НИР ОиАД

Сравнение классификаторов на основе данных о сердечных заболеваний

Гаджиев К.К. ИУ5-33М

Описание признаков heart dataset

age - age in years

sex(1 = male; 0 = female)

cp - chest pain type

trestbps - resting blood pressure (in mm Hg on admission to the hospital)

chol - serum cholestoral in mg/dl

fbs - (fasting blood sugar > 120 mg/dl) (1 = true; 0 = false)

restecg - resting electrocardiographic results

thalach - maximum heart rate achieved

exang - exercise induced angina (1 = yes; 0 = no)

oldpeak - ST depression induced by exercise relative to rest

slope - the slope of the peak exercise ST segment

ca - number of major vessels (0-3) colored by flourosopy

tha - l3 = normal; 6 = fixed defect; 7 = reversable defect

target - 1 or 0

```
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.metrics import classification_report
from sklearn.metrics import accuracy_score, recall_score, f1_score
from sklearn.metrics import make_scorer
from sklearn.model_selection import GridSearchCV
import matplotlib.pyplot as plt
from sklearn.linear_model import LogisticRegression
from sklearn.svm import SVC
from sklearn.tree import DecisionTreeClassifier
```

```
data = pd.read_csv('data/heart.csv', sep=",")

data.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	c
0	63	1	3	145	233	1	0	150	0	2.3	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	

```
# сколько пустых ячеек
for col in data.columns:
    # синтаксис пандаса, надо разобраться
    null_count = data[data[col].isnull()].shape[0]
    print('{} - {}'.format(col, null_count))
```

```
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
```

```
data.dtypes
```

```
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
```

```
# отделим целевой признак от остальных
```

```
X = data.loc[:, data.columns != 'target']
X.head()
```

	age	sex	cp	trestbps	chol	fbs	restecg	thalach	exang	oldpeak	slope	c
0	63	1	3	145	233	1	0	150	0	2.3	0	
1	37	1	2	130	250	0	1	187	0	3.5	0	
2	41	0	1	130	204	0	0	172	0	1.4	2	
3	56	1	1	120	236	0	1	178	0	0.8	2	
4	57	0	0	120	354	0	1	163	1	0.6	2	

```
Y = data['target']
Y.head()
```

```
0    1
1    1
2    1
3    1
4    1
Name: target, dtype: int64
```

```
# разобьем данные на обучающую и тестовую выборки
X_train, X_test, y_train, y_test = train_test_split(
    X, Y, test_size=0.25, random_state=1)
```

```
X_train.shape, y_train.shape
```

```
((227, 13), (227,))
```

```
X_test.shape, y_test.shape
```

```
((76, 13), (76,))
```

```
# если подумать о сущности датасета - определение возможной болезни сердца
# в случае, если мы сказали, что человек имеет болезнь и при этом ошиблись(FP) - это не страшно,
# тк после настоящего обследования все встанет на свои места
#
# в случае, если мы сказали, что человек имеет болезнь и не ошиблись (TP) - мы должны иметь хо
# чтобы люди шли на полное обследование
#
# в случае, если мы сказали, что человек не болен и при этом ошиблись (FN) - это очень плохо, т
# что здоров и будет жить в незнании
#
# в случае, если мы сказали, что человек не болен и при этом не ошиблись (TN) - опять же, лучш
#
#
# ИТОГО: главный приоритет - избежать FN. Далее важен случай TP.
```

```
# на основе этих размышлений выберем подходящие метрики
```

```
# на снове размышлений не смог придти к заключении о метриках

# выбрал recall, f1, accuracy
```

```
# логистическая регрессия
```

```
logistic = LogisticRegression()
logistic.fit(X_train, y_train)
logistic_y_test = logistic.predict(X_test)
```

```
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/linear_model
  FutureWarning)
```

```
# найдем показатели метрик для Логистической регрессии
```

```
classification_report(y_test, logistic_y_test, output_dict=True)["0"], \
classification_report(y_test, logistic_y_test, output_dict=True)["1"]
```

```
({'f1-score': 0.7384615384615385,
  'precision': 0.8,
  'recall': 0.6857142857142857,
  'support': 35},
 {'f1-score': 0.8045977011494252,
  'precision': 0.7608695652173914,
  'recall': 0.8536585365853658,
  'support': 41})
```

Метод опорных векторов

```
svc = SVC(kernel="rbf", C=0.5)
```

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

```
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
SVC(C=0.5, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

```
svc_y_test = svc.predict(X_test)
svc_y_test  # почему-то всегда голосует одним классом
```

```
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])
```

```
classification_report(y_test, svc_y_test, output_dict=True)["0"], \
classification_report(y_test, svc_y_test, output_dict=True)["1"]
```

```
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/metrics/clas
  'precision', 'predicted', average, warn_for)
({'f1-score': 0.0, 'precision': 0.0, 'recall': 0.0, 'support': 35},
 {'f1-score': 0.7008547008547009,
  'precision': 0.5394736842105263,
  'recall': 1.0,
  'support': 41})
```

```
# при помощи решетчатого поиска и кросс-валидации найдем оптимальное значение гиперпараметра C
scoring = {
    'recall': make_scorer(recall_score),
    'f1': make_scorer(f1_score),
    'accuracy': make_scorer(accuracy_score)
}
svc_n_range = [i/10 for i in np.array(range(1, 10, 1))]
svc_tuned_parameters = [{'C': svc_n_range}]
svc_tuned_parameters
```

```
[{'C': [0.1, 0.2, 0.3, 0.4, 0.5, 0.6, 0.7, 0.8, 0.9]}]
```

```
svc_gs = GridSearchCV(SVC(kernel="rbf"), svc_tuned_parameters, cv=5, scoring=scorin
svc_gs.fit(X_train, y_train)
```

```
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
```

```
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
```

```
# лучшая модель
best_svc = svc_gs.best_estimator_
best_svc
```

```
SVC(C=0.1, cache_size=200, class_weight=None, coef0=0.0,
  decision_function_shape='ovr', degree=3, gamma='auto_deprecated',
  kernel='rbf', max_iter=-1, probability=False, random_state=None,
  shrinking=True, tol=0.001, verbose=False)
```

```
# лучшее значение f1
svc_gs.best_score_
```

```
0.7065408778141771
```

```
# лучшее значение k
svc_gs.best_params_
```

```
{'C': 0.1}
```

```
# на начальном разбиении проверим метрики при новом значении c
best_svc.fit(X_train, y_train)
predicted_best_svc = best_svc.predict(X_test)
predicted_best_svc
```

```
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/svm/base.py:
  "avoid this warning.", FutureWarning)
```

```
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])
```

```
classification_report(y_test, predicted_best_svc, output_dict=True)["0"], \
classification_report(y_test, predicted_best_svc, output_dict=True)["1"]
```

```
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/metrics/clas
  'precision', 'predicted', average, warn_for)
({'f1-score': 0.0, 'precision': 0.0, 'recall': 0.0, 'support': 35},
 {'f1-score': 0.7008547008547009,
  'precision': 0.5394736842105263,
  'recall': 1.0,
  'support': 41})
```

Деревья решений

```
tree = DecisionTreeClassifier(random_state=1, max_depth=5)
tree.fit(X_train, y_train)
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=5,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
            splitter='best')
```

```
tree_y_test = tree.predict(X_test)
tree_y_test
```

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

```
classification_report(y_test, tree_y_test, output_dict=True)["0"], \
classification_report(y_test, tree_y_test, output_dict=True)["1"]
```

```
({'f1-score': 0.6857142857142857,
  'precision': 0.6857142857142857,
  'recall': 0.6857142857142857,
  'support': 35},
 {'f1-score': 0.7317073170731707,
  'precision': 0.7317073170731707,
  'recall': 0.7317073170731707,
  'support': 41})
```

```
# при помощи решетчатого поиска и кросс-валидации найдем оптимальное значение гиперпараметра C
tree_n_range = np.array(range(1, 20))
tree_tuned_parameters = [{'max_depth': tree_n_range}]
tree_tuned_parameters
```

```
[{'max_depth': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14,
        18, 19])}]
```

```
tree_gs = GridSearchCV(DecisionTreeClassifier(random_state=1), tree_tuned_parameter
tree_gs.fit(X_train, y_train)
```

```
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/model_select
  DeprecationWarning)
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=DecisionTreeClassifier(class_weight=None, criterion='gini', m
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
            splitter='best'),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid=[{'max_depth': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10
       18, 19])}],
       pre_dispatch='2*n_jobs', refit='f1', return_train_score='warn',
       scoring={'recall': make_scorer(recall_score), 'f1': make_scorer(f1_scor
       verbose=0)
```

```
# лучшая модель
best_tree = tree_gs.best_estimator_
best_tree
```

```
DecisionTreeClassifier(class_weight=None, criterion='gini', max_depth=4,
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
            splitter='best')
```

```
# лучшее значение f1
tree_gs.best_score_
```

```
0.8197614230237609
```

```
# на начальном разбиении проверим метрики при новом значении c
best_tree.fit(X_train, y_train)
predicted_best_tree = best_tree.predict(X_test)
predicted_best_tree
```

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

```
classification_report(y_test, predicted_best_tree, output_dict=True)["0"], \
classification_report(y_test, predicted_best_tree, output_dict=True)["1"]
```

```
({'f1-score': 0.7042253521126761,
  'precision': 0.6944444444444444,
  'recall': 0.7142857142857143,
```

```
  'support': 35},
 {'f1-score': 0.7407407407407408,
  'precision': 0.75,
  'recall': 0.7317073170731707,
  'support': 41})
```

```
# таким образом из трех моделей лучший результат показал метод Логистической регрессии
classification_report(y_test, logistic_y_test, output_dict=True)["0"], \
classification_report(y_test, logistic_y_test, output_dict=True)["1"]
```

```
({'f1-score': 0.7384615384615385,
  'precision': 0.8,
  'recall': 0.6857142857142857,
  'support': 35},
 {'f1-score': 0.8045977011494252,
  'precision': 0.7608695652173914,
  'recall': 0.8536585365853658,
  'support': 41})
```

Бэггинг

```
from sklearn.ensemble import BaggingClassifier
from sklearn.ensemble import ExtraTreesClassifier
```

```
bagging_tree = BaggingClassifier(DecisionTreeClassifier(random_state=1), n_estimato
```

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

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, cri
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
            splitter='best'),
         bootstrap=True, bootstrap_features=False, max_features=1.0,
         max_samples=1.0, n_estimators=100, n_jobs=None, oob_score=False,
         random_state=None, verbose=0, warm_start=False)
```

```
bagging_tree_y_test = bagging_tree.predict(X_test)
bagging_tree_y_test
```

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

```
classification_report(y_test, bagging_tree_y_test, output_dict=True)["0"], \
classification_report(y_test, bagging_tree_y_test, output_dict=True)["1"]
```

```
({'f1-score': 0.6865671641791045,
  'precision': 0.71875,
```

```
  'recall': 0.6571428571428571,
  'support': 35},
 {'f1-score': 0.7529411764705882,
  'precision': 0.7272727272727273,
  'recall': 0.7804878048780488,
  'support': 41})
```

```
bagging_tree_n_range = np.array(range(10, 200, 10))
bagging_tree_tuned_parameters = [{'n_estimators': bagging_tree_n_range}]
bagging_tree_tuned_parameters
```

```
[{'n_estimators': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110
        140, 150, 160, 170, 180, 190])}]
```

```
bagging_tree_gs = GridSearchCV(BaggingClassifier(DecisionTreeClassifier(random_stat
bagging_tree_gs.fit(X_train, y_train)
```

```
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/model_select
  DeprecationWarning)
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=BaggingClassifier(base_estimator=DecisionTreeClassifier(class
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            ...stimators=10, n_jobs=None, oob_score=False,
         random_state=None, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid=[{'n_estimators': array([ 10,  20,  30,  40,  50,  60,  70,
       140, 150, 160, 170, 180, 190])}],
       pre_dispatch='2*n_jobs', refit='f1', return_train_score='warn',
       scoring={'recall': make_scorer(recall_score), 'f1': make_scorer(f1_scor
       verbose=0)
```

```
best_bagging = bagging_tree_gs.best_estimator_
best_bagging
```

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, cri
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
            splitter='best'),
         bootstrap=True, bootstrap_features=False, max_features=1.0,
         max_samples=1.0, n_estimators=150, n_jobs=None, oob_score=False,
         random_state=None, verbose=0, warm_start=False)
```

```
bagging_tree_gs.best_score_
```

```
0.8712949651174036
```

```
bagging_tree_gs.best_params_
```

```
{'n_estimators': 150}
```

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

```
BaggingClassifier(base_estimator=DecisionTreeClassifier(class_weight=None, cri
            max_features=None, max_leaf_nodes=None,
            min_impurity_decrease=0.0, min_impurity_split=None,
            min_samples_leaf=1, min_samples_split=2,
            min_weight_fraction_leaf=0.0, presort=False, random_state=1,
            splitter='best'),
         bootstrap=True, bootstrap_features=False, max_features=1.0,
         max_samples=1.0, n_estimators=150, n_jobs=None, oob_score=False,
         random_state=None, verbose=0, warm_start=False)
```

```
best_bagging_y_test = best_bagging.predict(X_test)
best_bagging_y_test
```

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

```
classification_report(y_test, best_bagging_y_test, output_dict=True)["0"], \
classification_report(y_test, best_bagging_y_test, output_dict=True)["1"]
```

```
({'f1-score': 0.6956521739130436,
  'precision': 0.7058823529411765,
  'recall': 0.6857142857142857,
  'support': 35},
 {'f1-score': 0.746987951807229,
  'precision': 0.7380952380952381,
  'recall': 0.7560975609756098,
  'support': 41})
```

Сверхслучайные деревья

```
extra_trees = ExtraTreesClassifier(random_state=1, n_estimators=100)
```

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

```
ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini',
           max_depth=None, max_features='auto', max_leaf_nodes=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=1, verbose=0, warm_start=False)
```

```
extra_trees_y_test = extra_trees.predict(X_test)
extra_trees_y_test
```

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

```
classification_report(y_test, extra_trees_y_test, output_dict=True)["0"], \
classification_report(y_test, extra_trees_y_test, output_dict=True)["1"]
```

```
({'f1-score': 0.6956521739130436,
  'precision': 0.7058823529411765,
  'recall': 0.6857142857142857,
  'support': 35},
 {'f1-score': 0.746987951807229,
  'precision': 0.7380952380952381,
  'recall': 0.7560975609756098,
  'support': 41})
```

```
extra_tree_n_range = np.array(range(10, 200, 10))
extra_tree_tuned_parameters = [{'n_estimators': extra_tree_n_range}]
extra_tree_tuned_parameters
```

```
[{'n_estimators': array([ 10,  20,  30,  40,  50,  60,  70,  80,  90, 100, 110
         140, 150, 160, 170, 180, 190])}]
```

```
extra_tree_gs = GridSearchCV(ExtraTreesClassifier(random_state=1), extra_tree_tuned
extra_tree_gs.fit(X_train, y_train)
```

```
/Users/Kirill/py_learning/env/lib/python3.6/site-packages/sklearn/model_select
  DeprecationWarning)
GridSearchCV(cv=5, error_score='raise-deprecating',
       estimator=ExtraTreesClassifier(bootstrap=False, class_weight=None, crit
           max_depth=None, max_features='auto', max_leaf_nodes=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='warn', n_jobs=None,
           oob_score=False, random_state=1, verbose=0, warm_start=False),
       fit_params=None, iid='warn', n_jobs=None,
       param_grid=[{'n_estimators': array([ 10,  20,  30,  40,  50,  60,  70,
       140, 150, 160, 170, 180, 190])}],
       pre_dispatch='2*n_jobs', refit='f1', return_train_score='warn',
       scoring={'recall': make_scorer(recall_score), 'f1': make_scorer(f1_scor
       verbose=0)
```

```
best_extra_tree = extra_tree_gs.best_estimator_
best_extra_tree
```

```
ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini',
           max_depth=None, max_features='auto', max_leaf_nodes=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=140, n_jobs=None,
           oob_score=False, random_state=1, verbose=0, warm_start=False)
```

```
extra_tree_gs.best_params_
```

```
{'n_estimators': 140}
```

```
extra_tree_gs.best_score_
```

```
0.8871624773257318
```

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

```
ExtraTreesClassifier(bootstrap=False, class_weight=None, criterion='gini',
                     max_depth=None, max_features='auto', max_leaf_nodes=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=140, n_jobs=None,
                     oob_score=False, random_state=1, verbose=0, warm_start=False)
```

```
best_extra_tree_y_test = best_extra_tree.predict(X_test)
best_extra_tree_y_test
```

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

```
classification_report(y_test, best_extra_tree_y_test, output_dict=True)["0"], \
classification_report(y_test, best_extra_tree_y_test, output_dict=True)["1"]
```

```
({'f1-score': 0.6857142857142857,
  'precision': 0.6857142857142857,
  'recall': 0.6857142857142857,
  'support': 35},
 {'f1-score': 0.7317073170731707,
  'precision': 0.7317073170731707,
  'recall': 0.7317073170731707,
  'support': 41})
```

```
# для сравнения лучший результат беггинга
classification_report(y_test, best_bagging_y_test, output_dict=True)["0"], \
classification_report(y_test, best_bagging_y_test, output_dict=True)["1"]
```

```
({'f1-score': 0.6956521739130436,
  'precision': 0.7058823529411765,
  'recall': 0.6857142857142857,
  'support': 35},
 {'f1-score': 0.746987951807229,
  'precision': 0.7380952380952381,
  'recall': 0.7560975609756098,
  'support': 41})
```

Вывод

```
# таким образом, с небольшим перевесом себя лучше показал беггинг
# однако, лучший результат среди всех рассмотренных методов показал метод логистической регрессии с
```

```
classification_report(y_test, logistic_y_test, output_dict=True)["0"], \
classification_report(y_test, logistic_y_test, output_dict=True)["1"]
```

```
({'f1-score': 0.7384615384615385,
  'precision': 0.8,
  'recall': 0.6857142857142857,
  'support': 35},
 {'f1-score': 0.8045977011494252,
  'precision': 0.7608695652173914,
  'recall': 0.8536585365853658,
  'support': 41})
```