1. Постановка задачи

На языке Python программно реализовать два метрических алгоритма классификации: Naive Bayes и K Nearest Neighbours
Сравнить работу реализованных алгоритмов с библиотечными из scikit-learn
Для тренировки, теста и валидации использовать один из предложенных датасетов (либо найти самостоятельно и внести в таблицу)
Сформировать краткий отчет (постановка задачи, реализация, эксперимент с данными, полученные характеристики, вывод

2. Исходные данные

Датасет: http://archive.ics.uci.edu/ml/datasets/Statlog+%28Heart%29
Предметная область: медицина
Задача: определить, присутствует ли сердечная болезнь или нет
Количество записей: 270
Количество атрибутов: 13
Атрибуты:

```
      -- 1. age
      -- 2. sex
      -- 3. chest pain type  (4 values)
      -- 4. resting blood pressure
      -- 5. serum cholestoral in mg/dl
      -- 6. fasting blood sugar > 120 mg/dl
      -- 7. resting electrocardiographic results  (values 0,1,2)
      -- 8. maximum heart rate achieved
      -- 9. exercise induced angina
      -- 10. oldpeak = ST depression induced by exercise relative to rest
      -- 11. the slope of the peak exercise ST segment
      -- 12. number of major vessels (0-3) colored by flourosopy
      -- 13.  thal: 3 = normal; 6 = fixed defect; 7 = reversable defect
```

Классы:

```
      -- 14. Absence (1) or presence (2) of heart disease
```

3. Ход работы

Реализация алгоритма Naive Bayes.

```
from sklearn.naive_bayes import GaussianNB
import pandas as pd
import math
import numpy as np
import operator
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from collections import Counter

# разделение датасета на тестовую и обучающую выборку
def split_dataset(test_size):
    dataset = pd.read_csv('heart.dat', header=None).values
    attr = dataset[:, 0:-1] # атрибуты
    heart_class =  (dataset[:, -1]).astype(np.int64, copy=False) # классы
    data_train, data_test, class_train, class_test =
train_test_split(attr, heart_class, test_size=test_size, random_state=55)
    return data_train, class_train, data_test, class_test
##############################################data#######################
##

##############################################nbayesfunctions############
##########

#разделение данных по классам
```

```
def get_subsequences_by_classes(data_train,class_train):
    d={}
    class_train_set=set(class_train)
    for i in class_train_set:
        d[i]=[]
    for i in range(len(data_train)):
        d[class_train[i]].append(data_train[i])
    return d

#получение среднего значения
def get_y(arr):
    return sum(arr)/len(arr)

#получение дисперсии
def get_disp(arr):
    res=0.0
    y=get_y(arr)
    for i in arr:
        res+=((i-y)**2)
    return ((res/float(len(arr)-1.0))**0.5)


#обучение классификатора
def train_classifier(data_train,class_train):
    D=get_subsequences_by_classes(data_train,class_train)
    results={}
    for class_name,class_elements in D.items():
        results[class_name]=[(get_y(attribute),get_disp(attribute)) for
attribute in zip(*class_elements)]
    return results

# вычисление f(xj|y,disp)
def f(x, y, disp):
    if disp == 0.0:
        disp += 0.000001
    return (1. / (math.sqrt(2. * math.pi) * disp)) * math.exp(-
(math.pow(x - y, 2.) / (2. * math.pow(disp, 2.))))

# вычисление P(ci)
def p(summaries, instance_attr):
    probabilities = {}
    for class_name, class_summaries in summaries.items():
        probabilities[class_name] = 1.0
        for i in range(len(class_summaries)):
            y, disp = class_summaries[i]
            probabilities[class_name] *= f(instance_attr[i], y, disp)
    return probabilities

# тест-е одного объекта
def test_one(train_results,dt):
    probabilities = p(train_results, dt)
    return max(probabilities.items(),key=operator.itemgetter(1))[0]

# тест-е классификатора
def test_classifier(train_results,data_test,class_test):
    score=0.0
    predicts=[(test_one(train_results,dt)) for dt in data_test]
```

```
    score =sum( [i == j for i, j in zip(predicts, class_test)])
    return score/float(len(predicts))



##################################################nbayesfuncntion##########
##########



data_train, class_train,data_test, class_test = split_dataset(0.3)
gnb = GaussianNB()
gnb.fit(data_train, class_train)
print('Naive Bayes library algo', 'Result: ', gnb.score(data_test,
class_test))

cl_training_results=train_classifier(data_train,class_train)
cl_testing_results=test_classifier(cl_training_results,data_test,class_te
st)
print('Naive Bayes algo', 'Result: ', cl_testing_results)
```

Реализация алгоритма K Nearest Neighbours

```
from sklearn.naive_bayes import GaussianNB
import pandas as pd
import math
import numpy as np
import operator
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from collections import Counter

# разделение датасета на тестовую и обучающую выборку
def split_dataset(test_size):
    dataset = pd.read_csv('heart.dat', header=None).values
    attr = dataset[:, 0:-1] # атрибуты
    heart_class =  (dataset[:, -1]).astype(np.int64, copy=False) # классы
    data_train, data_test, class_train, class_test =
train_test_split(attr, heart_class, test_size=test_size, random_state=55)
    return data_train, class_train, data_test, class_test
##################################################data#####################
##
##################################################KNearestNeighborClassifier
##########

#определение расстояния Евклида
def dist(instance1, instance2):
    squares = [(i - j) ** 2.0 for i, j in zip(instance1, instance2)]
    return ((sum(squares))**0.5)

# определение самого распространенного класса среди соседей
def get_response(instance, data_train, class_train, k):
    distances = []
    for i in data_train:
        distances.append(dist(instance, i))
    distances = tuple(zip(distances, class_train))
    neigbours=sorted(distances, key=operator.itemgetter(0))[:k]
    return Counter(neigbours).most_common()[0][0][1]
```

```
# классификация тестовой выборки
def p2(data_train, class_train, data_test, k):
    predictions=[(get_response(i, data_train, class_train, k)) for i in
data_test]
    return predictions

# тест-е классификатора
def test_classifier2(data_train, class_train, data_test, class_test, k):
    predictions = p2(data_train, class_train, data_test, k)
    y = [i == j for i, j in zip(class_test, predictions)]
    return sum(y) / len(y)

##############################################KNearestNeighborClassifier
##########
neighbors_count=13
data_train, class_train,data_test, class_test = split_dataset(0.3)
knc = KNeighborsClassifier(n_neighbors=neighbors_count)
knc.fit(data_train, class_train)
print('KNeighborsClassifier library algo', 'Result: ',
knc.score(data_test, class_test))
cl2_testing_results=test_classifier2(data_train, class_train, data_test,
class_test, neighbors_count)
print('KNeighborsClassifier algo', 'Result: ', cl2_testing_results)
```

Результаты:

```
Naive Bayes library algo Result:  0.83950617284
Naive Bayes algo Result:  0.83950617284
KNeighborsClassifier library algo Result:  0.679012345679
KNeighborsClassifier algo Result:  0.666666666667
```

Из результатов видно, что результат разработанного алгоритма Naive Bayes совпал с результатом библиотечной функции из пакета sklearn.naive_bayes. Результат разработанного алгоритма K-neighbors немного отклонился от библиотечной функции из пакета from sklearn.neighbors, но незначительно.