Домашнее задание 5

In [1]:

```
import matplotlib.pyplot as plt
import random

from matplotlib.colors import ListedColormap
from sklearn import datasets

import numpy as np
```

1. Задание:

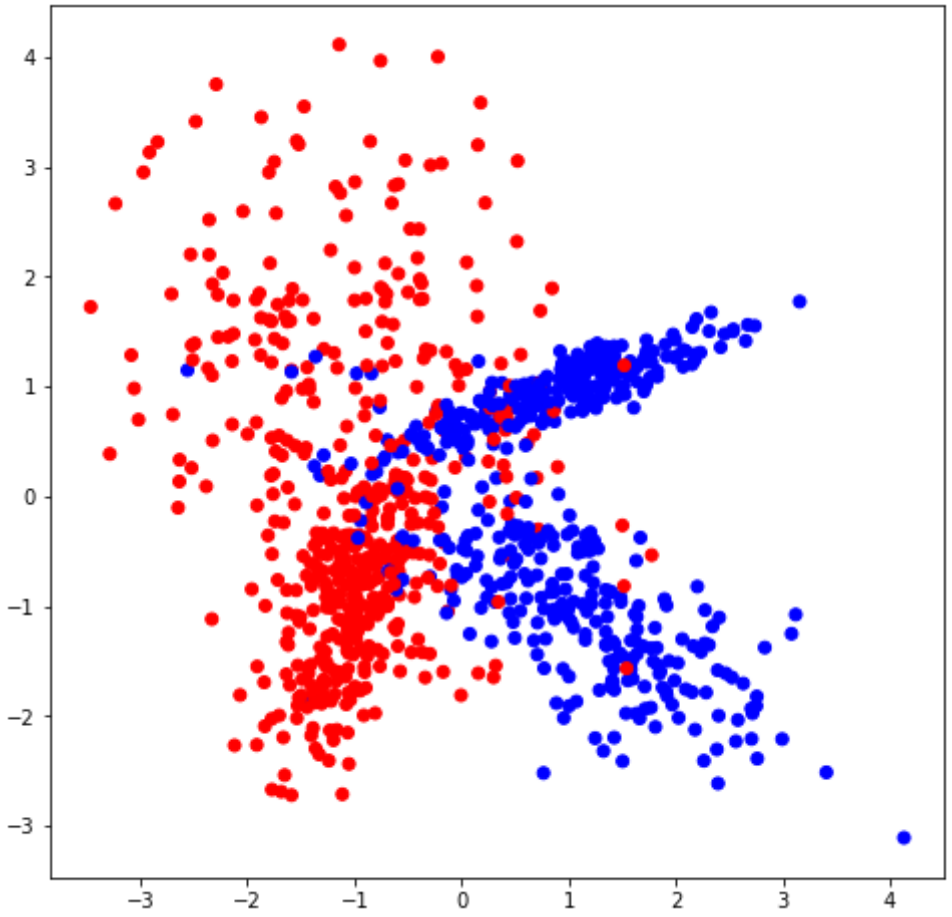
- Сформировать с помощью sklearn.make_classification датасет из 1000 объектов с двумя признаками.
- Обучить случайный лес из 1, 3, 10, 50, 100 и 200 деревьев (за основу взять реализацию построения этого алгоритма из урока).
- Визуализировать их разделяющие линии на графиках (по подобию визуализации деревьев из предыдущего урока, необходимо только заменить вызов функции predict на tree_vote).
- Сделать выводы о получаемой сложности гиперплоскости и недообучении или переобучении случайного леса в зависимости от количества деревьев в нем.

In [2]:

```
# сгенерируем данные, представляющие собой 500 объектов с 5-ю признаками
classification_data, classification_labels = datasets.make_classification(n_samples=100
                                                    n_features=2, n_informative=2,
                                                    n_classes=2, n_redundant=0,
                                                    n_clusters_per_class=2,
                                                    random_state=6)
```

In [3]:

```
# визуализируем сгенерированные данные

colors = ListedColormap(['red', 'blue'])
light_colors = ListedColormap(['lightcoral', 'lightblue'])

plt.figure(figsize=(8,8))
plt.scatter(list(map(lambda x: x[0], classification_data)), list(map(lambda x: x[1], cl
              c=classification_labels, cmap=colors);


```



In [4]:

```
random.seed(42)

def get_bootstrap(data, labels, N):
    n_samples = data.shape[0]
    bootstrap = []

    for i in range(N):
        b_data = np.zeros(data.shape)
        b_labels = np.zeros(labels.shape)

        for j in range(n_samples):
            sample_index = random.randint(0, n_samples-1)
            b_data[j] = data[sample_index]
            b_labels[j] = labels[sample_index]
        bootstrap.append((b_data, b_labels))

    return bootstrap
```

In [5]:

```
def get_subsample(len_sample):
    # будем сохранять не сами признаки, а их индексы
    sample_indexes = [i for i in range(len_sample)]

    len_subsample = int(np.sqrt(len_sample))
    subsample = []

    random.shuffle(sample_indexes)
    for _ in range(len_subsample):
        subsample.append(sample_indexes.pop())

    return subsample
```

Далее повторим реализацию построения дерева решений из предыдущего урока с некоторыми изменениями

In [6]:

```
# Реализуем класс узла

class Node:

    def __init__(self, index, t, true_branch, false_branch):
        self.index = index  # индекс признака, по которому ведется сравнение с порогом
        self.t = t  # значение порога
        self.true_branch = true_branch  # поддерево, удовлетворяющее условию в узле
        self.false_branch = false_branch  # поддерево, не удовлетворяющее условию в узл
```

In [7]:

```python
# И класс терминального узла (листа)

class Leaf:

    def __init__(self, data, labels):
        self.data = data
        self.labels = labels
        self.prediction = self.predict()

    def predict(self):
        # подсчет количества объектов разных классов
        classes = {}  # сформируем словарь "класс: количество объектов"
        for label in self.labels:
            if label not in classes:
                classes[label] = 0
            classes[label] += 1
        #  найдем класс, количество объектов которого будет максимальным в этом листе и
        prediction = max(classes, key=classes.get)
        return prediction
```

In [8]:

```python
# Расчет критерия Джини

def gini(labels):
    #  подсчет количества объектов разных классов
    classes = {}
    for label in labels:
        if label not in classes:
            classes[label] = 0
        classes[label] += 1

    #  расчет критерия
    impurity = 1
    for label in classes:
        p = classes[label] / len(labels)
        impurity -= p ** 2

    return impurity
```

In [9]:

```python
# Расчет качества

def quality(left_labels, right_labels, current_gini):

    # доля выбоки, ушедшая в левое поддерево
    p = float(left_labels.shape[0]) / (left_labels.shape[0] + right_labels.shape[0])

    return current_gini - p * gini(left_labels) - (1 - p) * gini(right_labels)
```

In [10]:

```python
# Разбиение датасета в узле

def split(data, labels, index, t):

    left = np.where(data[:, index] <= t)
    right = np.where(data[:, index] > t)

    true_data = data[left]
    false_data = data[right]
    true_labels = labels[left]
    false_labels = labels[right]

    return true_data, false_data, true_labels, false_labels
```

In [11]:

```python
# Нахождение наилучшего разбиения

def find_best_split(data, labels):

    #  обозначим минимальное количество объектов в узле
    min_leaf = 1

    current_gini = gini(labels)

    best_quality = 0
    best_t = None
    best_index = None

    n_features = data.shape[1]

    # выбор индекса из подвыборки длиной sqrt(n_features)
    subsample = get_subsample(n_features)

    for index in subsample:
        # будем проверять только уникальные значения признака, исключая повторения
        t_values = np.unique([row[index] for row in data])

        for t in t_values:
            true_data, false_data, true_labels, false_labels = split(data, labels, inde
            #  пропускаем разбиения, в которых в узле остается менее 1 объекта
            if len(true_data) < min_leaf or len(false_data) < min_leaf:
                continue

            current_quality = quality(true_labels, false_labels, current_gini)

            #  выбираем порог, на котором получается максимальный прирост качества
            if current_quality > best_quality:
                best_quality, best_t, best_index = current_quality, t, index

    return best_quality, best_t, best_index
```

In [12]:

```python
# Построение дерева с помощью рекурсивной функции

def build_tree(data, labels):

    quality, t, index = find_best_split(data, labels)

    #  Базовый случай - прекращаем рекурсию, когда нет прироста в качества
    if quality == 0:
        return Leaf(data, labels)

    true_data, false_data, true_labels, false_labels = split(data, labels, index, t)

    # Рекурсивно строим два поддерева
    true_branch = build_tree(true_data, true_labels)
    false_branch = build_tree(false_data, false_labels)

    # Возвращаем класс узла со всеми поддеревьями, то есть целого дерева
    return Node(index, t, true_branch, false_branch)
```

Теперь добавим функцию формирования случайного леса.

In [13]:

```python
def random_forest(data, labels, n_trees):
    forest = []
    bootstrap = get_bootstrap(data, labels, n_trees)

    for b_data, b_labels in bootstrap:
        forest.append(build_tree(b_data, b_labels))

    return forest
```

In [14]:

```python
# Функция классификации отдельного объекта

def classify_object(obj, node):

    #  Останавливаем рекурсию, если достигли листа
    if isinstance(node, Leaf):
        answer = node.prediction
        return answer

    if obj[node.index] <= node.t:
        return classify_object(obj, node.true_branch)
    else:
        return classify_object(obj, node.false_branch)
```

In [15]:

```
# функция формирования предсказания по выборке на одном дереве

def predict(data, tree):

    classes = []
    for obj in data:
        prediction = classify_object(obj, tree)
        classes.append(prediction)
    return classes
```

In [16]:

```
# предсказание голосованием деревьев

def tree_vote(forest, data):

    # добавим предсказания всех деревьев в список
    predictions = []
    for tree in forest:
        predictions.append(predict(data, tree))

    # сформируем список с предсказаниями для каждого объекта
    predictions_per_object = list(zip(*predictions))

    # выберем в качестве итогового предсказания для каждого объекта то,
    # за которое проголосовало большинство деревьев
    voted_predictions = []
    for obj in predictions_per_object:
        voted_predictions.append(max(set(obj), key=obj.count))

    return voted_predictions
```

Далее мы сделаем обычное разбиение выборки на обучающую и тестовую, как это делалось ранее.

In [17]:

```
# Разобьем выборку на обучающую и тестовую

from sklearn import model_selection

train_data, test_data, train_labels, test_labels = model_selection.train_test_split(cla
                                                                                    cl
                                                                                    te
                                                                                    ra
```

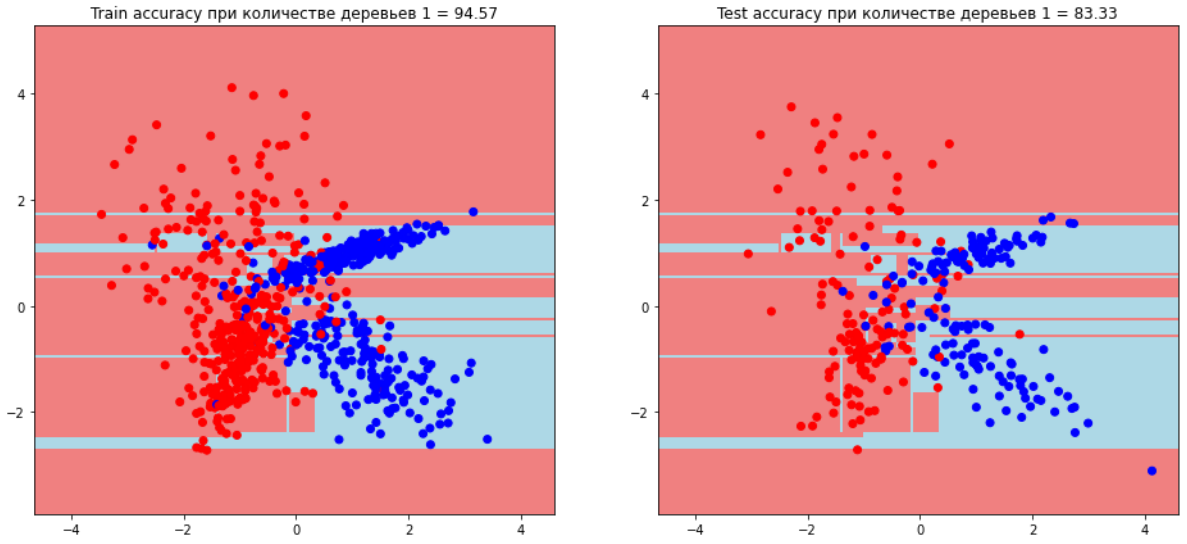
In [18]:

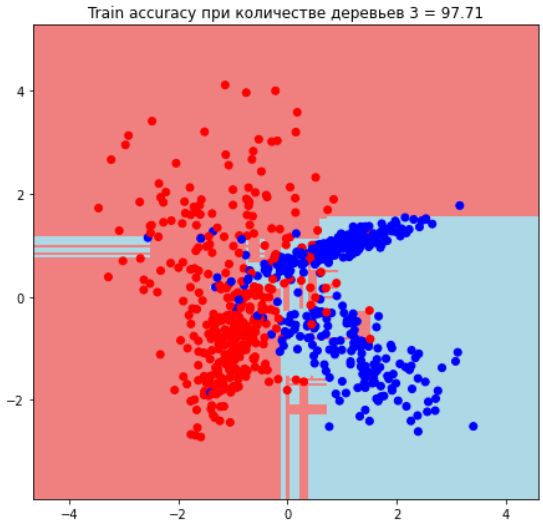
```
# Введем функцию подсчета точности как доли правильных ответов
def accuracy_metric(actual, predicted):
    correct = 0
    for i in range(len(actual)):
        if actual[i] == predicted[i]:
            correct += 1
    return correct / float(len(actual)) * 100.0
```

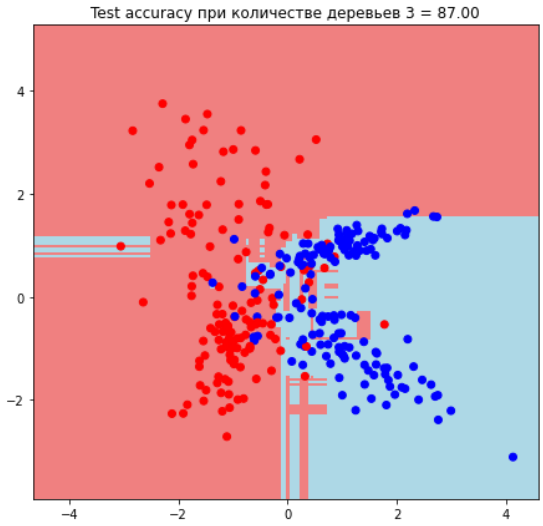
Теперь построим несколько случайных лесов с разным количеством деревьев в них.

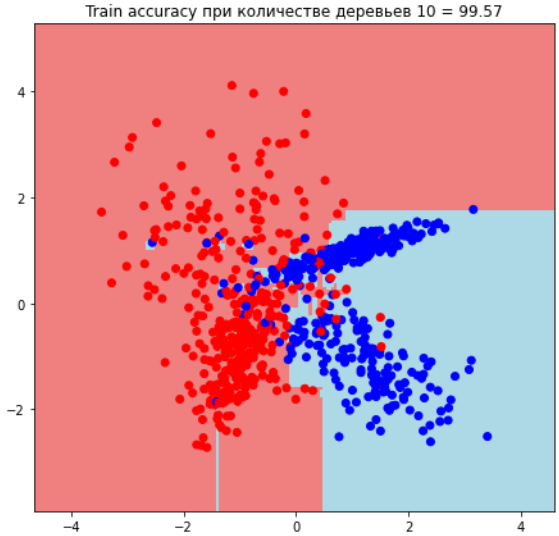
In [19]:

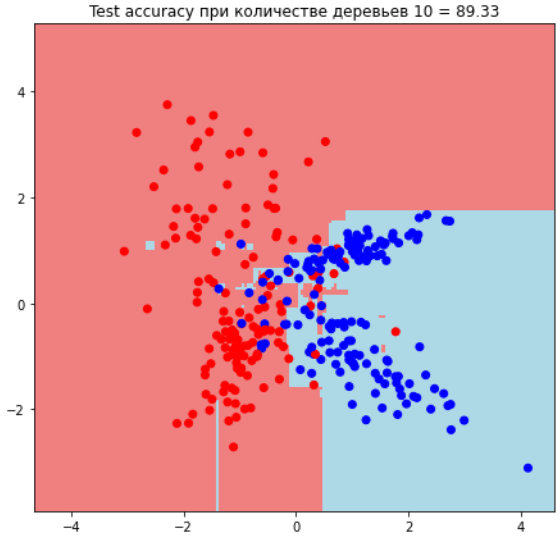
```
def get_meshgrid(data, step=.05, border=1.2):
    x_min, x_max = data[:, 0].min() - border, data[:, 0].max() + border
    y_min, y_max = data[:, 1].min() - border, data[:, 1].max() + border
    return np.meshgrid(np.arange(x_min, x_max, step), np.arange(y_min, y_max, step))

for i in [1, 3, 10, 50, 100, 200]:
    my_forest = random_forest(train_data, train_labels, i)
    train_answers = tree_vote(my_forest, train_data)
    test_answers = tree_vote(my_forest, test_data)
    train_accuracy = accuracy_metric(train_labels, train_answers)
    test_accuracy = accuracy_metric(test_labels, test_answers)
    plt.figure(figsize = (16, 7))

    # график обучающей выборки
    plt.subplot(1,2,1)
    xx, yy = get_meshgrid(train_data)
    mesh_predictions = np.array(tree_vote(my_forest, np.c_[xx.ravel(), yy.ravel()])).re
    plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
    plt.scatter(train_data[:, 0], train_data[:, 1], c = train_labels, cmap = colors)
    plt.title(f'Train accuracy при количестве деревьев {i} = {train_accuracy:.2f}')

    # график тестовой выборки
    plt.subplot(1,2,2)
    plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
    plt.scatter(test_data[:, 0], test_data[:, 1], c = test_labels, cmap = colors)
    plt.title(f'Test accuracy при количестве деревьев {i} = {test_accuracy:.2f}')

```

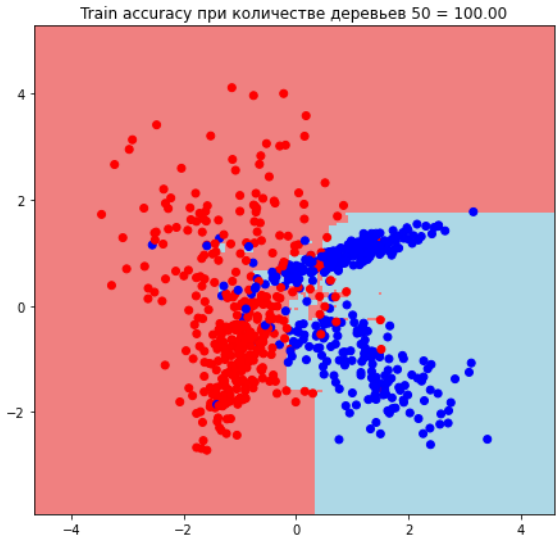


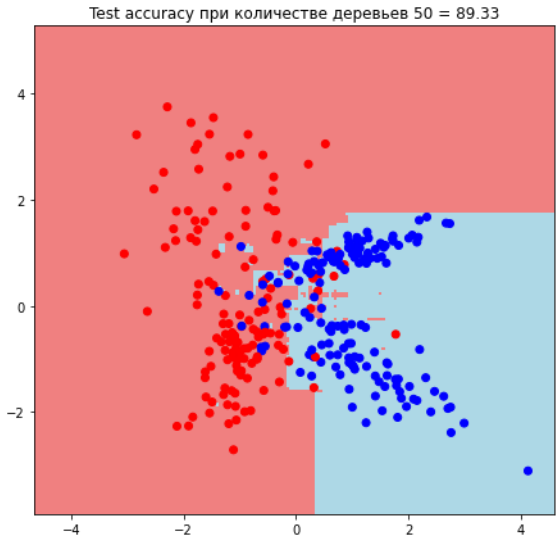

Train accuracy при количестве деревьев 3 = 97.71

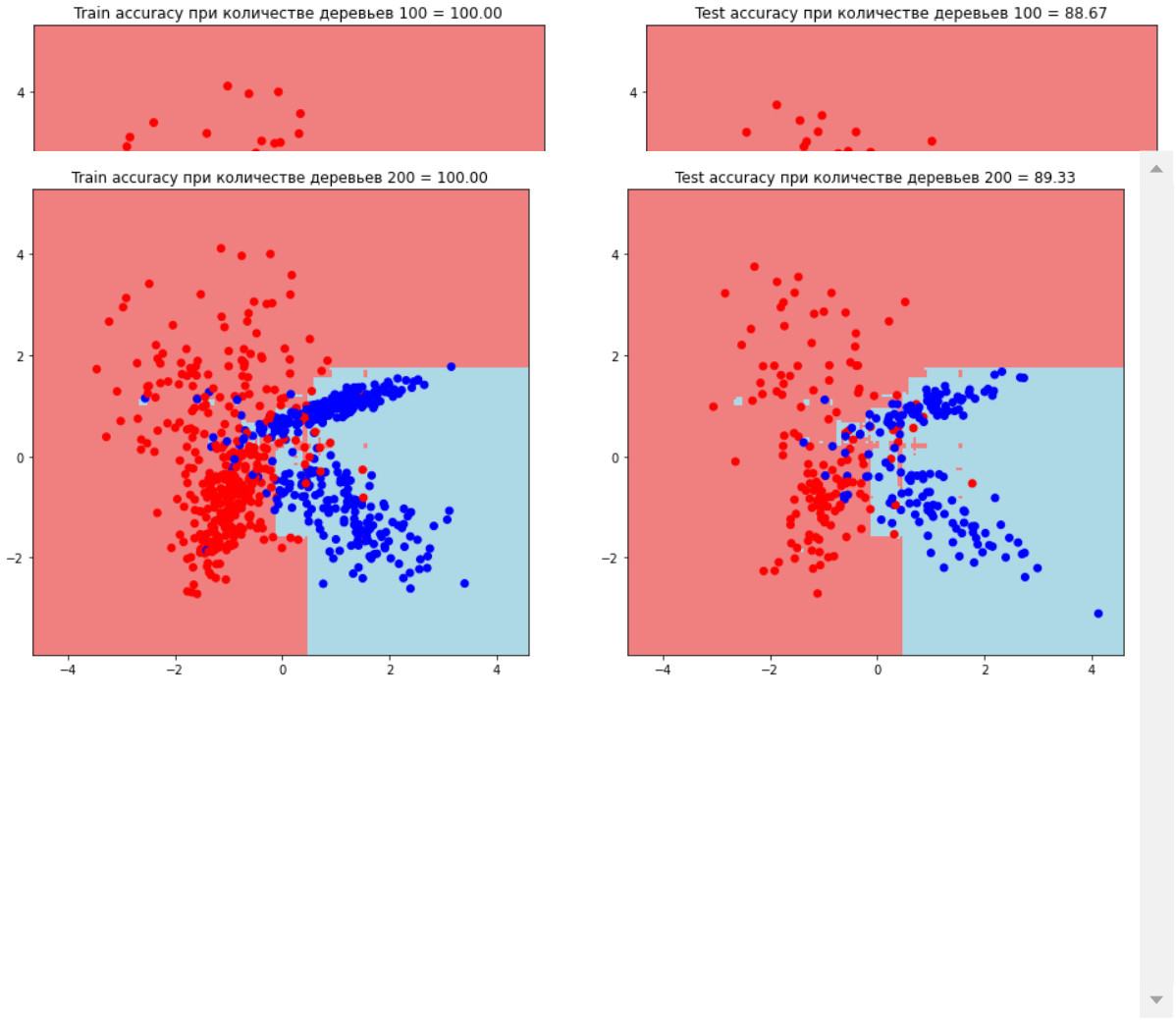

Test accuracy при количестве деревьев 3 = 87.00


Train accuracy при количестве деревьев 10 = 99.57


Test accuracy при количестве деревьев 10 = 89.33


Train accuracy при количестве деревьев 50 = 100.00


Test accuracy при количестве деревьев 50 = 89.33



Выводы:

1. Из графиков видно, что получаемая сложность гиперплоскости при увеличении кличества деревьев станвиться менее рваной и граница между классами более четкой.
2. Начиная с 50 деревьев точность на обучающей выборке становиться 100. На 100 деревьях точность тестовой выборки снижается, намекая на переобучение, но уже на 200 деревьях точность тестовой выборке становиться максимальной. Что еще раз подтвеждает факт того, что случайный лес очень устойчив к переобучени.

2. (*) Заменить в реализованном алгоритме проверку с помощью отложенной выборки на Out-of-Bag.

Для этого необходимо внести небольшие измененя в код

In [20]:

```
random.seed(42)

def get_bootstrap(data, labels, N):
    n_samples = data.shape[0]
    bootstrap = []
    bootstrap_oobe = []
    sample_index_com = []

    for i in range(N):
        b_data = np.zeros(data.shape)
        b_labels = np.zeros(labels.shape)

        for j in range(n_samples):
            sample_index = random.randint(0, n_samples-1)
#             Запоминаем выбранные индексы
            sample_index_com.append(sample_index)
            b_data[j] = data[sample_index]
            b_labels[j] = labels[sample_index]
#         Формируем список индексов, которые не участвовали в формировании бутстрап вы
        oobe_index = list(set([i for i in range(100)]) - set(sample_index_com))
#         Формируем на основе этих индексов выборки объетов и классов, которые не попал
        b_data_oobe = data[oobe_index]
        b_labels_oobe = labels[oobe_index]
        bootstrap.append((b_data, b_labels))
#         Фомируем наборы oobe выборок, в соответсвии с количествтом деревьев
        bootstrap_oobe.append((b_data_oobe, b_labels_oobe))

    return bootstrap, bootstrap_oobe
```

In [21]:

```
def random_forest(data, labels, n_trees):
    forest = []
    bootstrap, bootstrap_oobe = get_bootstrap(data, labels, n_trees)

    for b_data, b_labels in bootstrap:
        forest.append(build_tree(b_data, b_labels))
    # Допишев возвращаемые параметры
    return forest, bootstrap, bootstrap_oobe
```

Построим лес из одного дерева

In [22]:

```
n_trees = 1
my_forest_1, bootstrap, bootstrap_oobe = random_forest(classification_data, classificat
```

In [23]:

```
# Получим ответы для обучающей выборки
train_answers_1 = tree_vote(my_forest_1, classification_data)
```

In [24]:

```
# Точность на обучающей выборке
train_accuracy_1 = accuracy_metric(classification_labels, train_answers_1)
print(f'Точность случайного леса из {n_trees} деревьев на обучающей выборке: {train_acc
```

```
Точность случайного леса из 1 деревьев на обучающей выборке: 93.700
```

In [25]:

```
# Получим ответы для oobe выборки
oobe_answers_1 = tree_vote(my_forest_1, bootstrap_oobe[0][0])
```

In [26]:

```
# Точность на oobe выборке
oobe_accuracy_1 = accuracy_metric(bootstrap_oobe[0][1], oobe_answers_1)
print(f'Точность случайного леса из {n_trees} деревьев на oobe выборке: {oobe_accuracy_
```

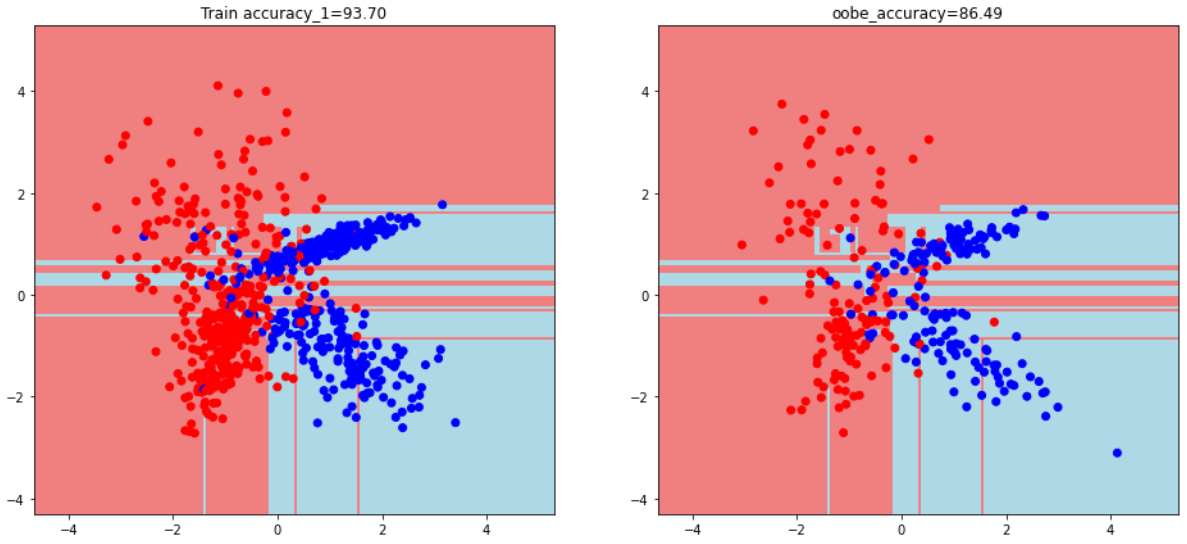
```
Точность случайного леса из 1 деревьев на oobe выборке: 86.486
```

In [27]:

```
# Визуализируем дерево на графике

def get_meshgrid(data, step=.05, border=1.2):
    x_min, x_max = data[:, 0].min() - border, data[:, 0].max() + border
    y_min, y_max = data[:, 1].min() - border, data[:, 1].max() + border
    return np.meshgrid(np.arange(x_min, x_max, step), np.arange(y_min, y_max, step))

plt.figure(figsize = (16, 7))

# график обучающей выборки на 1 дереве
plt.subplot(1,2,1)
xx, yy = get_meshgrid(classification_data)
mesh_predictions = np.array(tree_vote(my_forest_1, np.c_[xx.ravel(), yy.ravel()])).resh
plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
plt.scatter(train_data[:, 0], train_data[:, 1], c = train_labels, cmap = colors)
plt.title(f'Train accuracy_1={train_accuracy_1:.2f}')

# график тестовой выборки 1 дереве
plt.subplot(1,2,2)
plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
plt.scatter(test_data[:, 0], test_data[:, 1], c = test_labels, cmap = colors)
plt.title(f'oobe_accuracy={oobe_accuracy_1:.2f}')
```

Out[27]:

```
Text(0.5, 1.0, 'oobe_accuracy=86.49')
```



Построим лес из трех деревьев

In [28]:

```
n_trees = 3
my_forest_3, bootstrap, bootstrap_oobe = random_forest(classification_data, classificat
```

In [29]:

```
# Получим ответы для обучающей выборки
train_answers_3 = tree_vote(my_forest_3, classification_data)
```

In [30]:

```
# Точность на обучающей выборке
train_accuracy_3 = accuracy_metric(classification_labels, train_answers_3)
print(f'Точность случайного леса из {n_trees} деревьев на обучающей выборке: {train_acc
```

```
Точность случайного леса из 3 деревьев на обучающей выборке: 96.800
```

In [31]:

```
# Получим ответы для oobe выборки
oobe_answers_3 = tree_vote(my_forest_3, bootstrap_oobe[0][0])
```

In [32]:

```
# Точность на oobe выборке
oobe_accuracy_3 = accuracy_metric(bootstrap_oobe[0][1], oobe_answers_3)
print(f'Точность случайного леса из {n_trees} деревьев на oobe выборке: {oobe_accuracy_
```

```
Точность случайного леса из 3 деревьев на oobe выборке: 92.500
```

In [33]:

```
# Визуализируем дерево на графике

def get_meshgrid(data, step=.05, border=1.2):
    x_min, x_max = data[:, 0].min() - border, data[:, 0].max() + border
    y_min, y_max = data[:, 1].min() - border, data[:, 1].max() + border
    return np.meshgrid(np.arange(x_min, x_max, step), np.arange(y_min, y_max, step))

plt.figure(figsize = (16, 7))

# график обучающей выборки на 1 дереве
plt.subplot(1,2,1)
xx, yy = get_meshgrid(classification_data)
mesh_predictions = np.array(tree_vote(my_forest_3, np.c_[xx.ravel(), yy.ravel()])).resh
plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
plt.scatter(train_data[:, 0], train_data[:, 1], c = train_labels, cmap = colors)
plt.title(f'Train accuracy_3={train_accuracy_3:.2f}')

# график тестовой выборки 1 дереве
plt.subplot(1,2,2)
plt.pcolormesh(xx, yy, mesh_predictions, cmap = light_colors)
plt.scatter(test_data[:, 0], test_data[:, 1], c = test_labels, cmap = colors)
plt.title(f'oobe_accuracy_3={oobe_accuracy_3:.2f}')
```

Out[33]:

```
Text(0.5, 1.0, 'oobe_accuracy_3=92.50')
```