

1. Постановка задачи

1. На языке Python программно реализовать два метрических алгоритма классификации: Naive Bayes и K Nearest Neighbours
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://archive.ics.uci.edu/ml/datasets/Wine>

Предметная область: Состав вина разного географического происхождения

Задача: определить, в какой из 3 областей произведено вино

Количество записей: 178

Количество атрибутов: 13

Атрибуты:

- 1) Алкоголь
- 2) Малиновая кислота
- 3) Зола
- 4) Алкалия зола
- 5) Магний
- 6) Всего фенолов
- 7) Флаванойды
- 8) Нефлаванойдные фенолы
- 9) Проантоцианы
- 10) Интенсивность цвета
- 11) Оттенок
- 12) OD280 / OD315 разведенных вин
- 13) пролин

3. Ход работы

1. Реализация алгоритма Naive Bayes.

```
import math
import numpy as np
import pandas as pd
import sys
from sklearn.naive_bayes import GaussianNB
from sklearn.model_selection import train_test_split
```

```

# загрузка датасета
def load_data(filename):
    return pd.read_csv(filename, header=None).values

# разделение датасета на тестовую и обучающую выборку
def split_dataset(test_size):
    dataset = load_data('data.csv')
    occ_attr = dataset[:, 1:]
    occ_class = dataset[:, 0]
    occ_class = occ_class.astype(np.float)
    occ_attr = occ_attr.astype(np.float)
    data_train, data_test, class_train, class_test = train_test_split(occ_attr,
occ_class, test_size=test_size, random_state=55)
    return data_train, class_train, data_test, class_test

# Разделяет обучающую выборку по классам таким образом, чтобы можно было получить все
элементы,
# принадлежащие определенному классу.
def separate_by_class(data_train, class_train):
    classes_dict = {}
    for i in range(len(data_train)):
        classes_dict.setdefault(class_train[i], []).append(data_train[i])
    return classes_dict

# инструменты для обобщения данных
def mean(numbers): # Среднее значение
    numbers = [ float(x) for x in numbers ]
    return sum(numbers) / float(len(numbers))

def stand_dev(numbers): # вычисление дисперсии
    numbers = [ float(x) for x in numbers ]
    var = sum([pow(x - mean(numbers), 2) for x in numbers]) / float(len(numbers) - 1)
    return math.sqrt(var)

def summarize(data_train): # обобщение данных
    # Среднее значение и среднеквадратичное отклонение для каждого атрибута
    summaries = [(mean(att_numbers), stand_dev(att_numbers)) for att_numbers in
zip(*data_train)]
    return summaries

# Обучение классификатора
def summarize_by_class(data_train, class_train):
    # Разделяет обучающую выборку по классам таким образом, чтобы можно было получить
все элементы,
# принадлежащие определенному классу.
    classes_dict = separate_by_class(data_train, class_train)
    summaries = {}
    for class_name, instances in classes_dict.items():
        summaries[class_name] = summarize(instances)
    return summaries

# вычисление апостериорной вероятности принадлежности объекта к определенному классу
def calc_probability(x, mean, stdev):
    if stdev == 0:

```

```

        stdev += 0.000001 # добавляем эpsilon, если дисперсия равна 0
    exponent = math.exp(-(math.pow(x - mean, 2) / (2 * math.pow(stdev, 2))))
    return (1 / (math.sqrt(2 * math.pi) * stdev)) * exponent

# вычисление вероятности принадлежности объекта к каждому из классов
def calc_class_probabilities(summaries, instance_attr):
    probabilities = {}
    for class_name, class_summaries in summaries.items():
        probabilities[class_name] = 1.0
        for i in range(len(class_summaries)):
            mean, stdev = class_summaries[i]
            x = float(instance_attr[i])
            probabilities[class_name] *= calc_probability(x, mean, stdev)
    return probabilities

# классификация одного объекта
def predict_one(summaries, instance_attr):
    # вычисление вероятности принадлежности объекта к каждому из классов
    probabilities = calc_class_probabilities(summaries, instance_attr)
    best_class, max_prob = None, -1
    for class_name, probability in probabilities.items():
        if best_class is None or probability > max_prob:
            max_prob = probability
            best_class = class_name
    return best_class

# классификация тестовой выборки
def predict(summaries, data_test):
    predictions = []
    for i in range(len(data_test)):
        result = predict_one(summaries, data_test[i])
        predictions.append(result)
    return predictions

# сравнение результатов классификации с реальными, вычисление точности классификации
def calc_accuracy(summaries, data_test, class_test):
    correct_answ = 0
    # классификация тестовой выборки
    predictions = predict(summaries, data_test)
    for i in range(len(data_test)):
        if class_test[i] == predictions[i]:
            correct_answ += 1
    return correct_answ / float(len(data_test))

def main():
    data_train, class_train, data_test, class_test = split_dataset(0.33)
    summaries = summarize_by_class(data_train, class_train)
    accuracy = calc_accuracy(summaries, data_test, class_test)
    print('myNBClass ', 'Accuracy: ', accuracy)

    clf = GaussianNB()
    clf.fit(data_train, class_train)
    print('sklNBClass ', 'Accuracy: ', clf.score(data_test, class_test))
main()

```

Results:

```
('myNBClass ', 'Accuracy: ', 0.9491525423728814)
('sklNBClass ', 'Accuracy: ', 0.94915254237288138)
```

2. Реализация алгоритма K Nearest Neighbours

```
from __future__ import division
import pandas as pd
import numpy as np
import operator
from sklearn.model_selection import train_test_split
from math import sqrt
from collections import Counter
from sklearn.neighbors import KNeighborsClassifier

# загрузка датасета
def load_data(filename):
    return pd.read_csv(filename, header=None).values

# разделение датасета на тестовую и обучающую выборку
def split_dataset(test_size):
    dataset = load_data('data.csv')
    occ_attr = dataset[:, 1:]
    #print(occ_attr)
    occ_class = dataset[:, 0]
    occ_class = occ_class.astype(np.float)
    occ_attr = occ_attr.astype(np.float)
    data_train, data_test, class_train, class_test = train_test_split(occ_attr,
occ_class, test_size=test_size)
    return data_train, class_train, data_test, class_test

# евклидово расстояние от объекта №1 до объекта №2
def euclidean_distance(instance1, instance2):
    squares = [(i - j) ** 2 for i, j in zip(instance1, instance2)]
    return sqrt(sum(squares))

# расчет расстояний до всех объектов в датасете
def get_neighbours(instance, data_train, class_train, k):
    distances = []
    for i in data_train:
        distances.append(euclidean_distance(instance, i))
    distances = tuple(zip(distances, class_train))
    # сортировка расстояний по возрастанию
    # k ближайших соседей
    return sorted(distances, key=operator.itemgetter(0))[:k]

# определение самого распространенного класса среди соседей
def get_response(neighbours):
    return Counter(neighbours).most_common()[0][0][1]

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

```

        neighbours = get_neighbours(i, data_train, class_train, k)
        response = get_response(neighbours)
        predictions.append(response)
    return predictions

# измерение точности
def get_accuracy(data_train, class_train, data_test, class_test, k):
    predictions = get_predictions(data_train, class_train, data_test, k)
    mean = [i == j for i, j in zip(class_test, predictions)]
    return sum(mean) / len(mean)

def main():
    data_train, class_train, data_test, class_test = split_dataset(0.35)
    print('myKNClass', 'Accuracy: ', get_accuracy(data_train, class_train, data_test,
class_test, 15))

    clf = KNeighborsClassifier(n_neighbors=15)
    clf.fit(data_train, class_train)
    print('sklKNClass', 'Accuracy: ', clf.score(data_test, class_test))

```

main()

Results:

Naive Bayes

```

('myNBClass ', 'Accuracy: ', 0.9491525423728814)
('sklNBClass ', 'Accuracy: ', 0.94915254237288138)

```

K Nearest Neighbours

```

('myKNClass', 'Accuracy: ', 0.66666666666666663)
('sklKNClass', 'Accuracy: ', 0.73015873015873012)

```

В ходе проделанной работы были получены приведенные выше результаты. Результат разработанного алгоритма Naive Bayes совпал с библиотечным.

А точность разработанного алгоритма K Nearest Neighbours несколько ниже предоставляемого библиотекой.