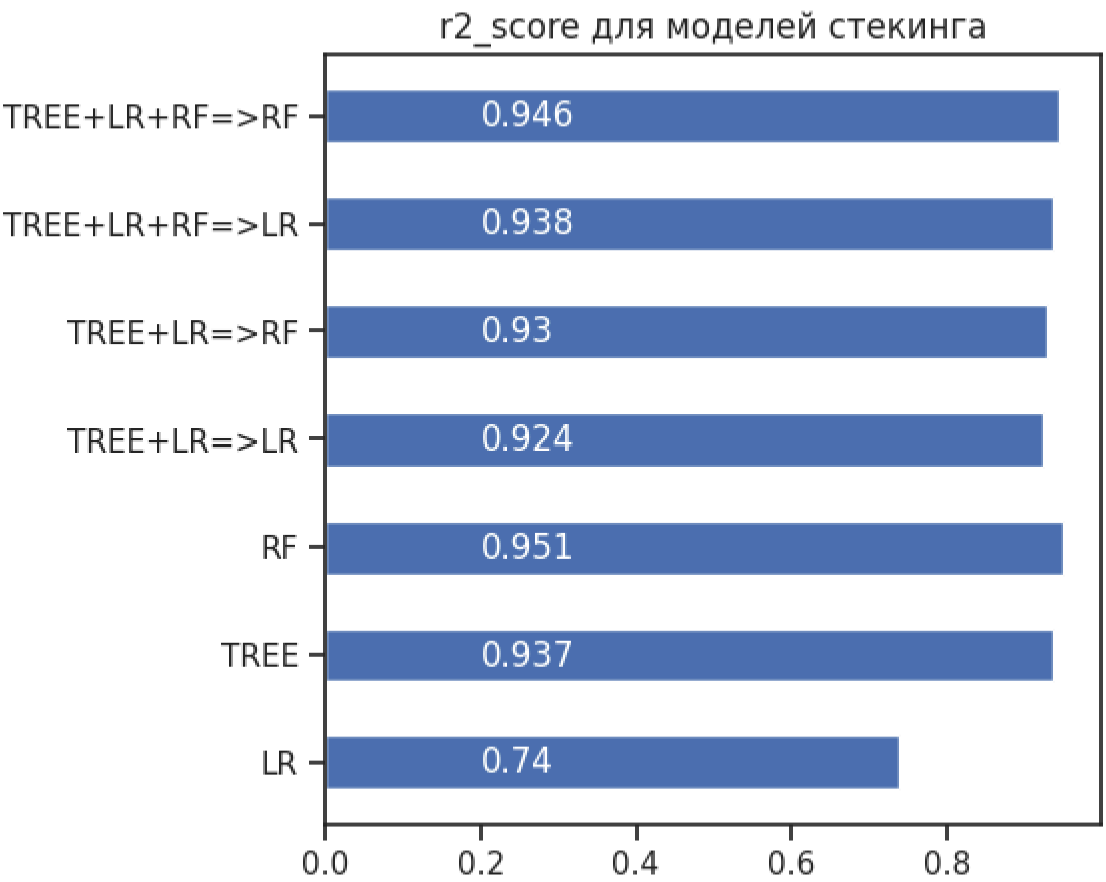
```
dataset = Dataset(X_train, y_train, X_test)

# модели первого уровня
model_tree = Regressor(dataset=dataset, estimator=DecisionTreeRegressor, name='t
model_lr = Regressor(dataset=dataset, estimator=LinearRegression,name='lr')
model_rf = Regressor(dataset=dataset, estimator=RandomForestRegressor, parameter
# 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=r2_score)
# 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=r2_score)

# 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=r2_score)

# 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=r2_score)
```

```
Metric: r2_score
Folds accuracy: [0.9604155834396174, 0.9676783124950149, 0.9143296216328749, 0.91
03283633184968, 0.9062750691257013, 0.9046892401277392, 0.9013266179105077, 0.931
0084850393554, 0.9364865736895556, 0.9057589689509976]
Mean accuracy: 0.923829683572986
Standard Deviation: 0.022919334557578944
Variance: 0.0005252958965622324
Metric: r2_score
Folds accuracy: [0.95138797499365, 0.9773912871236708, 0.9223681093091471, 0.8994
528281296649, 0.931307679439366, 0.8887400560499181, 0.9325782332837739, 0.930091
929662296, 0.9407304192542247, 0.9227460307492747]
Mean accuracy: 0.9296794547994987
Standard Deviation: 0.023605588845811064
Variance: 0.0005572238247574797
Metric: r2_score
Folds accuracy: [0.9630977928143095, 0.9697698899275307, 0.929893193940304, 0.913
0782516187702, 0.9321605448835882, 0.9105340558878181, 0.9377084214841552, 0.9483
430210608826, 0.9585861697773104, 0.9208689906648586]
Mean accuracy: 0.9384040332059527
Standard Deviation: 0.019842401115734324
Variance: 0.0003937208820376948
Metric: r2_score
Folds accuracy: [0.9600171377688029, 0.9680327777400004, 0.9420996007095698, 0.91
15444844797232, 0.9436416680269085, 0.9051533218104206, 0.9623257768695322, 0.953
8378378720032, 0.9631040899608148, 0.9462673652162642]
Mean accuracy: 0.9456024060454041
Standard Deviation: 0.02043853032950707
Variance: 0.00041773352203018047
```

In [22]:
```
# Результаты
array_labels = ['LR','TREE', 'RF', 'TREE+LR=>LR',
```

```
                'TREE+LR=>RF', 'TREE+LR+RF=>LR', 'TREE+LR+RF=>RF']
array_r2 = [0.7398864322395977, 0.9370575488858102, 0.9512316214891919,
            0.923829683572986, 0.9296794547994987, 0.9384040332059527,
            0.9456024060454041]
# Визуализация результатов
vis_models_quality(array_r2, array_labels, 'r2_score для моделей стекинга')
```



Вывод: лучший результат стекинга - модель 1 уровня (дерево, линейная регрессия, случайный лес)

Многослойный перцептрон

In [23]:
```
from sklearn.neural_network import MLPRegressor
mlp = MLPRegressor(hidden_layer_sizes=(300, 200, 150, 100, 50, 25),  activation=
mlp_model = MLPRegressor(hidden_layer_sizes=(100, 50), activation='relu', solver
```

In [26]:
```
pd.options.display.float_format = '{:.3f}'.format
models = [mlp, mlp_model]
training_df, train_models = training(models)

training_df
```

```
/home/sofi_flin/Documents/ml_spring_23/.venv/lib/python3.10/site-packages/sklear
n/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Op
timizer: Maximum iterations (200) reached and the optimization hasn't converged y
et.
  warnings.warn(
/home/sofi_flin/Documents/ml_spring_23/.venv/lib/python3.10/site-packages/sklear
n/neural_network/_multilayer_perceptron.py:686: ConvergenceWarning: Stochastic Op
timizer: Maximum iterations (200) reached and the optimization hasn't converged y
et.
  warnings.warn(
```

Out[26]:

	train_mse	train_mae	train_r2
MLPRegressor	16020791.730	2179.870	89.033
MLPRegressor_hyp	32329168.897	3886.080	77.869

In [27]:

```
test_df = testing(train_models)

test_df
```

Out[27]:

	test_mse	test_mae	test_r2
MLPRegressor	21596791.958	2505.951	85.929
MLPRegressor_hyp	36640353.847	3996.691	76.127

COMBI и MIA из семейства МГУА

In [28]:

```
from gmdhpy import gmdh
```

In [29]:

```
import gmdh
```

In [30]:

```
laptop_x_train, laptop_x_test, laptop_y_train, laptop_y_test = \
    gmdh.split_data(df.drop(['charges'], axis=1), df['charges'], test_size=0.8)
laptop_scaler = StandardScaler().fit(laptop_x_train)
laptop_x_train = laptop_scaler.transform(laptop_x_train)
laptop_x_test = laptop_scaler.transform(laptop_x_test)
print("train elements:", laptop_x_train.shape[0], "\ntest elements:", laptop_x_t
```

```
train elements: 554
test elements: 2218
```

In [31]:

```
def print_metrics(y_test, y_pred, squared=False):
    print(f"R^2: {r2_score(y_test, y_pred)}")
    crit_name = "MSE" if squared else "RMSE"
    print(f"{crit_name}: {mean_squared_error(y_test, y_pred, squared=squared)}")
    print(f"MAE: {mean_absolute_error(y_test, y_pred)}")
```

In [32]:

```
from sklearn.metrics import mean_absolute_error, mean_squared_error, r2_score

def print_metrics(y_test, y_pred):
    mse = mean_squared_error(y_test, y_pred)
    mae = mean_absolute_error(y_test, y_pred)
    r2 = r2_score(y_test, y_pred)

    print(f"MAE: {mae}")
```

```
    print(f"MSE: {mse}")
    print(f"R2: {r2}")
```

In [33]:

```
laptop_combi_model = gmdh.Combi()
laptop_combi_model.fit(laptop_x_train, laptop_y_train, verbose=1, n_jobs=-1, tes
                       criterion=gmdh.Criterion(gmdh.CriterionType.REGULARITY))
print()
print(laptop_combi_model.get_best_polynomial())
print()
laptop_y_pred_combi = laptop_combi_model.predict(laptop_x_test)
print_metrics(laptop_y_test, laptop_y_pred_combi)
```

```
LEVEL 1  [=========================] 100% [00m:00s] (6 combinations) error=6.8707
9e+09
LEVEL 2  [=========================] 100% [00m:00s] (15 combinations) error=5.225
29e+09
LEVEL 3  [=========================] 100% [00m:00s] (20 combinations) error=4.772
24e+09
LEVEL 4  [=========================] 100% [00m:00s] (15 combinations) error=4.768
17e+09
LEVEL 5  [=========================] 100% [00m:00s] (6 combinations) error=4.8628
7e+09

y = 3929.2597*x1 + 1927.5676*x3 + 9429.0544*x5 - 16.6626*x6 + 13415.4701

MAE: 4281.452207884143
MSE: 37361873.09104982
R2: 0.7475026136697335
```

In [34]:

```
laptop_mia_model = gmdh.Mia()
laptop_mia_model.fit(laptop_x_train, laptop_y_train, verbose=1, n_jobs=-1, test_
                     criterion=gmdh.Criterion(gmdh.CriterionType.SYM_REGULARITY)
                     polynomial_type=gmdh.PolynomialType.LINEAR)
laptop_y_pred_mia = laptop_mia_model.predict(laptop_x_test)
print_metrics(laptop_y_test, laptop_y_pred_mia)
```

```
LEVEL 1  [=========================] 100% [00m:00s] (15 combinations) error=2.307
06e+1
LEVEL 2  [=========================] 100% [00m:00s] (36 combinations) error=2.096
75e+1
LEVEL 3  [=========================] 100% [00m:00s] (36 combinations) error=2.091
11e+1
LEVEL 4  [=========================] 100% [00m:00s] (36 combinations) error=2.089
61e+1
LEVEL 5  [=========================] 100% [00m:00s] (36 combinations) error=2.089
81e+1
MAE: 4255.538959422512
MSE: 36938166.8105884
R2: 0.7503660870327393
```

Вывод: модель стекинга регрессия и методы из библиотеки МГУА дали худшие результаты, а модель стекинга дерево решений, линейная регрессия, случайный лес лучший результат.