1) Набор данных для решения задачи классификации или регрессии

В качестве набора данных используется набор данных по раку груди висконсин (диагностический) Файл содержит следующие колонки:

- радиус (среднее расстояние от центра до точек по периметру)
- текстура (стандартное отклонение значений шкалы серого)
- периметр
- область
- гладкость (локальное изменение длины радиуса)
- компактность (периметр ^ 2 / площадь - 1.0)
- вогнутость (выраженность вогнутых участков контура)
- вогнутые точки (количество вогнутых участков контура)
- симметрия
- фрактальная размерность («приближение береговой линии» - 1)

Классы:

- **WDBC-**злокачественный
- **WDBC-**доброкачественный

In [2]:

```
import os
import numpy as np
import pandas as pd
import seaborn as sns
import matplotlib.pyplot as plt
from typing import Dict, Tuple
from scipy import stats
from sklearn.preprocessing import MinMaxScaler
from sklearn.linear_model import LinearRegression, LogisticRegression
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
from sklearn.model_selection import KFold, RepeatedKFold, LeaveOneOut, LeavePOut, ShuffleSplit, StratifiedKFold
from sklearn.model_selection import cross_val_score, cross_validate
from sklearn.metrics import accuracy_score, balanced_accuracy_score
from sklearn.metrics import precision_score, recall_score, f1_score, classification_report
from sklearn.metrics import confusion_matrix
from sklearn.metrics import plot_confusion_matrix
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import RandomizedSearchCV
from sklearn.metrics import mean_absolute_error, mean_squared_error, mean_squared_log_error, median_absolute_error, r2_score
from sklearn.metrics import roc_curve, roc_auc_score
from sklearn.svm import SVC, NuSVC, LinearSVC, OneClassSVM, SVR, NuSVR, LinearSVR
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
from sklearn.ensemble import RandomForestClassifier, RandomForestRegressor
from sklearn.ensemble import ExtraTreesClassifier, ExtraTreesRegressor
from sklearn.ensemble import GradientBoostingClassifier, GradientBoostingRegressor
%matplotlib inline
sns.set(style="ticks")
from sklearn.datasets import *
```

In [3]:

```
breast = load_breast_cancer()
```

In [4]:

```
breast['feature_names']
```

Out[4]:

```
array(['mean radius', 'mean texture', 'mean perimeter', 'mean area',
       'mean smoothness', 'mean compactness', 'mean concavity',
       'mean concave points', 'mean symmetry', 'mean fractal dimension',
       'radius error', 'texture error', 'perimeter error', 'area error',
       'smoothness error', 'compactness error', 'concavity error',
       'concave points error', 'symmetry error',
       'fractal dimension error', 'worst radius', 'worst texture',
       'worst perimeter', 'worst area', 'worst smoothness',
       'worst compactness', 'worst concavity', 'worst concave points',
       'worst symmetry', 'worst fractal dimension'], dtype='<U23')
```

In [5]:

```
breast['target_names']
```

Out[5]:

```
array(['malignant', 'benign'], dtype='<U9')
```

In [6]:

```
breast['data'].shape
```

Out[6]:

```
(569, 30)
```

In [7]:

```
breast['target'].shape
```

Out[7]:

```
(569,)
```

In [8]:

```
data = pd.DataFrame(data= np.c_[breast['data'], breast['target']],
                    columns= list(breast['feature_names']) + ['target'])
```

In [9]:

```
data
```

Out[9]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst texture	pe
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.30010	0.14710	0.2419	0.07871	...	17.33	
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.08690	0.07017	0.1812	0.05667	...	23.41	
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.19740	0.12790	0.2069	0.05999	...	25.53	
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.24140	0.10520	0.2597	0.09744	...	26.50	
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.19800	0.10430	0.1809	0.05883	...	16.67	
...	...	...	...	...	...	...	...	...	...	...	...	...	
564	21.56	22.39	142.00	1479.0	0.11100	0.11590	0.24390	0.13890	0.1726	0.05623	...	26.40	
565	20.13	28.25	131.20	1261.0	0.09780	0.10340	0.14400	0.09791	0.1752	0.05533	...	38.25	
566	16.60	28.08	108.30	858.1	0.08455	0.10230	0.09251	0.05302	0.1590	0.05648	...	34.12	
567	20.60	29.33	140.10	1265.0	0.11780	0.27700	0.35140	0.15200	0.2397	0.07016	...	39.42	
568	7.76	24.54	47.92	181.0	0.05263	0.04362	0.00000	0.00000	0.1587	0.05884	...	30.37	

569 rows × 31 columns

In [10]:

```
# Значения целевого признака
np.unique(breast.target)
```

Out[10]:

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

In [11]:

```
# Наименования значений целевого признака
breast.target_names
```

Out[11]:

```
array(['malignant', 'benign'], dtype='<U9')
```

In [12]:

```
list(zip(np.unique(breast.target), breast.target_names))
```

Out[12]:

```
[(0, 'malignant'), (1, 'benign')]
```

In [13]:

```
# Значения целевого признака
breast.target
```

Out[13]:

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

In [14]:

```
# Размер выборки
breast.data.shape, breast.target.shape
```

Out[14]:

```
((569, 30), (569,))
```

In [15]:

```
# И выведем его статистические характеристики
```

```
# И выведем его статистические характеристики
data.describe()
```

Out[15]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	dim
count	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.000000	569.
mean	14.127292	19.289649	91.969033	654.889104	0.096360	0.104341	0.088799	0.048919	0.181162	0.
std	3.524049	4.301036	24.298981	351.914129	0.014064	0.052813	0.079720	0.038803	0.027414	0.
min	6.981000	9.710000	43.790000	143.500000	0.052630	0.019380	0.000000	0.000000	0.106000	0.
25%	11.700000	16.170000	75.170000	420.300000	0.086370	0.064920	0.029560	0.020310	0.161900	0.
50%	13.370000	18.840000	86.240000	551.100000	0.095870	0.092630	0.061540	0.033500	0.179200	0.
75%	15.780000	21.800000	104.100000	782.700000	0.105300	0.130400	0.130700	0.074000	0.195700	0.
max	28.110000	39.280000	188.500000	2501.000000	0.163400	0.345400	0.426800	0.201200	0.304000	0.

8 rows × 31 columns

Разделение выборки на обучающую и тестовую

In [16]:

```
breast_X_train, breast_X_test, breast_y_train, breast_y_test = train_test_split(
    breast.data, breast.target, test_size=0.5, random_state=1)
```

In [17]:

```
# Размер обучающей выборки
breast_X_train.shape, breast_y_train.shape
```

Out[17]:

```
((284, 30), (284,))
```

In [18]:

```
# Размер тестовой выборки
breast_X_test.shape, breast_y_test.shape
```

Out[18]:

```
((285, 30), (285,))
```

In [19]:

```
np.unique(breast_y_train)
```

Out[19]:

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

In [20]:

```
np.unique(breast_y_test)
```

Out[20]:

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

In [21]:

```
def class_proportions(array: np.ndarray) -> Dict[int, Tuple[int, float]]:
    """
    Вычисляет пропорции классов
```

```python
    array - массив, содержащий метки классов
    """
    # Получение меток классов и количества меток каждого класса
    labels, counts = np.unique(array, return_counts=True)
    # Превращаем количество меток в процент их встречаемости
    # делим количество меток каждого класса на общее количество меток
    counts_perc = counts/array.size
    # Теперь sum(counts_perc)==1.0
    # Создаем результирующий словарь,
    # ключом словаря явлется метка класса,
    # а значением словаря процент встречаемости метки
    res = dict()
    for label, count2 in zip(labels, zip(counts, counts_perc)):
        res[label] = count2
    return res

def print_class_proportions(array: np.ndarray):
    """
    Вывод пропорций классов
    """
    proportions = class_proportions(array)
    if len(proportions)>0:
        print('Метка \t Количество \t Процент встречаемости')
    for i in proportions:
        val, val_perc = proportions[i]
        val_perc_100 = round(val_perc * 100, 2)
        print('{} \t {} \t \t {}%'.format(i, val, val_perc_100))
```

In [22]:

```python
print_class_proportions(breast.target)
```

```
Метка    Количество    Процент встречаемости
0    212      37.26%
1    357      62.74%
```

In [23]:

```python
# Для обучающей выборки
print_class_proportions(breast_y_train)
```

```
Метка    Количество    Процент встречаемости
0    109      38.38%
1    175      61.62%
```

In [24]:

```python
# Для тестовой выборки
print_class_proportions(breast_y_test)
```

```
Метка    Количество    Процент встречаемости
0    103      36.14%
1    182      63.86%
```

In [25]:

```python
# 2 ближайших соседа
cl1_1 = KNeighborsClassifier(n_neighbors=2)
cl1_1.fit(breast_X_train, breast_y_train)
target1_1 = cl1_1.predict(breast_X_test)
len(target1_1), target1_1
```

Out[25]:

```
(285,
 array([0, 0, 1, 0, 0, 0, 0, 0, 1, 1, 1, 0, 0, 1, 0, 0, 1, 1, 1, 0, 1, 1,
        0, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1,
        1, 0, 1, 1, 0, 0, 0, 0, 1, 1, 1, 1, 1, 0, 1, 1, 1, 0, 1, 0, 1, 1,
        1, 0, 1, 1, 1, 1, 0, 0, 1, 0, 1, 1, 1, 0, 1, 0, 1, 0, 1, 1, 0, 1,
        0, 1, 1, 0, 1, 1, 0, 0, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 1, 0, 0,
        1, 1, 0, 1, 0, 0, 1, 1, 1, 1, 1, 0, 0, 1, 1, 0, 0, 0, 0, 0, 1, 0,
        1, 0, 1, 0, 0, 0, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 0, 1, 1, 1, 1,
```

```
0, 1, 1, 1, 1, 1, 1, 1, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 1, 0, 1, 1,
1, 1, 0, 1, 1, 1, 1, 1, 0, 0, 0, 1, 0, 1, 1, 1, 0, 0, 0, 1, 1, 0,
0, 1, 1, 1, 1, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0, 0, 1, 1, 1, 1,
1, 1, 0, 1, 0, 1, 1, 0, 0, 0, 1, 1, 1, 0, 0, 0, 0, 0, 1, 0, 1, 1,
1, 1, 1, 0, 1, 1, 0, 0, 0, 0, 1, 0, 0, 1, 0, 1, 0, 1, 1, 1, 1, 0,
1, 1, 0, 1, 1, 1, 0, 0, 1, 1, 0, 1, 1, 1, 1, 0, 1, 1, 1, 1, 1]))
```

In [26]:

```
# 2 ближайших соседа
accuracy_score(breast_y_test, target1_1)
```

Out[26]:

```
0.8842105263157894
```

Точность в случае **2** ближайших соседей составляет **91%.**

In [27]:

```
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики accuracy для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Accuracy для данного класса
    """
    # Для удобства фильтрации сформируем Pandas DataFrame
    d = {'t': y_true, 'p': y_pred}
    df = pd.DataFrame(data=d)
    # Метки классов
    classes = np.unique(y_true)
    # Результирующий словарь
    res = dict()
    # Перебор меток классов
    for c in classes:
        # отфильтруем данные, которые соответствуют
        # текущей метке класса в истинных значениях
        temp_data_flt = df[df['t']==c]
        # расчет accuracy для заданной метки класса
        temp_acc = accuracy_score(
            temp_data_flt['t'].values,
            temp_data_flt['p'].values)
        # сохранение результата в словарь
        res[c] = temp_acc
    return res

def print_accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray):
    """
    Вывод метрики accuracy для каждого класса
    """
    accs = accuracy_score_for_classes(y_true, y_pred)
    if len(accs)>0:
        print('Метка \t Accuracy')
    for i in accs:
        print('{} \t {}'.format(i, accs[i]))
```

In [28]:

```
# 2 ближайших соседа
print_accuracy_score_for_classes(breast_y_test, target1_1)
```

```
Метка 	 Accuracy
0 	 0.883495145631068
1 	 0.8846153846153846
```

In [29]:

```
balanced_accuracy_score(breast_y_test, target1_1)
```

Out[29]:

```
0.8840552651232263
```

Матрица ошибок или Confusion Matrix

In [30]:

```
confusion_matrix(breast_y_test, target1_1, labels=[0, 1])
```

Out[30]:

```
array([[ 91,  12],
       [ 21, 161]], dtype=int64)
```

In [31]:

```
tn, fp, fn, tp = confusion_matrix(breast_y_test, target1_1).ravel()
tn, fp, fn, tp
```

Out[31]:

```
(91, 12, 21, 161)
```

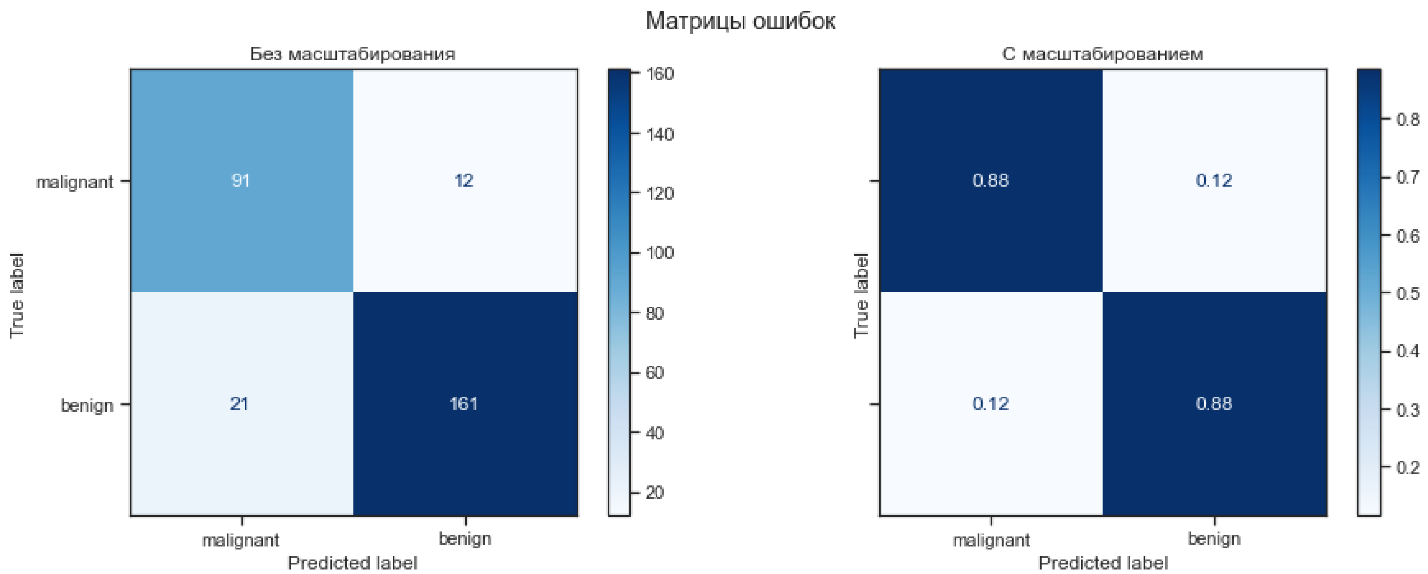
In [32]:

```
fig, ax = plt.subplots(1, 2, sharex='col', sharey='row', figsize=(15,5))

plot_confusion_matrix(cl1_1, breast_X_test, breast_y_test,
                      display_labels=breast.target_names, cmap=plt.cm.Blues, ax=ax[0])

plot_confusion_matrix(cl1_1, breast_X_test, breast_y_test,
                      display_labels=breast.target_names,
                      cmap=plt.cm.Blues, normalize='true', ax=ax[1])

fig.suptitle('Матрицы ошибок')
ax[0].title.set_text('Без масштабирования')
ax[1].title.set_text('С масштабированием')
```



In [33]:

```
# precision=TP/(TP+FP)
# recall=TP/(TP+FN)
# Для 2 ближайших соседей
precision_score(breast_y_test, target1_1), recall_score(breast_y_test, target1_1)
```

Out[33]:

```
(0.930635838150289, 0.8846153846153846)
```

In [34]:

```
# Параметры TP, TN, FP, FN считаются как сумма по всем классам
precision_score(breast_y_test, target1_1, average='micro')
```

Out[34]:

```
0.8842105263157894
```

In [35]:

```
# Параметры TP, TN, FP, FN считаются отдельно для каждого класса
# и берется среднее значение, дисбаланс классов не учитывается.
precision_score(breast_y_test, target1_1, average='macro')
```

Out[35]:

```
0.8715679190751445
```

In [36]:

```
# Параметры TP, TN, FP, FN считаются отдельно для каждого класса
# и берется средневзвешенное значение, дисбаланс классов учитывается
# в виде веса классов (вес - количество истинных значений каждого класса).
precision_score(breast_y_test, target1_1, average='weighted')
```

Out[36]:

```
0.8879411317310617
```

ROC-кривая

In [37]:

```
# Обучим модели на задаче бинарной классифкации,
# чтобы получить вероятности классов

# 2 ближайших соседа
bin_cl1_1 = KNeighborsClassifier(n_neighbors=2)
bin_cl1_1.fit(breast_X_train, breast_y_train)
# предскажем метки классов
bin_cl1_1.predict(breast_X_test)
```

Out[37]:

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

In [38]:

```
# предскажем вероятности классов
proba_target1_1 = bin_cl1_1.predict_proba(breast_X_test)
len(proba_target1_1), proba_target1_1
```

Out[38]:

```
(285,
 array([[1. , 0. ],
        [1. , 0. ],
        [0. , 1. ],
```

```
[1. , 0. ],
[0.5, 0.5],
[1. , 0. ],
[1. , 0. ],
[0.5, 0.5],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[1. , 0. ],
[0.5, 0.5],
[0. , 1. ],
[1. , 0. ],
[0.5, 0.5],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[1. , 0. ],
[0. , 1. ],
[0. , 1. ],
[1. , 0. ],
[0. , 1. ],
[1. , 0. ],
[0. , 1. ],
[0. , 1. ],
[1. , 0. ],
[1. , 0. ],
[1. , 0. ],
[1. , 0. ],
[0. , 1. ],
[1. , 0. ],
[0.5, 0.5],
[0. , 1. ],
[0. , 1. ],
[1. , 0. ],
[1. , 0. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[0.5, 0.5],
[0. , 1. ],
[0. , 1. ],
[0.5, 0.5],
[0.5, 0.5],
[1. , 0. ],
[1. , 0. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[1. , 0. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[1. , 0. ],
[0. , 1. ],
[1. , 0. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[1. , 0. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[0. , 1. ],
[0.5, 0.5],
[1. , 0. ],
[0. , 1. ],
```

```
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [0.5, 0.5],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [1. , 0. ],
       [0.5, 0.5],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
```

```
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0.5, 0.5],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
```

```
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [0.5, 0.5],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [1. , 0. ],
       [1. , 0. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0.5, 0.5],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [1. , 0. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ],
       [0. , 1. ]]))
```

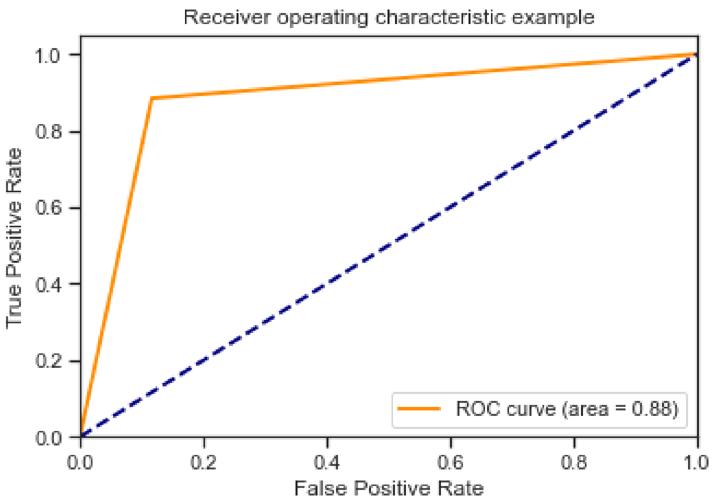
In [39]:

```
# Отрисовка ROC-кривой
def draw_roc_curve(y_true, y_score, pos_label, average):
```

```
    fpr, tpr, thresholds = roc_curve(y_true, y_score,
                                     pos_label=pos_label)
    roc_auc_value = roc_auc_score(y_true, y_score, average=average)
    plt.figure()
    lw = 2
    plt.plot(fpr, tpr, color='darkorange',
             lw=lw, label='ROC curve (area = %0.2f)' % roc_auc_value)
    plt.plot([0, 1], [0, 1], color='navy', lw=lw, linestyle='--')
    plt.xlim([0.0, 1.0])
    plt.ylim([0.0, 1.05])
    plt.xlabel('False Positive Rate')
    plt.ylabel('True Positive Rate')
    plt.title('Receiver operating characteristic example')
    plt.legend(loc="lower right")
    plt.show()
```

In [40]:

```
# Для 2 ближайших соседей
draw_roc_curve(breast_y_test, target1_1, pos_label=1, average='micro')
```



Кросс-валидация

Стратегия кросс-валидации определяется автоматически.

In [41]:

```
#кросс-валидации определяется автоматически.
scores = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                         breast.data, breast.target, cv=3)
```

In [42]:

```
# Значение метрики accuracy для 3 фолдов
scores
```

Out[42]:

```
array([0.89473684, 0.93157895, 0.88888889])
```

In [43]:

```
# Усредненное значение метрики accuracy для 3 фолдов
np.mean(scores)
```

Out[43]:

```
0.9050682261208577
```

In [44]:

```
# Укажем несколько метрик
```

```
scoring = {'precision': 'precision_weighted',
           'recall': 'recall_weighted',
           'f1': 'f1_weighted'}
```

In [45]:

```
scores = cross_validate(KNeighborsClassifier(n_neighbors=2),
                        breast.data, breast.target, scoring=scoring,
                        cv=3, return_train_score=True)
scores
```

Out[45]:

```
{'fit_time': array([0.00199914, 0.00100946, 0.00099993]),
 'score_time': array([0.00899625, 0.00799632, 0.00799727]),
 'test_precision': array([0.89432078, 0.93182303, 0.89533308]),
 'train_precision': array([0.9776781 , 0.97082105, 0.96429769]),
 'test_recall': array([0.89473684, 0.93157895, 0.88888889]),
 'train_recall': array([0.9762533 , 0.96833773, 0.96052632]),
 'test_f1': array([0.89442356, 0.93167383, 0.89006119]),
 'train_f1': array([0.97639169, 0.96857447, 0.96087426])}
```

K-fold стратегия

In [46]:

```
# Возвращаются индексы элементов
X = ["a", "b", "c"]
kf = KFold(n_splits=3)
for train, test in kf.split(X):
    print("%s %s" % (train, test))
```

```
[1 2] [0]
[0 2] [1]
[0 1] [2]
```

In [47]:

```
X = range(12)
kf = KFold(n_splits=3)
for train, test in kf.split(X):
    print("%s %s" % (train, test))
```

```
[ 4  5  6  7  8  9 10 11] [0 1 2 3]
[ 0  1  2  3  8  9 10 11] [4 5 6 7]
[0 1 2 3 4 5 6 7] [ 8  9 10 11]
```

In [48]:

```
%%time
kf = KFold(n_splits=5)
scores = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                         breast.data, breast.target, scoring='f1_weighted',
                         cv=kf)
scores
```

```
Wall time: 41 ms
```

Out[48]:

```
array([0.87833964, 0.9122807 , 0.94764978, 0.90663256, 0.89111744])
```

In [49]:

```
np.mean(scores)
```

Out[49]:

```
0.9072040226904188
```

In [50]:

```
%%time
kf = KFold(n_splits=5)
scores = cross_validate(KNeighborsClassifier(n_neighbors=2),
                        breast.data, breast.target, scoring=scoring,
                        cv=kf, return_train_score=True)
scores
```

```
Wall time: 127 ms
```

Out[50]:

```
{'fit_time': array([0.00200868, 0.00199819, 0.0020082 , 0.00199938, 0.0019989 ]),
 'score_time': array([0.00698733, 0.00699687, 0.00499845, 0.0059886 , 0.00498891]),
 'test_precision': array([0.89296517, 0.9122807 , 0.94855194, 0.9172852 , 0.91524629]),
 'train_precision': array([0.97753988, 0.96981109, 0.96967738, 0.96766249, 0.97335361]),
 'test_recall': array([0.87719298, 0.9122807 , 0.94736842, 0.90350877, 0.88495575]),
 'train_recall': array([0.97582418, 0.96703297, 0.96703297, 0.96483516, 0.97149123]),
 'test_f1': array([0.87833964, 0.9122807 , 0.94764978, 0.90663256, 0.89111744]),
 'train_f1': array([0.97605126, 0.96732351, 0.96727296, 0.9650388 , 0.97162092])}
```

Repeated K-fold стратегия

In [51]:

```
X = range(12)
kf = RepeatedKFold(n_splits=3, n_repeats=2)
for train, test in kf.split(X):
    print("%s %s" % (train, test))
```

```
[ 1  4  5  6  7  8  9 10] [ 0  2  3 11]
[ 0  2  3  4  5  9 10 11] [1 6 7 8]
[ 0  1  2  3  6  7  8 11] [ 4  5  9 10]
[ 0  1  2  4  7  8  9 10] [ 3  5  6 11]
[ 0  2  3  4  5  6 10 11] [1 7 8 9]
[ 1  3  5  6  7  8  9 11] [ 0  2  4 10]
```

In [52]:

```
%%time
kf = RepeatedKFold(n_splits=5)
scores = cross_val_score(KNeighborsClassifier(n_neighbors=2),
                         breast.data, breast.target, scoring='f1_weighted',
                         cv=kf)
scores
```

```
Wall time: 321 ms
```

Out[52]:

```
array([0.86189999, 0.87914292, 0.93853426, 0.92171288, 0.91207887,
       0.9476707 , 0.87902047, 0.89606972, 0.9472189 , 0.93791991,
       0.89473684, 0.93867864, 0.9216177 , 0.89579443, 0.91372835,
       0.90609094, 0.90414591, 0.92073577, 0.9135146 , 0.92885929,
       0.93111176, 0.92907427, 0.89413876, 0.91329774, 0.86960529,
       0.9386992 , 0.91358605, 0.86051282, 0.93068826, 0.91160183,
       0.88619536, 0.9122807 , 0.9312096 , 0.90415422, 0.90281227,
       0.93944738, 0.90370377, 0.90538074, 0.90316877, 0.90361291,
       0.91289649, 0.90207423, 0.9296252 , 0.89623206, 0.89525248,
       0.92170903, 0.91321856, 0.92982456, 0.9122807 , 0.90330255])
```

In [53]:

```
np.mean(scores)
```

Out[53]:

```
0.9111973733025737
```

In [77]:

```
np.mean(scores)
```

Out[77]:

```
0.9086115992970123
```

Подбор гиперпараметров GridSearchCV

In [54]:

```
breast_X_train.shape
```

Out[54]:

```
(284, 30)
```

In [63]:

```
n_range = np.array(range(1,100,1))
tuned_parameters = [{'n_neighbors': n_range}]
tuned_parameters
```

Out[63]:

```
[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17
,
         18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
         35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
         52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
         69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
         86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}]
```

In [64]:

```
%%time
clf_gs = GridSearchCV(KNeighborsClassifier(), tuned_parameters, cv=6, scoring='accuracy'
)
clf_gs.fit(breast_X_train, breast_y_train)
```

```
Wall time: 2.05 s
```

Out[64]:

```
GridSearchCV(cv=6, estimator=KNeighborsClassifier(),
             param_grid=[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7,  8,  9, 10, 1
1, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}],
             scoring='accuracy')
```

Лучшая модель

In [65]:

```
clf_gs.best_estimator_
```

Out[65]:

```
KNeighborsClassifier(n_neighbors=10)
```

In [66]:

```
clf_gs.best_params_
```

Out[66]:

```
{'n_neighbors': 10}
```

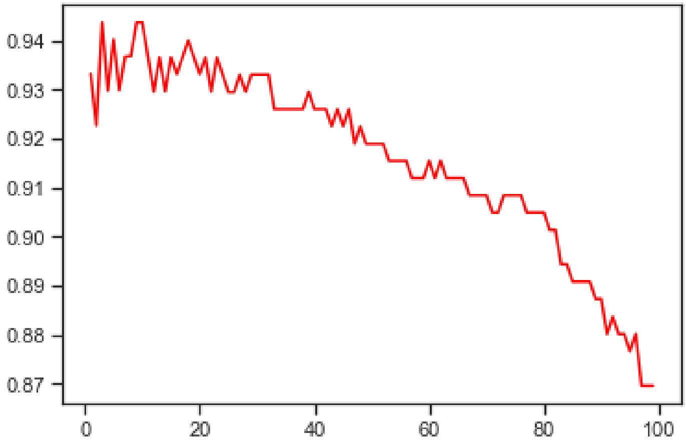
Изменение качества на тестовой выборке в зависимости от K-соседей

In [67]:

```
plt.plot(n_range, clf_gs.cv_results_['mean_test_score'], color="red")
```

Out[67]:

```
[<matplotlib.lines.Line2D at 0x278cf38e6d0>]
```



Подбор гиперпараметров **RandomizedSearchCV**

In [68]:

```
%%time
clf_rs = RandomizedSearchCV(KNeighborsClassifier(), tuned_parameters, cv=5, scoring='acc
uracy')
clf_rs.fit(breast_X_train, breast_y_train)
```

```
Wall time: 194 ms
```

Out[68]:

```
RandomizedSearchCV(cv=5, estimator=KNeighborsClassifier(),
                   param_distributions=[{'n_neighbors': array([ 1,  2,  3,  4,  5,  6,  7
,  8,  9, 10, 11, 12, 13, 14, 15, 16, 17,
       18, 19, 20, 21, 22, 23, 24, 25, 26, 27, 28, 29, 30, 31, 32, 33, 34,
       35, 36, 37, 38, 39, 40, 41, 42, 43, 44, 45, 46, 47, 48, 49, 50, 51,
       52, 53, 54, 55, 56, 57, 58, 59, 60, 61, 62, 63, 64, 65, 66, 67, 68,
       69, 70, 71, 72, 73, 74, 75, 76, 77, 78, 79, 80, 81, 82, 83, 84, 85,
       86, 87, 88, 89, 90, 91, 92, 93, 94, 95, 96, 97, 98, 99])}],
                   scoring='accuracy')
```

In [69]:

```
clf_rs.best_score_, clf_rs.best_params_
```

Out[69]:

```
(0.936466165413534, {'n_neighbors': 13})
```

In [70]:

```
clf_gs.best_score_, clf_gs.best_params_
```

Out[70]:

```
(0.9437795508274233, {'n_neighbors': 10})
```

Качество оптимальной модели.

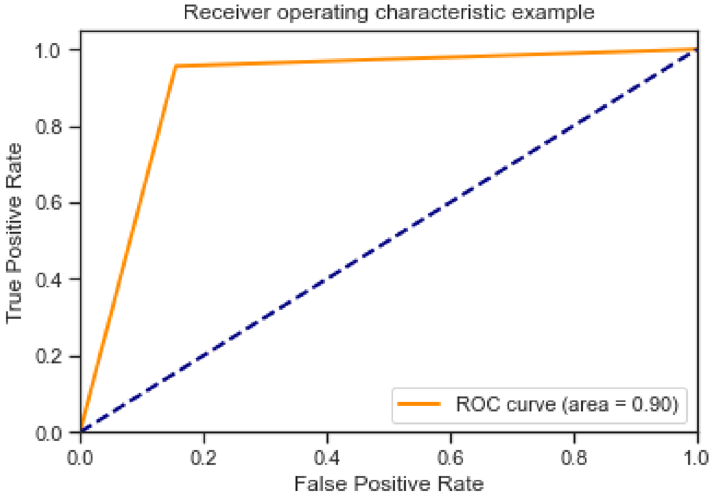
In [71]:

```
# 10 ближайших соседа
cl1_3 = KNeighborsClassifier(n_neighbors=10)
```

```
cl1_3.fit(breast_X_train, breast_y_train)
target1_3 = cl1_3.predict(breast_X_test)
```

In [72]:

```
# Для 10 ближайших соседей
draw_roc_curve(breast_y_test, target1_3, pos_label=1, average='micro')
```



In [73]:

```
# Для 10 ближайших соседей
print_accuracy_score_for_classes(breast_y_test, target1_3)
```

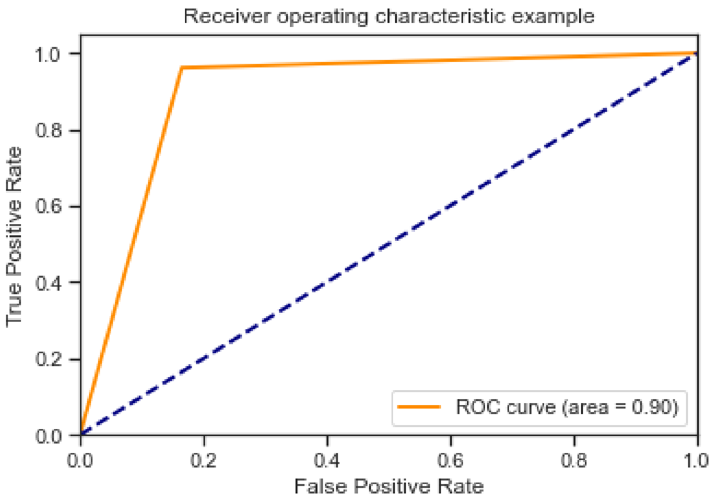
```
Метка   Accuracy
0    0.8446601941747572
1    0.9560439560439561
```

In [74]:

```
# 13 ближайших соседа
cl1_4 = KNeighborsClassifier(n_neighbors=13)
cl1_4.fit(breast_X_train, breast_y_train)
target1_4 = cl1_4.predict(breast_X_test)
```

In [75]:

```
# Для 13 ближайших соседей
draw_roc_curve(breast_y_test, target1_4, pos_label=1, average='micro')
```



In [76]:

```
# Для 13 ближайших соседей
print_accuracy_score_for_classes(breast_y_test, target1_4)
```

```
Метка   Accuracy
0    0.8349514563106796
1    0.9615384615384616
```

```
1    0.9615384615384616
```

Сравнение метрики качества исходной и оптимальной модели

In [77]:

```
fig, ax = plt.subplots(1, 3, sharex='col', sharey='row', figsize=(15,5))

plot_confusion_matrix(cl1_1, breast_X_test, breast_y_test,
                      display_labels=breast.target_names,
                      cmap=plt.cm.Blues, normalize='true', ax=ax[0])

plot_confusion_matrix(cl1_3, breast_X_test, breast_y_test,
                      display_labels=breast.target_names,
                      cmap=plt.cm.Blues, normalize='true', ax=ax[1])

plot_confusion_matrix(cl1_4, breast_X_test, breast_y_test,
                      display_labels=breast.target_names,
                      cmap=plt.cm.Blues, normalize='true', ax=ax[2])

fig.suptitle('Матрицы ошибок')
ax[0].title.set_text('K=2')
ax[1].title.set_text('K=10')
ax[2].title.set_text('K=13')
```