# 1. Рубежный контроль №2 Наседкин Игорь ИУ5-23М

## 1.1. Тема: Методы построения моделей машинного обучения.

### 1.1.1. Задача 2. Выполнение классификации/регрессии/кластеризации данных (по вариантам).

Для заданного набора данных решите задачу кластеризации с использованием методов 1) K-Means, 2) DBSCAN и 3) Birch. Оцените качество модели на основе подходящих метрик качества (не менее двух метрик, если это возможно). Какие метрики качества Вы использовали и почему? Какие выводы Вы можете сделать о качестве построенных моделей?

Набор данных - https://www.kaggle.com/ronitf/heart-disease-uci

age;—возраст;

sex;—пол;

chest pain type (4 values);—Тип боли;

resting blood pressure;—Кровяное давление в покое;

serum cholestoral in mg/dl;—Холестерин;

fasting blood sugar > 120 mg/dl;—Сахар в крови;

resting electrocardiographic results (values 0,1,2);—Электрокардиография в покое;

maximum heart rate achieved;—Максимальный сердечный ритм;

exercise induced angina;—Стенокардия вызванная физической нагрузкой;

oldpeak = ST depression induced by exercise relative to rest;—депрессия вызванная физ упражнениями;

the slope of the peak exercise ST segment;—Наклон пика упражнений;

number of major vessels (0-3) colored by flourosopy;—Кол-во крупных сосоудов по цвету thal: 3 = normal; 6 = fixed defect; 7 = reversable defect;

The "goal" field refers to the presence of heart disease in the patient. It is integer valued from 0 (no presence) to 4.

Настройка:

[2]:
```
from sklearn.metrics import accuracy_score, balanced_accuracy_score
import numpy as np
from sklearn.model_selection import train_test_split
import pandas as pd

from typing import Dict, Tuple
from scipy import stats
from IPython.display import Image
from sklearn import cluster, datasets, mixture
from sklearn.neighbors import kneighbors_graph
from sklearn.preprocessing import StandardScaler
from sklearn.metrics import adjusted_rand_score
from sklearn.metrics import adjusted_mutual_info_score
from sklearn.metrics import homogeneity_completeness_v_measure
from sklearn.metrics import silhouette_score
from itertools import cycle, islice
import seaborn as sns
import matplotlib.pyplot as plt
%matplotlib inline
sns.set(style="ticks")
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19:
FutureWarning: pandas.util.testing is deprecated. Use the functions in the
public API at pandas.testing instead.
  import pandas.util.testing as tm
```

[0]:
```python
data = pd.read_csv('heart.csv')
```

[5]:
```python
data.head()
```

[5]:
```
   age  sex  cp  trestbps  chol  fbs  …  exang  oldpeak  slope  ca  thal
target
0   63    1   3       145   233    1  …      0      2.3      0   0     1
1
1   37    1   2       130   250    0  …      0      3.5      0   0     2
1
2   41    0   1       130   204    0  …      0      1.4      2   0     2
1
3   56    1   1       120   236    0  …      0      0.8      2   0     2
1
4   57    0   0       120   354    0  …      1      0.6      2   0     2
1

[5 rows x 14 columns]
```

[6]:
```python
data.dtypes
```

[6]:
```
age           int64
sex           int64
cp            int64
trestbps      int64
chol          int64
fbs           int64
restecg       int64
thalach       int64
exang         int64
oldpeak     float64
slope         int64
ca            int64
thal          int64
target        int64
dtype: object
```

[7]:
```python
data.isnull().sum()
```

[7]:
```
age         0
sex         0
cp          0
trestbps    0
chol        0
fbs         0
restecg     0
```

```
thalach     0
exang       0
oldpeak     0
slope       0
ca          0
thal        0
target      0
dtype: int64
```

пропусков нет

Делим датасет на тестовую и обучающую выборки

[0]:
```
col_target='target'
```

[0]:
```
x = data.drop(col_target,axis = 1).values
y = data['target'].values
#x_train, x_test, y_train, y_test = train_test_split(x, y, test_size=0.33,␣
↪random_state=324)
```

[11]:
```
x
```

[11]:
```
array([[63.,  1.,  3., …,  0.,  0.,  1.],
       [37.,  1.,  2., …,  0.,  0.,  2.],
       [41.,  0.,  1., …,  2.,  0.,  2.],
       …,
       [68.,  1.,  0., …,  1.,  2.,  3.],
       [57.,  1.,  0., …,  1.,  1.,  3.],
       [57.,  0.,  1., …,  1.,  1.,  2.]])
```

[12]:
```
y
```

[12]:
```
array([1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1,
       1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0,
       0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0])
```

[14]:
```
print("Dataset: ", x.shape)
print("Marks: ", y.shape)
```

```
Dataset:  (303, 13)
Marks:  (303,)
```

[0]:
```
def visualize_clusters(x, y):
    """
    Визуализация результатов кластерного анализа
    """
    plt.subplots(figsize=(10,7))
    plot_num = 0
    for X, y_pred in zip(x, y):
        plot_num += 1
        plt.subplot(2, 3, plot_num)
        # Цвета точек как результат кластеризации
        colors = np.array(list(islice(cycle(['#377eb8', '#ff7f00',␣
↪'#4daf4a',
                                             '#f781bf', '#a65628',␣
↪'#984ea3',
                                             '#999999', '#e41a1c',␣
↪'#dede00']),
                                      int(max(y_pred) + 1))))
        # черный цвет для выделяющихся значений
        colors = np.append(colors, ["#000000"])
        plt.scatter(X[:, 0], X[:, 1], s=3, color=colors[y_pred])
        plt.xlim(-2.5, 2.5)
        plt.ylim(-2.5, 2.5)
        plt.xticks(())
        plt.yticks(())
        plt.title(datasets_names[plot_num-1])

    plt.show()
```

[0]:
```
def do_clustering(x, method):
    """
    Выполнение кластеризации для данных примера
    """
    cluster_results = []
    for X in x:
        temp_cluster = method.fit_predict(X)
        cluster_results.append(temp_cluster)
    return cluster_results
```

[0]:
```
from sklearn.cluster import KMeans, MiniBatchKMeans
```

**Метрики качества кластеризации**
Adjusted Rand index
Adjusted Mutual Information
Homogeneity, completeness, V-measure
Коэффициент силуэта

[0]:
```
import warnings
warnings.simplefilter(action='ignore', category=FutureWarning)

def claster_metrics(method, cluster_datasets, cluster_true_y):
    """
```

```
    Вычисление метрик кластеризации
    """
    ari = []
    ami = []
    hl = []
    cl = []
    vl = []
    sl = []
    for X, true_y in zip(cluster_datasets, cluster_true_y):
        temp_cluster = method.fit_predict(X)
        ari.append(adjusted_rand_score(true_y, temp_cluster))
        ami.append(adjusted_mutual_info_score(true_y, temp_cluster))

        h, c, v = homogeneity_completeness_v_measure(true_y, temp_cluster)
        hl.append(h)
        cl.append(c)
        vl.append(v)

        sl.append(silhouette_score(X, temp_cluster))

    result = pd.DataFrame({ 'ARI':ari, 'AMI':ami,
                           'Homogeneity':hl,
                           'Completeness':cl,
                           'V-measure':vl, 'Silhouette':sl})
    return result
```

[0]:
```
model = KMeans(n_clusters=2, random_state=1)
```

[0]:
```
model.fit(x)
```

[0]:
```
KMeans(algorithm='auto', copy_x=True, init='k-means++', max_iter=300,
       n_clusters=2, n_init=10, n_jobs=None, precompute_distances='auto',
       random_state=1, tol=0.0001, verbose=0)
```

[0]:
```
all_predictions = model.predict(x)
```

[0]:
```
print(all_predictions)
```

```
[1 1 1 1 0 1 0 0 1 1 1 0 0 1 0 1 0 1 1 1 1 1 1 1 1 0 1 1 0 1 1 1 1 0 1 1 0
 1 0 0 0 1 1 0 0 0 1 1 1 1 1 0 1 1 1 1 1 1 1 0 0 0 1 1 1 1 1 1 1 1 1 1 1 0
 1 1 1 1 1 1 1 0 0 0 0 0 0 1 1 1 1 1 1 0 1 1 0 1 0 1 1 0 1 1 1 1 1 1 1 1 0
 1 0 1 0 1 1 1 1 1 0 0 0 0 1 1 1 0 1 0 1 0 0 1 0 0 1 1 1 0 0 0 1 1 1 1 1 1
 1 1 1 1 1 0 1 1 1 1 1 1 1 0 1 1 1 0 1 0 1 1 1 1 0 1 1 1 1 0 1 0 0 1 0 1 1
 0 1 0 1 1 0 1 1 0 1 0 1 1 0 1 1 0 0 0 1 1 1 0 1 1 1 0 1 0 1 0 0 0 1 1 0 1
 0 0 1 1 0 1 0 0 1 0 0 1 0 0 0 0 0 0 0 1 1 0 1 0 0 1 0 1 0 1 0 0 0 0 0 1 1
 1 1 1 0 0 1 1 0 1 0 0 1 1 1 1 0 1 1 0 0 1 0 1 1 1 1 0 1 1 0 1 1 0 1 1 1 1
 1 1 1 0 1 1 1]
```

[0]:
```
print(y.values)
```

```
[1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1
 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 1 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0 0
 0 0 0 0 0 0 0]
```

[0]:
```
from sklearn.cluster import DBSCAN
```

[0]:
```
dbscan = DBSCAN()
```

[0]:
```
dbscan.fit(x)
```

[0]:
```
DBSCAN(algorithm='auto', eps=0.5, leaf_size=30, metric='euclidean',
       metric_params=None, min_samples=5, n_jobs=None, p=None)
```

## 2. Метод K-средних

[0]:
```
import pandas as pd
from sklearn.cluster import KMeans
from sklearn.preprocessing import MinMaxScaler, StandardScaler
```

[20]:
```
# read data (drop last empty column, caused by an extra (last) colon in␣
↪the header)
data = pd.read_csv('heart.csv', sep=',')

# normalize data
scaler = StandardScaler()
X = scaler.fit_transform(data.drop('target', 1))

# clustering
n_clusters = 2
km = KMeans(n_clusters=n_clusters, random_state=1)

# fit & predict clusters
data['cluster'] = km.fit_predict(X)

# results - we should have 2 clusters: [0,1]
print(data)

# cluster's centroids
print(km.cluster_centers_)
```

```
     age  sex  cp  trestbps  chol  …  slope  ca  thal  target  cluster
0     63    1   3       145   233  …      0   0     1       1        0
```

```
1      37    1  2       130   250  ...     0   0     2       1         0
2      41    0  1       130   204  ...     2   0     2       1         0
3      56    1  1       120   236  ...     2   0     2       1         0
4      57    0  0       120   354  ...     2   0     2       1         0
..    ... ...  ..     ...  ... ...   ... ..   ...   ...    ...
298    57    0  0       140   241  ...     1   0     3       0         1
299    45    1  3       110   264  ...     1   0     3       0         0
300    68    1  0       144   193  ...     1   2     3       0         1
301    57    1  0       130   131  ...     1   1     3       0         1
302    57    0  1       130   236  ...     1   1     2       0         0

[303 rows x 15 columns]
[[-0.24092798 -0.08974737  0.37025174 -0.12677477 -0.06229806 -0.0341701
   0.06200725  0.46180314 -0.43819277 -0.39449405  0.36064556 -0.30877277
  -0.22373865]
 [ 0.45432133  0.16923791 -0.69818899  0.23906099  0.11747634  0.06443504
  -0.11692796 -0.87082878  0.82630636  0.74390307 -0.68007448  0.58225722
   0.42190717]]
```

## 3. Алгоритм DBSCAN

[0]:
```python
from sklearn.cluster import DBSCAN
```

[0]:
```python
eps = 0.25
dbscan = DBSCAN(eps=eps)
```

[0]:
```python
data['cluster'] = dbscan.fit_predict(X)
```

[0]:
```python
col_target='target'
```

[0]:
```python
test = data.drop(col_target,axis = 1)
```

[30]:
```python
print(data)
```

```
     age  sex  cp  trestbps  chol  ...  slope  ca  thal  target  cluster
0     63    1   3       145   233  ...      0   0     1       1       -1
1     37    1   2       130   250  ...      0   0     2       1       -1
2     41    0   1       130   204  ...      2   0     2       1       -1
3     56    1   1       120   236  ...      2   0     2       1       -1
4     57    0   0       120   354  ...      2   0     2       1       -1
..    ... ...  ..     ...  ... ...   ... ..   ...   ...    ...
298    57    0   0       140   241  ...      1   0     3       0       -1
299    45    1   3       110   264  ...      1   0     3       0       -1
300    68    1   0       144   193  ...      1   2     3       0       -1
301    57    1   0       130   131  ...      1   1     3       0       -1
302    57    0   1       130   236  ...      1   1     2       0       -1

[303 rows x 15 columns]
```

## 4. Алгоритм BIRCH

[0]:
```python
from sklearn.cluster import Birch
```

[0]:
```python
birch = Birch()
```

[0]:
```python
data['cluster'] = dbscan.fit_predict(X)
```

[0]:
```python
print(data)
```

```
     age  sex  cp  trestbps  chol  …  slope  ca  thal  target  cluster
0     63    1   3       145   233  …      0   0     1       1        0
1     37    1   2       130   250  …      0   0     2       1        0
2     41    0   1       130   204  …      2   0     2       1        2
3     56    1   1       120   236  …      2   0     2       1        0
4     57    0   0       120   354  …      2   0     2       1        2
..   …  …  ..     …   …  …   …  ..  …   …    …
298   57    0   0       140   241  …      1   0     3       0        2
299   45    1   3       110   264  …      1   0     3       0        0
300   68    1   0       144   193  …      1   2     3       0        1
301   57    1   0       130   131  …      1   1     3       0        1
302   57    0   1       130   236  …      1   1     2       0        2

[303 rows x 15 columns]
```

## 5. Сравнение алгоритмов по метрикам:

[53]:
```python
from sklearn import metrics
import pandas as pd
from sklearn.cluster import KMeans, AgglomerativeClustering,␣
↪AffinityPropagation, SpectralClustering, DBSCAN, Birch

data = pd.read_csv('heart.csv', sep=',')
a = data.drop('target',axis = 1)
b = data['target']

algorithms = []
algorithms.append(KMeans(n_clusters=2, random_state=1))
algorithms.append(DBSCAN(eps=0.25))
algorithms.append(Birch())

data = []
for algo in algorithms:
    algo.fit(a)
    data.append(({
        'ARI': metrics.adjusted_rand_score(b, algo.labels_),
        'AMI': metrics.adjusted_mutual_info_score(b, algo.labels_),
        'Homogenity': metrics.homogeneity_score(b, algo.labels_),
        'Completeness': metrics.completeness_score(b, algo.labels_),
```

```
        'V-measure': metrics.v_measure_score(b, algo.labels_),
        #'Silhouette': metrics.silhouette_score(a, algo.labels_)
        }))

results = pd.DataFrame(data=data, columns=['ARI', 'AMI', 'Homogenity',
                                           'Completeness', 'V-measure',
                                           #'Silhouette'
                                           ],
                       index=['K-means', 'DBSCAN',
                              'Birch'])

results
```

[53]:

| | ARI | AMI | Homogenity | Completeness | V-measure |
|---|---|---|---|---|---|
| K-means | 0.020501 | 0.011402 | 0.013501 | 0.014202 | 0.013843 |
| DBSCAN | 0.000000 | 0.000000 | 0.000000 | 1.000000 | 0.000000 |
| Birch | 0.029682 | 0.019219 | 0.029011 | 0.019004 | 0.022965 |

**Вывод: по большинству метрик выигрывает метод Birch, однако качество его кластеризации не очень хорошее(согласно этим же метрикам). По полноте у DBSCAN большое преимущество, что свидетельствует о хорошем качестве кластеризации.**