**Липецкий государственный технический университет**

Факультет автоматизации и информатики

Кафедра автоматизированных систем управления

ЛАБОРАТОРНАЯ РАБОТА №2

по дисциплине

«Прикладные интеллектуальные системы и экспертные системы»

Студент Первушин О. С.

Группа М-ИАП-23-1

Руководитель Кургасов В. В.

доцент, канд. пед. наук

Липецк 2023 г.

Цель работы

Получить практические навыки решения задачи бинарной классификации данных в среде Jupiter Notebook. Научиться загружать данные, обучать классификаторы и проводить классификацию. Научиться оценивать точность полученных моделей.

Задание кафедры

1. В среде Jupiter Notebook создать новый ноутбук (Notebook)
2. Импортировать необходимые для работы библиотеки и модули
3. Загрузить данные в соответствии с вариантом
4. Вывести первые 15 элементов выборки (координаты точек и метки класса)
5. Отобразить на графике сгенерированную выборку. Объекты разных классов должны иметь разные цвета.
6. Разбить данные на обучающую (train) и тестовую (test) выборки в пропорции 75% — 25% соответственно.
7. Отобразить на графике обучающую и тестовую выборки. Объекты разных классов должны иметь разные цвета.
8. Реализовать модели классификаторов, обучить их на обучающем множестве. Применить модели на тестовой выборке, вывести результаты классификации:
   * Истинные и предсказанные метки классов
   * Матрицу ошибок (confusion matrix)
   * Значения полноты, точности, f1-меры и аккуратности
   * Значение площади под кривой ошибок (AUC ROC)
   * Отобразить на графике область принятия решений по каждому классу

В качестве методов классификации использовать:

* + 1. Метод к-ближайших соседей (n\_neighbors = {1, 3, 5, 9})
    2. Наивный байесовский метод
    3. Случайный лес (n\_estimators = {5, 10, 15, 20, 50})

1. По каждому пункту работы занести в отчет программный код и результат вывода.
2. По результатам п.8 занести в отчет таблицу с результатами классификации всеми методами и выводы о наиболее подходящем методе классификации ваших данных.
3. Изучить, как изменится качество классификации, если на тестовую часть выделить 10% выборки, 35% выборки. Для этого повторить п.п. 6 – 10.

Ход работы

На Листинге 1 представлены все необходимые и импортированные библиотеки.

Листинг 1 — Импортированные библиотеки

# импортируем библиотеки

import numpy as np

import matplotlib.pyplot as plt

from sklearn.datasets import make\_blobs

from sklearn.metrics import confusion\_matrix

from sklearn.metrics import classification\_report

from sklearn.metrics import accuracy\_score

from sklearn.model\_selection import train\_test\_split

from sklearn.linear\_model import LogisticRegression

from sklearn.neighbors import KNeighborsClassifier

from sklearn.ensemble import RandomForestClassifier

from sklearn.naive\_bayes import GaussianNB

from sklearn.metrics import roc\_auc\_score

from matplotlib.colors import ListedColormap, colorConverter, LinearSegmentedColormap

Загрузим данные в соответствии с вариантом (Листинг 2).

Листинг 2 — Загрузка данных

# генерируем выборку в соответствии с вариантом

centers = 2

random\_state = 28

cluster\_std = 4.5

x, y = make\_blobs(centers = centers, random\_state = random\_state, cluster\_std = cluster\_std, shuffle = 1)

Выведем первые пятнадцать координат и меток. Код, реализующий вывод представлен в листинге 3, вывод представлен на рисунке 1.

Листинг 3 — Вывод первых 15-ти элементов

# выводим первые 15 координат и меток

print ("Координаты точек: ")

print (x[:15])

print ("Метки класса: ")

print (y[:15])

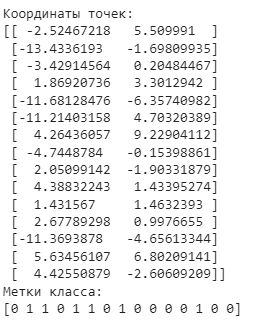


Рисунок 1 — Координаты точек и метки класса 15-ти первых элементов

Код, реализующий построение графика приведен в листинге 4; график изображен на рисунке 2.

Листинг 4 — Построение графика

# выводим сгенерированные данные по меткам

plt.scatter (x[:, 0], x[:, 1], c = y)

plt.show()

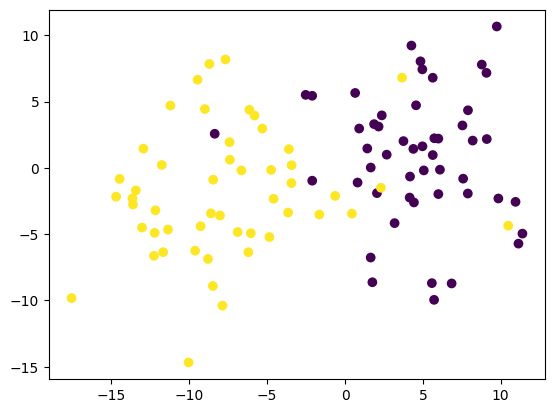


Рисунок 2 — График

Напишем функцию (листинг 5), реализующую метод k-ближних соседей с принимаемыми параметрами.

Листинг 5 — Функция KNeighbors

# Метод к-ближайших соседей

def KNeighbors(x, y, test\_size = 0.25, n\_neighbors = 5):

  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = test\_size, random\_state = 1)

  knn = KNeighborsClassifier(n\_neighbors = n\_neighbors, metric = 'euclidean')

  knn.fit(x\_train, y\_train)

  prediction = knn.predict(x\_test)

  print ('Prediction and test:')

  print ('Prediction: \t', prediction)

  print ('Test: \t\t', y\_test)

  print("\n\n")

  print ('Confusion matrix: ')

  print (confusion\_matrix(y\_test, prediction)[0])

  print (confusion\_matrix(y\_test, prediction)[1])

  print("\n\n")

  print ('Accuracy score: ', accuracy\_score(prediction, y\_test))

  print("\n\n")

  print('Classification Report\n', classification\_report(y\_test, prediction))

  print("\n\n")

  print('ROC AUC')

  print(roc\_auc\_score(y\_test, prediction))

  print("\n\n")

  # обучающая и тестовая выборки

  plt.title('Division into training (Blue) and test (Red) samples')

  plt.scatter (x\_train[:, 0], x\_train[:, 1], color = 'blue')

  plt.scatter (x\_test[:, 0], x\_test[:, 1], color = 'red')

  plt.show()

  plt.xlabel("first feature")

  plt.ylabel("second feature")

  plot\_2d\_separator(knn, x, fill=True)

  plt.scatter(x[:, 0], x[:, 1], c=y, s=70)

Напишем функцию (листинг 6), реализующую наивный байесовский метод с принимаемыми параметрами.

Листинг 6 — Функция naiveBayes

# Наивный байесовский метод

def naiveBayes(x, y, test\_size = 0.25):

  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = test\_size, random\_state = 1)

  gnb = GaussianNB()

  gnb.fit(x\_train, y\_train)

  prediction = gnb.predict(x\_test)

  print ('Prediction and test:')

  print ('Prediction: \t', prediction)

  print ('Test: \t\t', y\_test)

  print("\n\n")

  print ('Confusion matrix: ')

  print (confusion\_matrix(y\_test, prediction)[0])

  print (confusion\_matrix(y\_test, prediction)[1])

  print("\n\n")

  print ('Accuracy score: ', accuracy\_score(prediction, y\_test))

  print("\n\n")

  print('Classification Report\n', classification\_report(y\_test, prediction))

  print("\n\n")

  print('ROC AUC')

  print(roc\_auc\_score(y\_test, prediction))

  print("\n\n")

  # обучающая и тестовая выборки

  plt.title('Division into training (Blue) and test (Red) samples')

  plt.scatter (x\_train[:, 0], x\_train[:, 1], color = 'blue')

  plt.scatter (x\_test[:, 0], x\_test[:, 1], color = 'red')

  plt.show()

  plt.xlabel("first feature")

  plt.ylabel("second feature")

  plot\_2d\_separator(gnb, x, fill=True)

  plt.scatter(x[:, 0], x[:, 1], c=y, s=70)

Напишем функцию (листинг 7), реализующую метод случайного леса с принимаемыми параметрами.

Листинг 7 — Функция randomForest

# Случайный лес

def randomForest(x, y, test\_size = 0.25, n\_estimators = 100):

  x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = test\_size, random\_state = 1)

  rfc = RandomForestClassifier(n\_estimators = n\_estimators)

  rfc.fit(x\_train, y\_train)

  prediction = rfc.predict(x\_test)

  print ('Prediction and test:')

  print ('Prediction: \t', prediction)

  print ('Test: \t\t', y\_test)

  print("\n\n")

  print ('Confusion matrix: ')

  print (confusion\_matrix(y\_test, prediction)[0])

  print (confusion\_matrix(y\_test, prediction)[1])

  print("\n\n")

  print ('Accuracy score: ', accuracy\_score(prediction, y\_test))

  print("\n\n")

  print('Classification Report\n', classification\_report(y\_test, prediction))

  print("\n\n")

  print('ROC AUC')

  print(roc\_auc\_score(y\_test, prediction))

  print("\n\n")

  # обучающая и тестовая выборки

  plt.title('Division into training (Blue) and test (Red) samples')

  plt.scatter (x\_train[:, 0], x\_train[:, 1], color = 'blue')

  plt.scatter (x\_test[:, 0], x\_test[:, 1], color = 'red')

  plt.show()

  plt.xlabel("first feature")

  plt.ylabel("second feature")

  plot\_2d\_separator(rfc, x, fill=True)

  plt.scatter(x[:, 0], x[:, 1], c=y, s=70)

Проведем классификацию методом k-ближних соседей с параметрами по умолчанию. Графики выхода представлены на рисунках 3 — 4. Вывод функции представлен в листинге 8.

Листинг 8 — Выходные данные

Prediction and test:

Prediction: [1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 1 0 0 0]

Test: [1 1 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 1 1 0 0]

Confusion matrix:

[13 0]

[ 2 10]

Accuracy score: 0.92

Classification Report

precision recall f1-score support

0 0.87 1.00 0.93 13

1 1.00 0.83 0.91 12

accuracy 0.92 25

macro avg 0.93 0.92 0.92 25

weighted avg 0.93 0.92 0.92 25

ROC AUC

0.9166666666666667

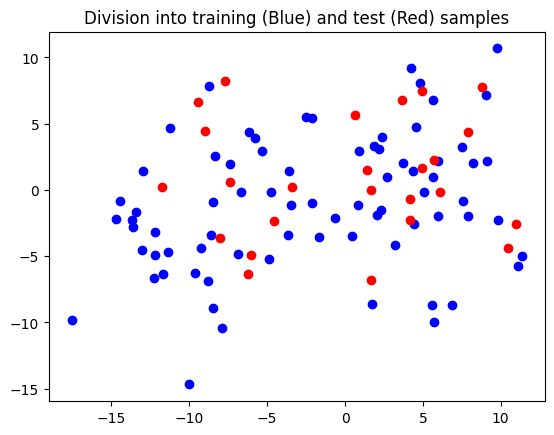


Рисунок 3 — Обучающая и тестовая выборки

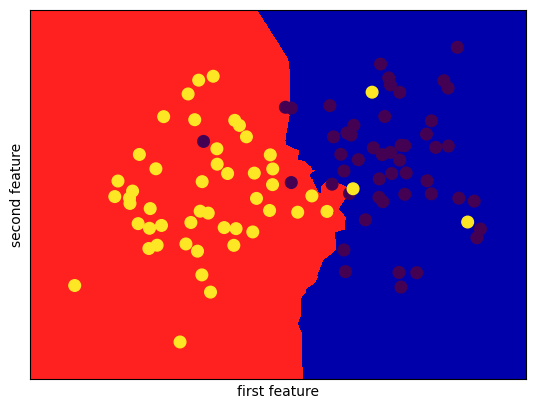


Рисунок 4 — Области принятия решений

Проведем классификацию наивным байесовским методом с параметрами по умолчанию. Графики выхода представлены на рисунках 5 — 6. Вывод функции представлен в листинге 9.

Листинг 9 — Выходные данные

Prediction and test:

Prediction: [1 1 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 1 0 0 0]

Test: [1 1 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 1 1 0 0]

Confusion matrix:

[13 0]

[ 2 10]

Accuracy score: 0.92

Classification Report

precision recall f1-score support

0 0.87 1.00 0.93 13

1 1.00 0.83 0.91 12

accuracy 0.92 25

macro avg 0.93 0.92 0.92 25

weighted avg 0.93 0.92 0.92 25

ROC AUC

0.9166666666666667

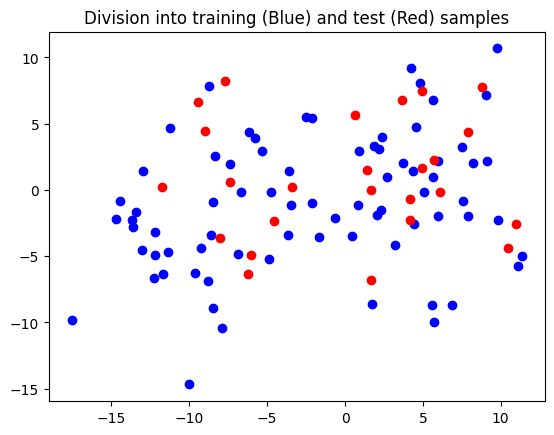


Рисунок 5 — Обучающая и тестовая выборки

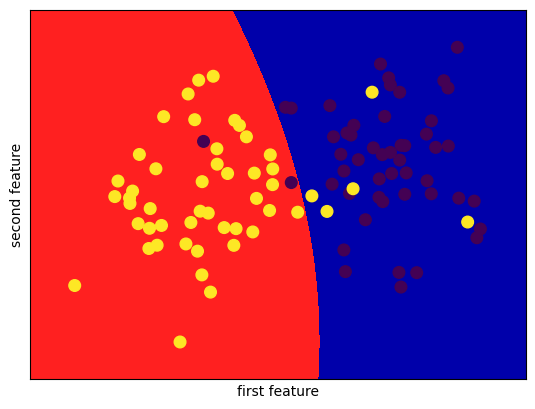


Рисунок 6 — Области принятия решений

Проведем классификацию методом случайного леса с параметром «n\_estimators» = 10. Графики выхода представлены на рисунках 7 — 8. Вывод функции представлен в листинге 10.

Листинг 10 — Выходные данные

Prediction and test:

Prediction: [1 0 0 1 0 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 1 0 0 0]

Test: [1 1 0 1 1 1 0 0 0 0 0 1 0 0 0 1 0 1 1 0 1 1 1 0 0]

Confusion matrix:

[13 0]

[3 9]

Accuracy score: 0.88

Classification Report

precision recall f1-score support

0 0.81 1.00 0.90 13

1 1.00 0.75 0.86 12

accuracy 0.88 25

macro avg 0.91 0.88 0.88 25

weighted avg 0.90 0.88 0.88 25

ROC AUC

0.875

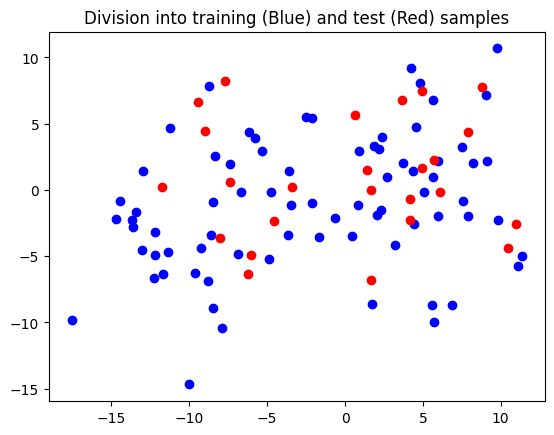


Рисунок 7 — Обучающая и тестовая выборки

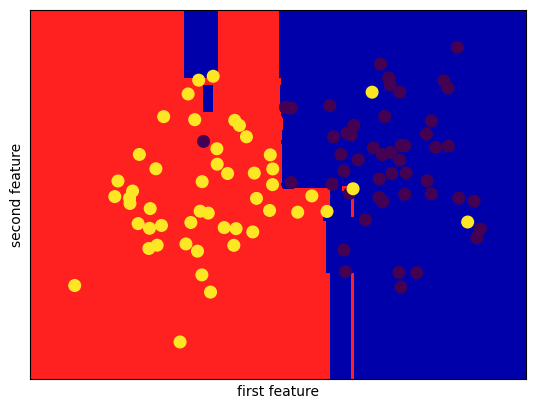


Рисунок 8 — Области принятия решений

При увеличении размера тестовой выборки результаты классификации будут только ухудшаться (пропорционально размеру тестовой выборки). Лучшие результаты получились при классификации методом k-ближних и наивный байесовским. Заполним сводную таблицу (таблица 1) для каждого метода и размера тестовой выборки 10%, 25% и 35%.

Таблица 1 — Сводная таблица

|  |  |  |
| --- | --- | --- |
| Метод | Размер тестовой выборки | Результат |
| k-ближних | 10 % | Confusion matrix:  [5 0]  [1 4]  Accuracy score: 0.9  ROC AUC  0.9 |
| Наивный байесовский | Confusion matrix:  [5 0]  [1 4]  Accuracy score: 0.9  ROC AUC  0.9 |
| Рандомный лес | Confusion matrix:  [5 0]  [1 4]  Accuracy score: 0.9  ROC AUC  0.9 |
| k-ближних | 25 % | Confusion matrix:  [13 0]  [ 2 10]  Accuracy score: 0.92  ROC AUC  0.9166666666666667 |
| Наивный байесовский | Confusion matrix:  [13 0]  [ 2 10]  Accuracy score: 0.92  ROC AUC  0.9166666666666667 |
| Рандомный лес | Confusion matrix:  [13 0]  [3 9]  Accuracy score: 0.88  ROC AUC  0.875 |
| k-ближних | 35 % | Confusion matrix:  [18 0]  [ 3 14]  Accuracy score: 0.9142857142857143  ROC AUC  0.9117647058823529 |
| Наивный байесовский | Confusion matrix:  [18 0]  [ 3 14]  Accuracy score: 0.9142857142857143  ROC AUC  0.9117647058823529 |
| Рандомный лес | Confusion matrix:  [18 0]  [ 3 14]  Accuracy score: 0.9142857142857143  ROC AUC  0.9117647058823529 |

Вывод

Мы получили практические навыки решения задачи бинарной классификации данных.

Приложение А (рекомендованное)

Исходный код

*<b>Бинарная классификация фактографических данных.</b>  
Цель работы: получить практические навыки решения задачи бинарной классификации  
данных в среде Jupiter Notebook, научиться загружать данные, обучать  
классификаторы и проводить классификацию, научиться оценивать точность  
полученных моделей.  
"""*# импортируем библиотеки  
import numpy as np  
import matplotlib.pyplot as plt  
  
from sklearn.datasets import make\_blobs  
from sklearn.metrics import confusion\_matrix  
from sklearn.metrics import classification\_report  
from sklearn.metrics import accuracy\_score  
from sklearn.model\_selection import train\_test\_split  
from sklearn.linear\_model import LogisticRegression  
from sklearn.neighbors import KNeighborsClassifier  
from sklearn.ensemble import RandomForestClassifier  
from sklearn.naive\_bayes import GaussianNB  
from sklearn.metrics import roc\_auc\_score  
from matplotlib.colors import ListedColormap, colorConverter, LinearSegmentedColormap  
  
# plot\_2d\_separator.py by amueller https://github.com/amueller/mglearn/blob/master/mglearn/plot\_2d\_separator.py  
  
cm\_cycle = ListedColormap(['#0000aa', '#ff5050', '#50ff50', '#9040a0', '#fff000'])  
cm3 = ListedColormap(['#0000aa', '#ff2020', '#50ff50'])  
cm2 = ListedColormap(['#0000aa', '#ff2020'])  
  
def plot\_2d\_classification(classifier, X, fill=False, ax=None, eps=None, alpha=1, cm=cm3):  
 # multiclass  
 if eps is None:  
 eps = X.std() / 2.  
  
 if ax is None:  
 ax = plt.gca()  
  
 x\_min, x\_max = X[:, 0].min() - eps, X[:, 0].max() + eps  
 y\_min, y\_max = X[:, 1].min() - eps, X[:, 1].max() + eps  
 xx = np.linspace(x\_min, x\_max, 1000)  
 yy = np.linspace(y\_min, y\_max, 1000)  
  
 X1, X2 = np.meshgrid(xx, yy)  
 X\_grid = np.c\_[X1.ravel(), X2.ravel()]  
 decision\_values = classifier.predict(X\_grid)  
 ax.imshow(decision\_values.reshape(X1.shape), extent=(x\_min, x\_max,  
 y\_min, y\_max),  
 aspect='auto', origin='lower', alpha=alpha, cmap=cm)  
 ax.set\_xlim(x\_min, x\_max)  
 ax.set\_ylim(y\_min, y\_max)  
 ax.set\_xticks(())  
 ax.set\_yticks(())  
  
  
def plot\_2d\_scores(classifier, X, ax=None, eps=None, alpha=1, cm="viridis",  
 function=None):  
 # binary with fill  
 if eps is None:  
 eps = X.std() / 2.  
  
 if ax is None:  
 ax = plt.gca()  
  
 x\_min, x\_max = X[:, 0].min() - eps, X[:, 0].max() + eps  
 y\_min, y\_max = X[:, 1].min() - eps, X[:, 1].max() + eps  
 xx = np.linspace(x\_min, x\_max, 100)  
 yy = np.linspace(y\_min, y\_max, 100)  
  
 X1, X2 = np.meshgrid(xx, yy)  
 X\_grid = np.c\_[X1.ravel(), X2.ravel()]  
 if function is None:  
 function = getattr(classifier, "decision\_function",  
 getattr(classifier, "predict\_proba"))  
 else:  
 function = getattr(classifier, function)  
 decision\_values = function(X\_grid)  
 if decision\_values.ndim > 1 and decision\_values.shape[1] > 1:  
 # predict\_proba  
 decision\_values = decision\_values[:, 1]  
 grr = ax.imshow(decision\_values.reshape(X1.shape),  
 extent=(x\_min, x\_max, y\_min, y\_max), aspect='auto',  
 origin='lower', alpha=alpha, cmap=cm)  
  
 ax.set\_xlim(x\_min, x\_max)  
 ax.set\_ylim(y\_min, y\_max)  
 ax.set\_xticks(())  
 ax.set\_yticks(())  
 return grr  
  
  
def plot\_2d\_separator(classifier, X, fill=False, ax=None, eps=None, alpha=1,  
 cm=cm2, linewidth=None, threshold=None,  
 linestyle="solid"):  
 # binary?  
 if eps is None:  
 eps = X.std() / 2.  
  
 if ax is None:  
 ax = plt.gca()  
  
 x\_min, x\_max = X[:, 0].min() - eps, X[:, 0].max() + eps  
 y\_min, y\_max = X[:, 1].min() - eps, X[:, 1].max() + eps  
 xx = np.linspace(x\_min, x\_max, 1000)  
 yy = np.linspace(y\_min, y\_max, 1000)  
  
 X1, X2 = np.meshgrid(xx, yy)  
 X\_grid = np.c\_[X1.ravel(), X2.ravel()]  
 try:  
 decision\_values = classifier.decision\_function(X\_grid)  
 levels = [0] if threshold is None else [threshold]  
 fill\_levels = [decision\_values.min()] + levels + [  
 decision\_values.max()]  
 except AttributeError:  
 # no decision\_function  
 decision\_values = classifier.predict\_proba(X\_grid)[:, 1]  
 levels = [.5] if threshold is None else [threshold]  
 fill\_levels = [0] + levels + [1]  
 if fill:  
 ax.contourf(X1, X2, decision\_values.reshape(X1.shape),  
 levels=fill\_levels, alpha=alpha, cmap=cm)  
 else:  
 ax.contour(X1, X2, decision\_values.reshape(X1.shape), levels=levels,  
 colors="black", alpha=alpha, linewidths=linewidth,  
 linestyles=linestyle, zorder=5)  
  
 ax.set\_xlim(x\_min, x\_max)  
 ax.set\_ylim(y\_min, y\_max)  
 ax.set\_xticks(())  
 ax.set\_yticks(())  
  
# генерируем выборку в соответсвии с вариантом  
  
centers = 2  
random\_state = 28  
cluster\_std = 4.5  
  
x, y = make\_blobs(centers = centers, random\_state = random\_state, cluster\_std = cluster\_std, shuffle = 1)  
  
# выводим первые 15 координат и меток  
  
print ("Координаты точек: ")  
print (x[:15])  
print ("Метки класса: ")  
print (y[:15])  
  
# выводим сгенерированные данные по меткам  
  
plt.scatter (x[:, 0], x[:, 1], c = y)  
plt.show()  
  
# Метод к-ближайших соседей  
  
def KNeighbors(x, y, test\_size = 0.25, n\_neighbors = 5):  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = test\_size, random\_state = 1)  
 knn = KNeighborsClassifier(n\_neighbors = n\_neighbors, metric = 'euclidean')  
 knn.fit(x\_train, y\_train)  
 prediction = knn.predict(x\_test)  
  
 print ('Prediction and test:')  
 print ('Prediction: \t', prediction)  
 print ('Test: \t\t', y\_test)  
 print("\n\n")  
 print ('Confusion matrix: ')  
 print (confusion\_matrix(y\_test, prediction)[0])  
 print (confusion\_matrix(y\_test, prediction)[1])  
 print("\n\n")  
 print ('Accuracy score: ', accuracy\_score(prediction, y\_test))  
 print("\n\n")  
 print('Classification Report\n', classification\_report(y\_test, prediction))  
 print("\n\n")  
 print('ROC AUC')  
 print(roc\_auc\_score(y\_test, prediction))  
 print("\n\n")  
  
 # обучающая и тестовая выборки  
 plt.title('Division into training (Blue) and test (Red) samples')  
 plt.scatter (x\_train[:, 0], x\_train[:, 1], color = 'blue')  
 plt.scatter (x\_test[:, 0], x\_test[:, 1], color = 'red')  
 plt.show()  
  
 plt.xlabel("first feature")  
 plt.ylabel("second feature")  
 plot\_2d\_separator(knn, x, fill=True)  
 plt.scatter(x[:, 0], x[:, 1], c=y, s=70)  
  
# Наивный байесовский метод  
  
def naiveBayes(x, y, test\_size = 0.25):  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = test\_size, random\_state = 1)  
 gnb = GaussianNB()  
 gnb.fit(x\_train, y\_train)  
 prediction = gnb.predict(x\_test)  
  
 print ('Prediction and test:')  
 print ('Prediction: \t', prediction)  
 print ('Test: \t\t', y\_test)  
 print("\n\n")  
 print ('Confusion matrix: ')  
 print (confusion\_matrix(y\_test, prediction)[0])  
 print (confusion\_matrix(y\_test, prediction)[1])  
 print("\n\n")  
 print ('Accuracy score: ', accuracy\_score(prediction, y\_test))  
 print("\n\n")  
 print('Classification Report\n', classification\_report(y\_test, prediction))  
 print("\n\n")  
 print('ROC AUC')  
 print(roc\_auc\_score(y\_test, prediction))  
 print("\n\n")  
  
 # обучающая и тестовая выборки  
 plt.title('Division into training (Blue) and test (Red) samples')  
 plt.scatter (x\_train[:, 0], x\_train[:, 1], color = 'blue')  
 plt.scatter (x\_test[:, 0], x\_test[:, 1], color = 'red')  
 plt.show()  
  
 plt.xlabel("first feature")  
 plt.ylabel("second feature")  
 plot\_2d\_separator(gnb, x, fill=True)  
 plt.scatter(x[:, 0], x[:, 1], c=y, s=70)  
  
# Случайный лес  
  
def randomForest(x, y, test\_size = 0.25, n\_estimators = 100):  
 x\_train, x\_test, y\_train, y\_test = train\_test\_split(x, y, test\_size = test\_size, random\_state = 1)  
 rfc = RandomForestClassifier(n\_estimators = n\_estimators)  
 rfc.fit(x\_train, y\_train)  
 prediction = rfc.predict(x\_test)  
  
 print ('Prediction and test:')  
 print ('Prediction: \t', prediction)  
 print ('Test: \t\t', y\_test)  
 print("\n\n")  
 print ('Confusion matrix: ')  
 print (confusion\_matrix(y\_test, prediction)[0])  
 print (confusion\_matrix(y\_test, prediction)[1])  
 print("\n\n")  
 print ('Accuracy score: ', accuracy\_score(prediction, y\_test))  
 print("\n\n")  
 print('Classification Report\n', classification\_report(y\_test, prediction))  
 print("\n\n")  
 print('ROC AUC')  
 print(roc\_auc\_score(y\_test, prediction))  
 print("\n\n")  
  
 # обучающая и тестовая выборки  
 plt.title('Division into training (Blue) and test (Red) samples')  
 plt.scatter (x\_train[:, 0], x\_train[:, 1], color = 'blue')  
 plt.scatter (x\_test[:, 0], x\_test[:, 1], color = 'red')  
 plt.show()  
  
 plt.xlabel("first feature")  
 plt.ylabel("second feature")  
 plot\_2d\_separator(rfc, x, fill=True)  
 plt.scatter(x[:, 0], x[:, 1], c=y, s=70)  
  
"""Метод к-ближайших соседей (1, 3, 5, 9)"""  
  
KNeighbors(x, y, n\_neighbors = 1)  
  
KNeighbors(x, y, n\_neighbors = 3)  
  
KNeighbors(x, y, n\_neighbors = 5)  
  
KNeighbors(x, y, n\_neighbors = 9)  
  
"""Наивный байесовский метод"""  
  
naiveBayes(x, y)  
  
"""Случайный лес (5, 10, 15, 20, 50)"""  
  
randomForest(x, y, n\_estimators = 5)  
  
randomForest(x, y, n\_estimators = 10)  
  
randomForest(x, y, n\_estimators = 15)  
  
randomForest(x, y, n\_estimators = 20)  
  
randomForest(x, y, n\_estimators = 50)  
  
"""Тестовая часть — 10% выборки"""  
  
KNeighbors(x, y, test\_size = 0.1)  
  
naiveBayes(x, y, test\_size = 0.1)  
  
randomForest(x, y, test\_size = 0.1)  
  
"""Тестовая часть — 35% выборки"""  
  
KNeighbors(x, y, test\_size = 0.35)  
  
naiveBayes(x, y, test\_size = 0.35)  
  
randomForest(x, y, test\_size = 0.35)  
  
randomForest(x, y, test\_size = 0.35, n\_estimators = 10)