## 1. Постановка задачи

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

## 2. Исходные данные

Датасет: https://www.openml.org/d/44

Предметная область: спам в электронной рассылке

Задача: определить, является ли электронное письмо спамом

Количество записей: 4601

Количество атрибутов: 57

Атрибуты:

1. Частота использования строки «make» (вещественный тип, [0,100])
2. Частота использования строки «address» (вещественный тип, [0,100])
3. Частота использования строки «all» (вещественный тип, [0,100])
4. Частота использования строки «3d» (вещественный тип, [0,100])
5. Частота использования строки «our» (вещественный тип, [0,100])
6. Частота использования строки «over» (вещественный тип, [0,100])
7. Частота использования строки «remove» (вещественный тип, [0,100])
8. Частота использования строки «internet» (вещественный тип, [0,100])
9. Частота использования строки «order» (вещественный тип, [0,100])
10. Частота использования строки «mail» (вещественный тип, [0,100])
11. Частота использования строки «receive» (вещественный тип, [0,100])
12. Частота использования строки «will» (вещественный тип, [0,100])
13. Частота использования строки «people» (вещественный тип, [0,100])

14. Частота использования строки «report» (вещественный тип, [0,100])
15. Частота использования строки «addresses» (вещественный тип, [0,100])
16. Частота использования строки «free» (вещественный тип, [0,100])
17. Частота использования строки «business» (вещественный тип, [0,100])
18. Частота использования строки «email» (вещественный тип, [0,100])
19. Частота использования строки «you» (вещественный тип, [0,100])
20. Частота использования строки «credit» (вещественный тип, [0,100])
21. Частота использования строки «your» (вещественный тип, [0,100])
22. Частота использования строки «font» (вещественный тип, [0,100])
23. Частота использования строки «000» (вещественный тип, [0,100])
24. Частота использования строки «money» (вещественный тип, [0,100])
25. Частота использования строки «hp» (вещественный тип, [0,100])
26. Частота использования строки «hpl» (вещественный тип, [0,100])
27. Частота использования строки «george» (вещественный тип, [0,100])
28. Частота использования строки «650» (вещественный тип, [0,100])
29. Частота использования строки «lab» (вещественный тип, [0,100])
30. Частота использования строки «labs» (вещественный тип, [0,100])
31. Частота использования строки «telnet» (вещественный тип, [0,100])
32. Частота использования строки «857» (вещественный тип, [0,100])
33. Частота использования строки «data» (вещественный тип, [0,100])
34. Частота использования строки «415» (вещественный тип, [0,100])
35. Частота использования строки «85» (вещественный тип, [0,100])
36. Частота использования строки «technology» (вещественный тип, [0,100])
37. Частота использования строки «1999» (вещественный тип, [0,100])
38. Частота использования строки «parts» (вещественный тип, [0,100])
39. Частота использования строки «pm» (вещественный тип, [0,100])
40. Частота использования строки «direct» (вещественный тип, [0,100])
41. Частота использования строки «cs» (вещественный тип, [0,100])
42. Частота использования строки «meeting» (вещественный тип, [0,100])

43. Частота использования строки «original» (вещественный тип, [0,100])

44. Частота использования строки «project» (вещественный тип, [0,100])

45. Частота использования строки «re» (вещественный тип, [0,100])

46. Частота использования строки «edu» (вещественный тип, [0,100])

47. Частота использования строки «table» (вещественный тип, [0,100])

48. Частота использования строки «conference» (вещественный тип, [0,100])

49. Частота использования символа “<” (вещественный тип, [0,100])

50. Частота использования символа “(” (вещественный тип, [0,100])

51. Частота использования символа “[” (вещественный тип, [0,100])

52. Частота использования символа “!” (вещественный тип, [0,100])

53. Частота использования символа “$” (вещественный тип, [0,100])

54. Частота использования символа “#” (вещественный тип, [0,100])

55. Средняя длина непрерывной последовательности заглавных букв (вещественный тип, [1, …])

56. Самая длинная непрерывная последовательность заглавных букв (целый тип, [1, …])

57. Сумма длин всех непрерывных последовательностей заглавных букв (целый тип, [1, …])

58. Класс: 0 – не спам, 1 – спам.

## 3. Ход работы

Реализация наивного байесовского классификатора (naive_bayes.py):

```
import math
import numpy as np
import pandas as pd

from data import accuracy
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split

# Определение математического ожидания и дисперсии каждого атрибута
def mu_sigma(X):
    m = [sum(x) / len(x) for x in X]
    d = [sum(map(lambda y: y**2, x)) / len(x)
         - pow(sum(x) / len(x), 2) for x in X]
    return m, d

# Функция правдоподобия
def liklihood(x, m, d):
    if d == 0:
        d = 0.000001
```

```
    return 1./(d * math.sqrt(2 * math.pi)) * math.exp(-pow(x - m, 2)/(2 * d**2))

# Получение мат. ожидания и дисперсии для каждого класса
def mu_sigma_by_class(c_dict):
    print('- определение мат. ожидания и дисперсии для классов...')
    ms_dict = {}
    for class_name, attributes in c_dict.items():
        ms_dict[class_name] = mu_sigma(attributes)
    return ms_dict

# Формирование словаря классов
def class_dictionary(attributes, classes):
    print('- формирование словаря классов...')
    temp = {}
    c_dict = {}
    for i in range(len(attributes)):
        temp.setdefault(classes[i], []).append(attributes[i])
    for key, value in temp.items():
        c_dict[key] = np.asarray(value).transpose()
    return c_dict

# Получение априорных вероятностей для всех классов
def class_apriory(c_dict):
    print('- получение априорных вероятностей для классов...')
    train_size = float(sum([len(x[0]) for x in list(c_dict.values())]))
    return [len(x[0]) / train_size for x in list(c_dict.values())]

# Определение наиболее подходящего класса для вектора атрибутов x
def get_class_number(x, ms_dict):
    result = 0
    pre_p = 0
    for key, summary in ms_dict.items():
        m, d = summary
        p = 1
        for i in range(len(x)):
            p *= liklihood(x[i], m[i], d[i])
        if p > pre_p:
            result = key
            pre_p = p
    return result

# Определение вероятности принадлежности объекта к каждому классу
def nb_classification(x_test, ms_dict):
    print('- определение наиболее подходящих классов...')
    y_result = []
    for x in x_test:
        y_result.append(get_class_number(x, ms_dict))
    return np.array(y_result)

# Основная функция Наивного Байесовского классификатора
def naive_bayes(x_train, x_test, y_train, y_test):
    # Обучение
    print('Обучение...')
    c_dict = class_dictionary(x_train, y_train)
    p_c = class_apriory(c_dict)
    ms_dict = mu_sigma_by_class(c_dict)

    # Тестирование
    print('Тестирование...')
    y_result = nb_classification(x_test, ms_dict)
    print('Точность разработанного ПО: {:.3f}'.format(accuracy(y_result, y_test)))

    # Сравнение со стандартными инструментами
    clf = GaussianNB()
    clf.fit(x_train, y_train)
    print('Точность стандартных средств: {:.3f}'.format(clf.score(x_test, y_test)))
    return
```

Реализация классификатора K ближайших соседей (nearest_neighbors.py):

```
import numpy as np

from math import sqrt
from data import accuracy
from collections import Counter
```

```
from sklearn.neighbors import KNeighborsClassifier

# Расчёт расстояния между двумя объектами
def distance(x1, x2):
    return sqrt(sum([(i - j)*(i - j) for i, j in zip(x1, x2)]))

# Расчёт расстояний для каждого объекта
def get_neighbors(x, x_train, y_train, k):
    return sorted(tuple(zip([distance(x, i) for i in x_train], y_train)))[:k]

# Получение списка соседей для каждого объекта
def get_neighbors_list(x_train, y_train, x_test, k):
    print('Получение списка соседей (время выполнения ~ 3 мин)...')
    return [get_neighbors(x, x_train, y_train, k) for x in x_test]

# Получение списка самых частых классов среди соседей
def nn_classification(neighbors):
    print('Классификация...')
    return [Counter(neighbor).most_common()[0][0][1] for neighbor in neighbors]

# Основной алгоритм метода ближайших соседей
def nearest_neighbors(x_train, x_test, y_train, y_test, k):
    # Обучение и тестирование
    neighbors = get_neighbors_list(x_train, y_train, x_test, k)
    y_result = nn_classification(neighbors)
    print('Точность разработанного ПО: {:.3f}'.format(accuracy(y_result, y_test)))

    # Сравнение со стандартными инструментами
    clf = KNeighborsClassifier(n_neighbors=k)
    clf.fit(x_train, y_train)
    print('Точность стандартных средств: {:.3f}'.format(clf.score(x_test, y_test)))
    return
```

## Получение данных (data.py):

```
import numpy as np
import pandas as pd

from sklearn.model_selection import train_test_split

# Загрузка данных
def load_data(filename):
    print('Загрузка данных из файла...')
    return pd.read_csv(filename, header = None).values

# Разделение датасета
def split_data(data):
    print('Разделение набора данных...')
    attributes = data[:, :-1]
    classes = np.ravel(data[:, -1:].astype(np.int64, copy=False))
    return train_test_split(
        attributes, classes, test_size=0.3, random_state=42)

# Получение данных
def get_data():
    return split_data(load_data('data/spambase.data'))

# Определение точности вычислений
def accuracy(y_result, y_test):
    c = 0
    for i in range(len(y_test)):
        if(y_result[i] == y_test[i]):
            c += 1
    return c / float(len(y_test))
```

## Главная функция (source.py):

```
# Машинное обучение.
# Лаб. 1. Метрические алгоритмы классификации

from data import get_data
from naive_bayes import naive_bayes
from nearest_neighbors import nearest_neighbors

def main():
```

```
    print('Сбор данных:')
    d = get_data()
    print('-------------')
    print('Наивный Байесовский классификатор:')
    naive_bayes(*d)
    print('-------------')
    print('Метод ближайших соседей:')
    nearest_neighbors(*d, 5)

main()
```

Тестовый запуск:

```
Сбор данных:
Загрузка данных из файла...
Разделение набора данных...
-------------
Наивный Байесовский классификатор:
Обучение...
- формирование словаря классов...
- получение априорных вероятностей для классов...
- определение мат. ожидания и дисперсии для классов...
Тестирование...
- определение наиболее подходящих классов...
Точность разработанного ПО: 0.857
Точность стандартных средств: 0.825
-------------
Метод ближайших соседей:
Получение списка соседей (время выполнения ~ 3 мин)...
Классификация...
Точность разработанного ПО: 0.789
Точность стандартных средств: 0.782
```

**Вывод**

По результатам тестового запуска, наивный байесовский классификатор (точность: 0,857 и 0,825 для разработанного ПО и библиотеки sklearn соответственно) продемонстрировал более высокую точность, чем метод k ближайших соседей (0,789 и 0,782 для разработанного ПО и библиотеки sklearn соответственно). В обоих случаях точность разработанных алгоритмов оказалась выше точности библиотечных. При этом собственный алгоритм для метода k ближайших соседей работает во много раз медленнее соответствующего библиотечного и любого из алгоритмов наивного байесовского классификатора.