

**Цель лабораторной работы:** изучение линейных моделей, SVM и деревьев решений.

**Задание:**

1. Выберите набор данных (датасет) для решения задачи классификации или регрессии.
2. В случае необходимости проведите удаление или заполнение пропусков и кодирование категориальных признаков.
3. С использованием метода `train_test_split` разделите выборку на обучающую и тестовую.
4. Обучите следующие модели:
  - одну из линейных моделей;
  - SVM;
  - дерево решений.
5. Оцените качество моделей с помощью двух подходящих для задачи метрик. Сравните качество полученных моделей.

**Дополнительные задания:**

- Проведите эксперименты с важностью признаков в дереве решений.
- Визуализируйте дерево решений.

# Линейные модели, SVM и деревья решений.

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 [215]:

```
import numpy as np
import pandas as pd
from typing import Dict, Tuple
from scipy import stats
from sklearn.datasets import load_breast_cancer
from sklearn.model_selection import train_test_split
from sklearn.linear_model import LogisticRegression
from sklearn.neighbors import KNeighborsRegressor, KNeighborsClassifier
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 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.linear_model import LinearRegression
from sklearn.linear_model import SGDRegressor
from sklearn.linear_model import SGDClassifier
from sklearn.tree import DecisionTreeClassifier, DecisionTreeRegressor, export_graphviz
import seaborn as sns
import matplotlib.pyplot as plt
import graphviz
import pydotplus
%matplotlib inline
sns.set(style="ticks")
```

In [8]:

```
breast = load_breast_cancer()
```

In [9]:

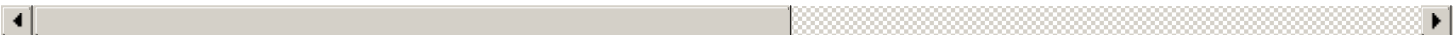
```
df_breast = pd.DataFrame(breast.data, columns=breast.feature_names)
df_breast['target'] = pd.Series(breast.target)
df_breast.head()
```

Out[9]:

mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave	mean symmetry	mean fractal	...	worst texture	v perim
----------------	-----------------	-------------------	--------------	--------------------	---------------------	-------------------	-----------------	------------------	-----------------	-----	------------------	------------

	radius	texture	perimeter	area	smoothness	compactness	concavity	points	mean	dimension	mean	texture	perim
0	mean 17.99	mean 10.38	mean 122.80	mean 1001.0	mean 0.11840	mean 0.27760	mean 0.3001	mean 0.14716	mean 0.1812	mean 0.05667	...	worst 17.33	perim
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	23.41	15
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	25.53	15
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	26.50	9
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	16.67	15

5 rows x 31 columns



In [12]:

```
# Значения и наименования значений целевого признака
list(zip(np.unique(breast.target), breast.target_names))
```

Out[12]:

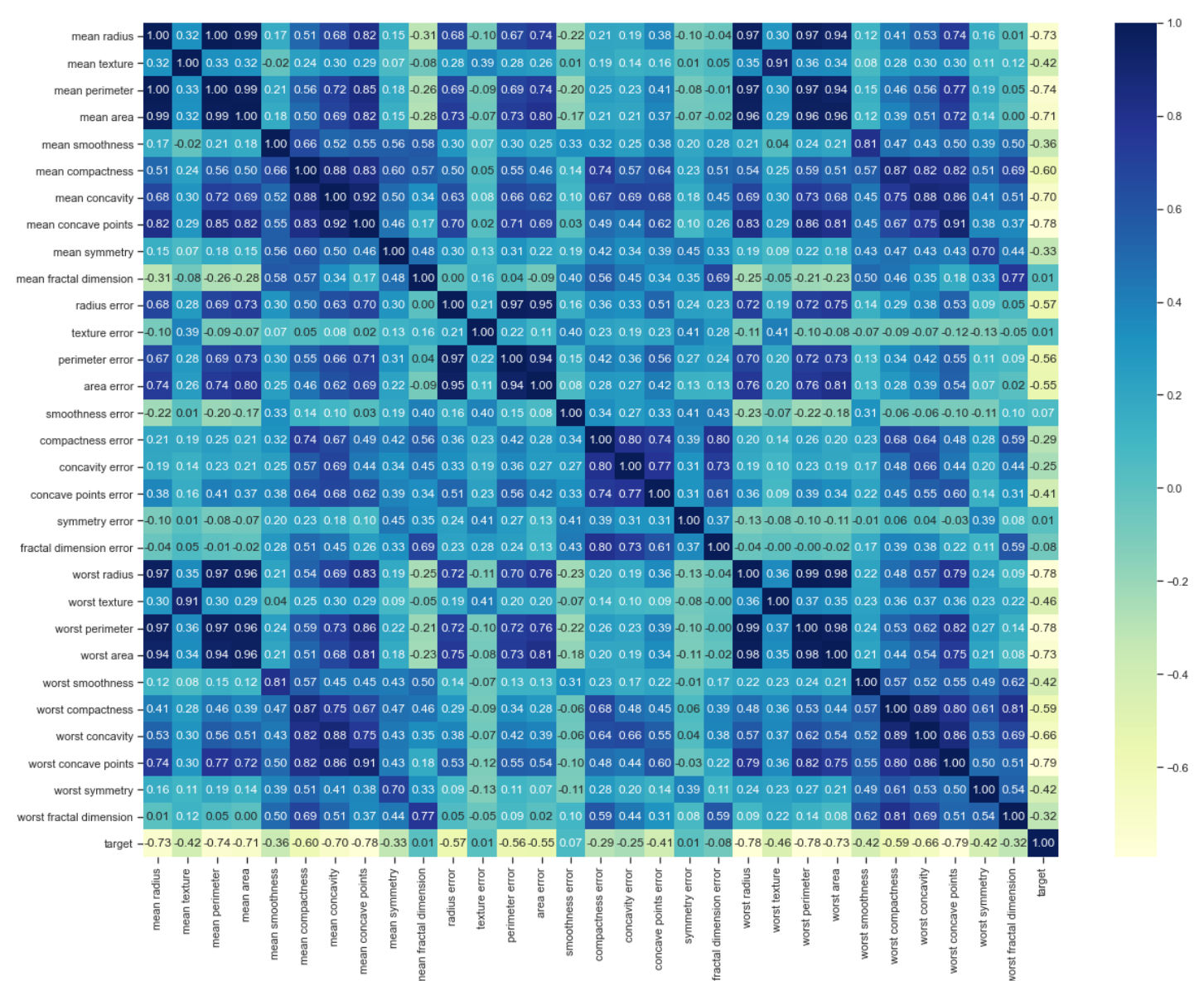
[(0, 'malignant'), (1, 'benign')]

In [206]:

```
#Построим корреляционную матрицу
fig, ax = plt.subplots(figsize=(20,15))
sns.heatmap(df_breast.corr(method='pearson'), ax=ax, annot=True, fmt='.2f', cmap="YlGnBu")
```

Out[206]:

<AxesSubplot:>



# Логистическая регрессия (LogisticRegression)

In [19]:

```
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 [34]:

```
cl1 = LogisticRegression(max_iter=10000)
```

In [35]:

```
cl1.fit(breast_X_train, breast_y_train)
```

Out[35]:

```
LogisticRegression(max_iter=10000)
```

In [24]:

```
pred_breast_y_test = cl1.predict(breast_X_test)
pred_breast_y_test
```

Out[24]:

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

In [25]:

```
pred_breast_y_test_proba = cl1.predict_proba(breast_X_test)
pred_breast_y_test_proba[:10]
```

Out[25]:

```
array([[1.49258350e-01, 8.50741650e-01],
       [8.53428519e-01, 1.46571481e-01],
       [6.17810193e-05, 9.99938219e-01],
       [9.99911960e-01, 8.80397272e-05],
       [5.52972743e-01, 4.47027257e-01],
       [9.99953966e-01, 4.60336276e-05],
       [9.99839687e-01, 1.60312607e-04],
       [5.85687939e-01, 4.14312061e-01],
       [7.51937022e-03, 9.92480630e-01],
       [5.46643320e-04, 9.99453357e-01]])
```

In [26]:

```
# Вероятность принадлежности к 0 классу
[round(x, 4) for x in pred_breast_y_test_proba[:10,0]]
```

Out[26]:

```
[0.1493, 0.8534, 0.0001, 0.9999, 0.553, 1.0, 0.9998, 0.5857, 0.0075, 0.0005]
```

In [27]:

```
# Вероятность принадлежности к 1 классу
[round(x, 4) for x in pred_breast_y_test_proba[:10,1]]
```

Out[27]:

```
[0.8507, 0.1466, 0.9999, 0.0001, 0.447, 0.0, 0.0002, 0.4143, 0.9925, 0.9995]
```

In [28]:

```
# Сумма вероятностей равна 1
pred_breast_y_test_proba[:10,0] + pred_breast_y_test_proba[:10,1]
```

Out[28]:

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

In [29]:

```
accuracy_score(breast_y_test, pred_breast_y_test)
```

Out[29]:

```
0.9052631578947369
```

In [30]:

```
def accuracy_score_for_classes(
    y_true: np.ndarray,
    y_pred: np.ndarray) -> Dict[int, float]:
    """
    Вычисление метрики ассигуры для каждого класса
    y_true - истинные значения классов
    y_pred - предсказанные значения классов
    Возвращает словарь: ключ - метка класса,
    значение - Ассигура 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 ассигуры 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 ассигуры 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 [31]:

```
print_accuracy_score_for_classes(breast_y_test, pred_breast_y_test)
```

```
Метка    Accuracy
0    0.8446601941747572
1    0.9395604395604396
```

# Метод опорных векторов (SVC)

In [102]:

```
breast_X_a = df_breast['worst radius'].values
breast_X_b = df_breast['worst texture'].values
breast_X = np.column_stack((breast_X_a, breast_X_b))
breast_y = breast.target
```

In [104]:

```
def make_meshgrid(x, y, h=.02):
    """Create a mesh of points to plot in

    Parameters
    -----
    x: data to base x-axis meshgrid on
    y: data to base y-axis meshgrid on
    h: stepsize for meshgrid, optional

    Returns
    -----
    xx, yy : ndarray
    """
    x_min, x_max = x.min() - 1, x.max() + 1
    y_min, y_max = y.min() - 1, y.max() + 1
    xx, yy = np.meshgrid(np.arange(x_min, x_max, h),
                          np.arange(y_min, y_max, h))
    return xx, yy

def plot_contours(ax, clf, xx, yy, **params):
    """Plot the decision boundaries for a classifier.

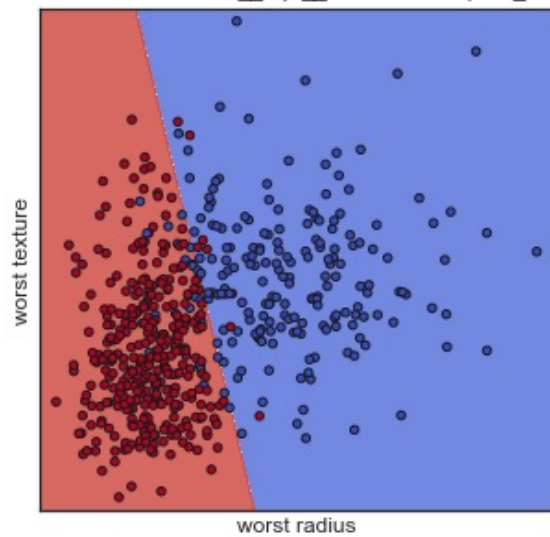
    Parameters
    -----
    ax: matplotlib axes object
    clf: a classifier
    xx: meshgrid ndarray
    yy: meshgrid ndarray
    params: dictionary of params to pass to contourf, optional
    """
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
    Z = Z.reshape(xx.shape)
    #Можно проверить все ли метки классов предсказываются
    #print(np.unique(Z))
    out = ax.contourf(xx, yy, Z, **params)
    return out

def plot_cl(clf):
    title = clf.__repr__
    clf.fit(breast_X, breast_y)
    fig, ax = plt.subplots(figsize=(5,5))
    X0, X1 = breast_X[:, 0], breast_X[:, 1]
    xx, yy = make_meshgrid(X0, X1)
    plot_contours(ax, clf, xx, yy, cmap=plt.cm.coolwarm, alpha=0.8)
    ax.scatter(X0, X1, c=breast_y, cmap=plt.cm.coolwarm, s=20, edgecolors='k')
    ax.set_xlim(xx.min(), xx.max())
    ax.set_ylim(yy.min(), yy.max())
    ax.set_xlabel('worst radius')
    ax.set_ylabel('worst texture')
    ax.set_xticks(())
    ax.set_yticks(())
    ax.set_title(title)
    plt.show()
```

In [105]:

```
plot_cl(LinearSVC(C=1.0, max_iter=100000))
```

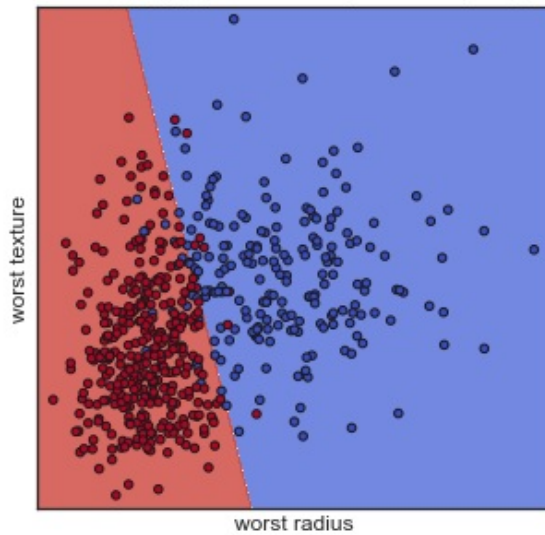
<bound method BaseEstimator.\_\_repr\_\_ of LinearSVC(max\_iter=100000)>



In [106]:

```
plot_cl(LinearSVC(C=1.0, penalty='l1', dual=False, max_iter=10000))
```

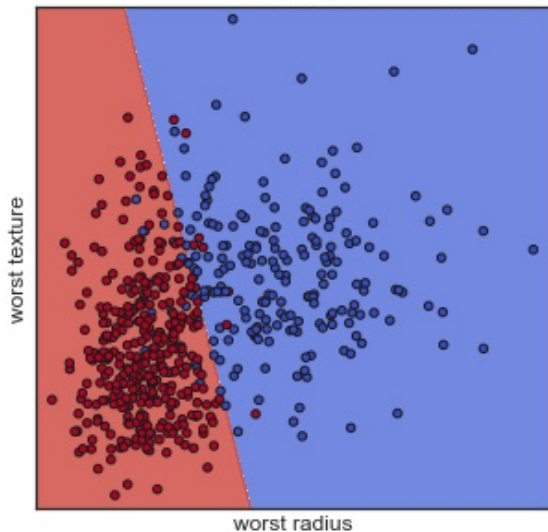
<bound method BaseEstimator.\_\_repr\_\_ of LinearSVC(dual=False, max\_iter=10000, penalty='l1')>



In [107]:

```
plot_cl(SVC(kernel='linear', C=1.0))
```

<bound method BaseEstimator.\_\_repr\_\_ of SVC(kernel='linear')>



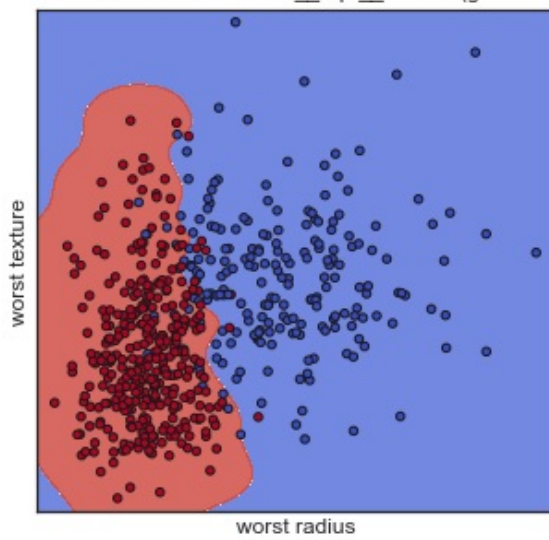
In [108]:

```
plot_cl(SVC(kernel='rbf', gamma=0.2, C=1.0))
```

<bound method BaseEstimator.\_\_repr\_\_ of SVC(gamma=0.2)>



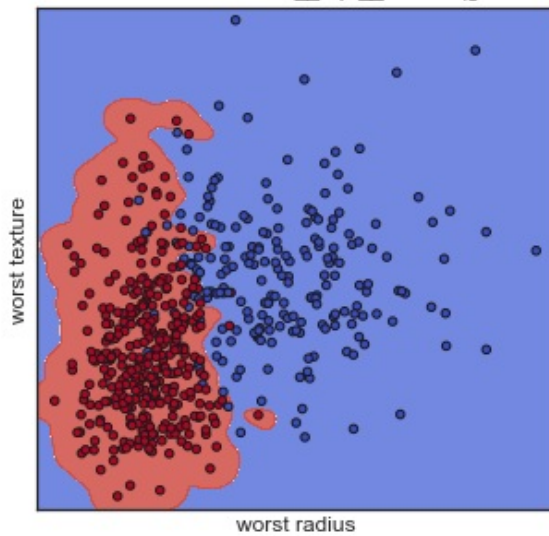
<bound method BaseEstimator.\_\_repr\_\_ of SVC(gamma=0.2)>



In [109]:

```
plot_cl(SVC(kernel='rbf', gamma=0.9, C=1.0))
```

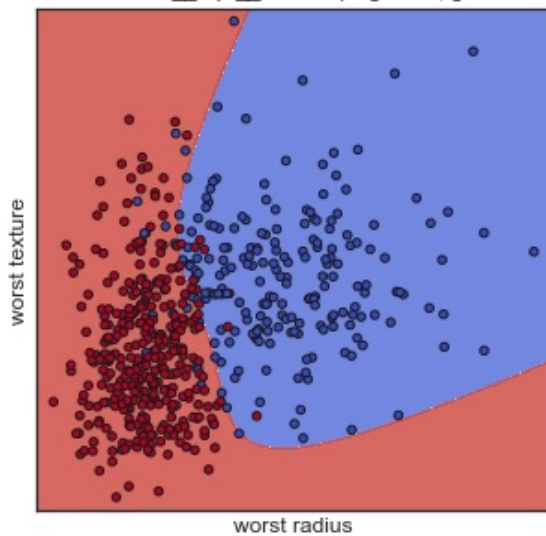
<bound method BaseEstimator.\_\_repr\_\_ of SVC(gamma=0.9)>



In [110]:

```
plot_cl(SVC(kernel='poly', degree=4, gamma=0.2, C=1.0))
```

<bound method BaseEstimator.\_\_repr\_\_ of SVC(degree=4, gamma=0.2, kernel='poly')>



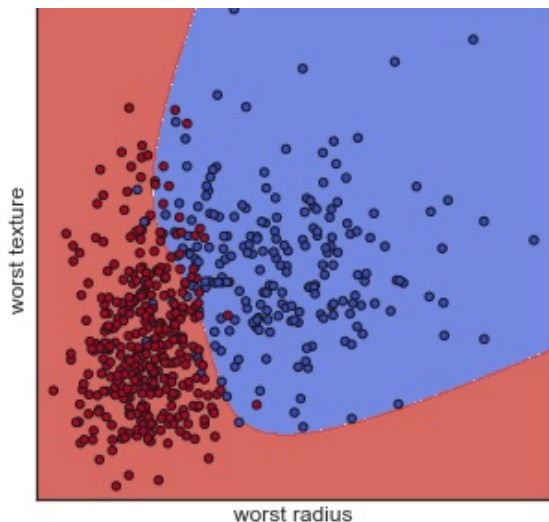
In [111]:

```
plot_cl(SVC(kernel='poly', degree=4, gamma=0.9, C=1.0))
```

<bound method BaseEstimator.\_\_repr\_\_ of SVC(degree=4, gamma=0.9, kernel='poly')>







In [135]:

```
breast_X_train_1, breast_X_test_1, breast_y_train_1, breast_y_test_1 = train_test_split(
    breast.data, breast.target, test_size=0.5, random_state=1)
breast_X_train_1.shape, breast_X_test_1.shape
```

Out[135]:

```
((284, 30), (285, 30))
```

In [192]:

```
svc_1 = SVC()
svc_1.fit(breast_X_train_1, breast_y_train_1)
```

Out[192]:

```
SVC()
```

In [193]:

```
breast_y_pred_1 = svc_1.predict(breast_X_test_1)
```

In [194]:

```
mean_absolute_error(breast_y_test_1, breast_y_pred_1), mean_squared_error(breast_y_test_1,
    breast_y_pred_1)
```

Out[194]:

```
(0.0912280701754386, 0.0912280701754386)
```

## Деревья решений (DecisionTreeClassifier)

In [120]:

```
breast_x_ds = pd.DataFrame(data=breast['data'], columns=breast['feature_names'])
breast_x_ds.head()
```

Out[120]:

	mean radius	mean texture	mean perimeter	mean area	mean smoothness	mean compactness	mean concavity	mean concave points	mean symmetry	mean fractal dimension	...	worst radius	worst texture
0	17.99	10.38	122.80	1001.0	0.11840	0.27760	0.3001	0.14710	0.2419	0.07871	...	25.38	17.3
1	20.57	17.77	132.90	1326.0	0.08474	0.07864	0.0869	0.07017	0.1812	0.05667	...	24.99	23.4
2	19.69	21.25	130.00	1203.0	0.10960	0.15990	0.1974	0.12790	0.2069	0.05999	...	23.57	25.5
3	11.42	20.38	77.58	386.1	0.14250	0.28390	0.2414	0.10520	0.2597	0.09744	...	14.91	26.5
4	20.29	14.34	135.10	1297.0	0.10030	0.13280	0.1980	0.10430	0.1809	0.05883	...	22.54	16.6

◀ ▶

```
# Обучим дерево на всех признаках
breast_tree_cl = DecisionTreeClassifier(random_state=1)
breast_tree_cl.fit(breast_x_ds, breast.target)
breast_tree_cl
```

```
DecisionTreeClassifier(random_state=1)
```

```
from IPython.core.display import HTML
from sklearn.tree.export import export_text
tree_rules = export_text(breast_tree_cl, feature_names=list(breast_x_ds.columns))
HTML('<pre>' + tree_rules + '</pre>')
```

Out[122]:

```
|--- worst radius <= 16.80
| |--- worst concave points <= 0.14
| | |--- worst symmetry <= 0.16
| | | |--- class: 0
| | |--- worst symmetry > 0.16
| | | |--- area error <= 38.60
| | | | |--- smoothness error <= 0.00
| | | | | |--- fractal dimension error <= 0.00
| | | | | |--- class: 0
| | | | | |--- fractal dimension error > 0.00
| | | | | |--- class: 1
| | | | |--- smoothness error > 0.00
| | | | |--- worst texture <= 33.27
| | | | | |--- class: 1
| | | | |--- worst texture > 33.27
| | | | | |--- worst texture <= 33.56
| | | | | | |--- class: 0
| | | | | |--- worst texture > 33.56
| | | | | | |--- class: 1
| | | |--- area error > 38.60
| | | |--- symmetry error <= 0.02
| | | | |--- class: 0
| | | |--- symmetry error > 0.02
| | | | |--- area error <= 39.15
| | | | | |--- class: 0
| | | | |--- area error > 39.15
| | | | | |--- class: 1
| |--- worst concave points > 0.14
| |--- worst texture <= 25.67
| | |--- worst area <= 810.30
| | | |--- mean smoothness <= 0.12
| | | | |--- class: 1
| | | |--- mean smoothness > 0.12
| | | | |--- class: 0
| | |--- worst area > 810.30
```

```

| | | | | |---worst area <= 844.65
| | | | | |--- class: 0
| | | | | |--- worst area > 844.65
| | | | | |--- class: 1
| | |--- worst texture > 25.67
| | | |--- mean concavity <= 0.10
| | | |--- mean texture <= 19.44
| | | |--- class: 1
| | | |--- mean texture > 19.44
| | | |--- class: 0
| | | |--- mean concavity > 0.10
| | | |--- class: 0
|--- worst radius > 16.80
| |--- worst texture <= 19.91
| | |--- worst concave points <= 0.15
| | | |--- class: 1
| | |--- worst concave points > 0.15
| | | |--- class: 0
| |--- worst texture > 19.91
| | |--- worst smoothness <= 0.09
| | | |--- class: 1
| | |--- worst smoothness > 0.09
| | | |--- worst concavity <= 0.18
| | | |--- compactness error <= 0.02
| | | |--- class: 0
| | | |--- compactness error > 0.02
| | | |--- class: 1
| | | |--- worst concavity > 0.18
| | | |--- class: 0

```

In [222]:

```

dot_data = export_graphviz(breast_tree_cl, out_file=None,
                           feature_names=breast.feature_names,
                           class_names=breast.target_names,
                           filled=True, rounded=True, special_characters=True)
graph = graphviz.Source(dot_data)
graph

```

Out[222]:



In [124]:

```

# Важность признаков
list(zip(breast_x_ds.columns.values, breast_tree_cl.feature_importances_))

```

Out[124]:

```

[('mean radius', 0.0),
 ('mean texture', 0.011277152370382112),
 ('mean perimeter', 0.0),
 ('mean area', 0.0),
 ('mean smoothness', 0.007016894808237761),
 ('mean compactness', 0.0),
 ('mean concavity', 0.008771118510297198),
 ('mean concave points', 0.0),
 ('mean symmetry', 0.0),
 ('mean fractal dimension', 0.0),
 ('radius error', 0.0),
 ('texture error', 0.0),
 ('perimeter error', 0.0),
 ('area error', 0.008936807857691388),
 ('smoothness error', 0.0010038401246787545),
 ('compactness error', 0.005638576185191056),

```

```
(
    'concavity error', 0.0),
    ('concave points error', 0.0),
    ('symmetry error', 0.005831348020582203),
    ('fractal dimension error', 0.00644408706878978),
    ('worst radius', 0.6955935182252058),
    ('worst texture', 0.07728413233375948),
    ('worst perimeter', 0.0),
    ('worst area', 0.016724478603093007),
    ('worst smoothness', 0.007387982534891388),
    ('worst compactness', 0.0),
    ('worst concavity', 0.0018358155021552392),
    ('worst concave points', 0.1389382501669036),
    ('worst symmetry', 0.007315997688141106),
    ('worst fractal dimension', 0.0)]
```

In [126]:

```
# Важность признаков в сумме дает почти единицу
sum(breast_tree_cl.feature_importances_)
```

Out[126]:

```
0.9999999999999999
```

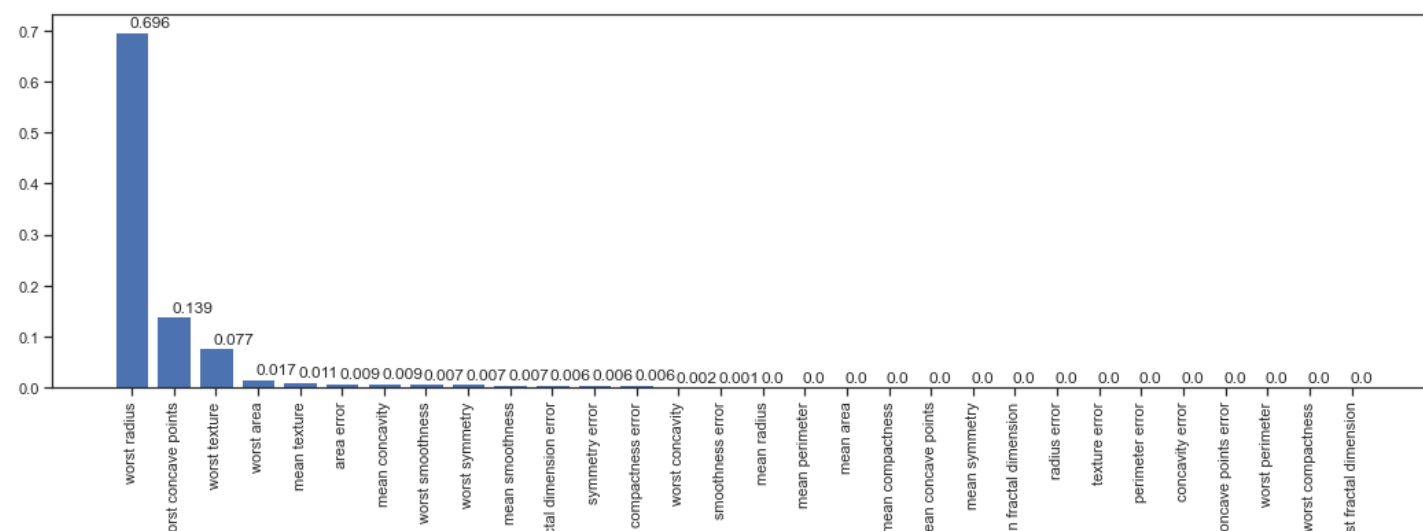
In [127]:

```
from operator import itemgetter

def draw_feature_importances(tree_model, X_dataset, figsize=(18,5)):
    """
    Вывод важности признаков в виде графика
    """
    # Сортировка значений важности признаков по убыванию
    list_to_sort = list(zip(X_dataset.columns.values, tree_model.feature_importances_))
    sorted_list = sorted(list_to_sort, key=itemgetter(1), reverse = True)
    # Названия признаков
    labels = [x for x,_ in sorted_list]
    # Важности признаков
    data = [x for _,x in sorted_list]
    # Вывод графика
    fig, ax = plt.subplots(figsize=figsize)
    ind = np.arange(len(labels))
    plt.bar(ind, data)
    plt.xticks(ind, labels, rotation='vertical')
    # Вывод значений
    for a,b in zip(ind, data):
        plt.text(a-0.05, b+0.01, str(round(b,3)))
    plt.show()
    return labels, data
```

In [128]:

```
breast_tree_cl_fl, breast_tree_cl_fd = draw_feature_importances(breast_tree_cl, breast_x_
ds)
```



In [129]:

```
# Список признаков, отсортированный на основе важности, и значения важности
breast_tree_cl_fl, breast_tree_cl_fd
```

Out [129] :

[illegible]

In [130]:

```
# Пересортируем признаки на основе важности
breast_x_ds_sorted = breast_x_ds[breast_tree_cl_fl]
breast_x_ds_sorted.head()
```

Out[130]:

	worst radius	worst concave points	worst texture	worst area	mean texture	area error	mean concavity	worst smoothness	worst symmetry	mean smoothness	...	mean symmetry	mean fractal dimension
0	25.38	0.2654	17.33	2019.0	10.38	153.40	0.3001	0.1622	0.4601	0.11840	...	0.2419	0.07871
1	24.99	0.1860	23.41	1956.0	17.77	74.08	0.0869	0.1238	0.2750	0.08474	...	0.1812	0.05667
2	23.57	0.2430	25.53	1709.0	21.25	94.03	0.1974	0.1444	0.3613	0.10960	...	0.2069	0.05999
3	14.91	0.2575	26.50	567.7	20.38	27.23	0.2414	0.2098	0.6638	0.14250	...	0.2597	0.09744
4	22.54	0.1625	16.67	1575.0	14.34	94.44	0.1980	0.1374	0.2364	0.10030	...	0.1809	0.05883

5 rows x 30 columns



In [149]:

```
breast_X_train_2, breast_X_test_2, breast_y_train_2, breast_y_test_2 = train_test_split(
    breast_x_ds_sorted, breast.target, test_size=0.5, random_state=1)
breast_X_train_2.shape, breast_X_test_2.shape
```

Out[149]:

((284, 30), (285, 30))

In [150]:

```
# Обучим дерево и предскажем результаты на всех признаках
breast_tree_cl_feat_1 = DecisionTreeClassifier(random_state=1).fit(breast_X_train_2, breast_y_train_2)
breast_y_test_predict = breast_tree_cl_feat_1.predict(breast_X_test_2)
breast_y_test_predict.shape
```

Out[150]:

(285,)

In [151]:

```
# Проверим точность по классам
print_accuracy_score_for_classes(breast_y_test_2, breast_y_test_predict)
```

```
Метка    Accuracy
0    0.8737864077669902
1    0.9010989010989011
```

In [152]:

```
# Обучим дерево и предскажем результаты на единственном самом важном признаке
breast_tree_cl_feat_2 = DecisionTreeClassifier(random_state=1).fit(breast_X_train_2[[breast_tree_cl_fl[0]]], breast_y_train_2)
breast_y_test_predict_2 = breast_tree_cl_feat_2.predict(breast_X_test_2[[breast_tree_cl_fl[0]]])
breast_y_test_predict_2.shape
```

Out[152]:

(285,)

In [153]:

```
# Проверим точность по классам
print_accuracy_score_for_classes(breast_y_test_2, breast_y_test_predict_2)
```

```
Метка    Accuracy
0    0.7766990291262136
1    0.9056012056012056
```

# Качество моделей с помощью двух подходящих для задачи метрик. Сравнение качеств полученных моделей.

В качестве метрик для решения задачи классификации будем использовать:

- Метрика **precision**: *precision*

$$= \frac{TP}{TP+FP}$$

- Метрика **recall** (полнота): *recall*

$$= \frac{TP}{TP+FN}$$

In [207]:

```
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()
```

In [200]:

```
# Логистическая регрессия (LogisticRegression)
precision_score(breast_y_test, pred_breast_y_test), recall_score(breast_y_test, pred_breast_y_test)
```

Out[200]:

```
(0.9144385026737968, 0.9395604395604396)
```

In [201]:

```
# Метод опорных векторов (SVC)
precision_score(breast_y_test_1, breast_y_pred_1), recall_score(breast_y_test_1, breast_y_pred_1)
```

Out[201]:

```
(0.89, 0.978021978021978)
```

In [203]:

```
# Деревья решений (DecisionTreeClassifier)
precision_score(breast_y_test_2, breast_y_test_predict_2), recall_score(breast_y_test_2, breast_y_test_predict_2)
```

Out[203]:

```
(0.8763440860215054, 0.8956043956043956)
```

In [199]:

```
# Логистическая регрессия
accuracy_score(breast_y_test, pred_breast_y_test)
```

Out[199]:

```
0.9052631578947369
```

In [205]:

```
# Метод опорных векторов (SVC)
```



```
accuracy_score(breast_y_test_1, breast_y_pred_1)
```

Out[205]:

0.9087719298245615

In [204]:

```
# Деревья решений (DecisionTreeClassifier)
accuracy_score(breast_y_test_2, breast_y_test_predict_2)
```

Out[204]:

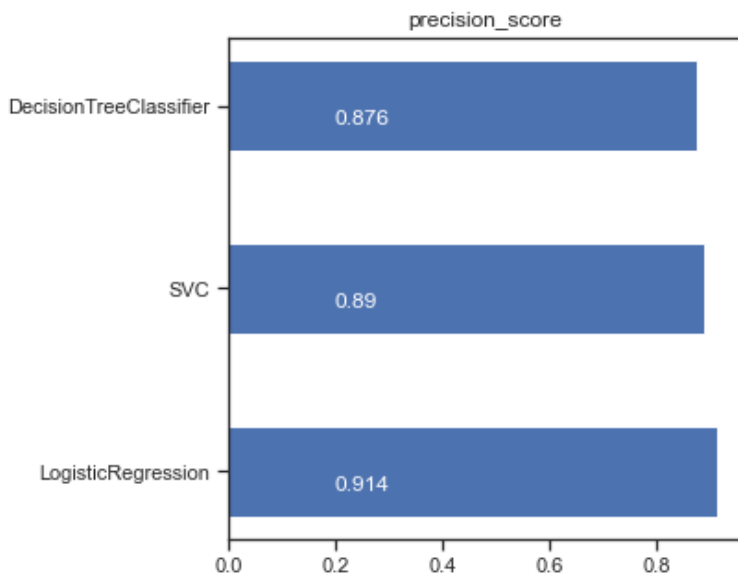
0.8526315789473684

In [231]:

```
# Результаты
array_labels = ['LogisticRegression', 'SVC', 'DecisionTreeClassifier']
array_mae = [precision_score(breast_y_test, pred_breast_y_test),
             precision_score(breast_y_test_1, breast_y_pred_1),
             precision_score(breast_y_test_2, breast_y_test_predict_2)]
```

In [232]:

```
# Визуализация результатов
vis_models_quality(array_mae, array_labels, 'precision_score')
```

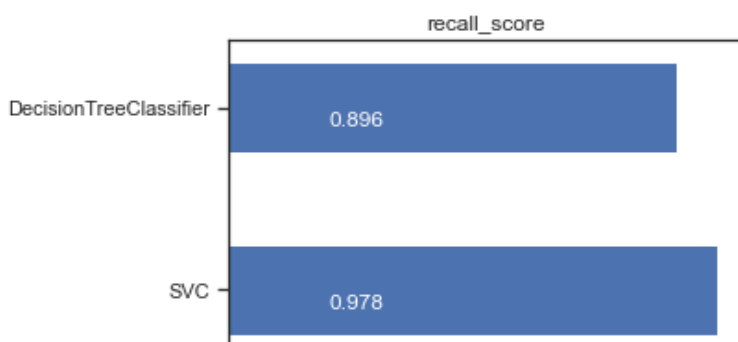


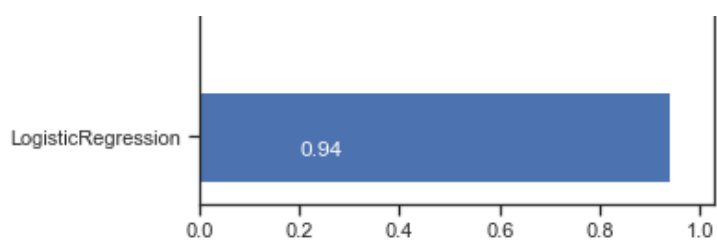
In [233]:

```
# Результаты
array_labels = ['LogisticRegression', 'SVC', 'DecisionTreeClassifier']
array_mae = [recall_score(breast_y_test, pred_breast_y_test),
             recall_score(breast_y_test_1, breast_y_pred_1),
             recall_score(breast_y_test_2, breast_y_test_predict_2)]
```

In [234]:

```
# Визуализация результатов
vis_models_quality(array_mae, array_labels, 'recall_score')
```





In [235]:

```
# Результаты
array_labels = ['LogisticRegression', 'SVC', 'DecisionTreeClassifier']
array_mae = [accuracy_score(breast_y_test, pred_breast_y_test),
             accuracy_score(breast_y_test_1, breast_y_pred_1),
             accuracy_score(breast_y_test_2, breast_y_test_predict_2)]
```

In [236]:

```
# Визуализация результатов
vis_models_quality(array_mae, array_labels, 'accuracy_score')
```

