Таблица 1.

Сравнение результатов аппаратной реализации различных поточных шифров.

Алгоритм	Nk	Тактов на инициализацию	GE	Cl/byte
Enocoro-128 v2	128	4869	4100	46.3
Grain v1	80	321	1294	31.2
Mickey-128 v2	128	56592	5000	1231.4
Trivium	80	1333	3488	8.0
AES-CTR	128	470	4400	17.8

Обозначения: Nk - длина ключа (в битах), GE - условные логические элементы (Gate Equivalent), Cl/byte - число тактов работы алгоритма на байт зашифрованной информации – мера скорости работы алгоритма, пропускная способность.

Рассмотренные криптографические алгоритмы и их реализация построены и применяются для обеспечения информационной безопасности соответствующих систем с учетом их взаимодействия с интернетом и сетевыми технологиями [4].

Проанализировав результаты аппаратной реализации различных алгоритмов шифрования, становятся видны преимущества алгоритма Grain v1. Он занимает наименьшее количество логических элементов среди рассмотренных шифров. Так же ему требуется меньшее количество тактов на инициализацию, и он имеет достаточно хороший показатель скорости работы, что позволяет использовать его для различных устройств "Интернета вещей".

1. Жукова, А.Е. Легковесная криптография. Части 1 и 2/ А.Е. Жукова. [Электронный ресурс]. – URL: https://cyberrus.com/wp-content/uploads/2015/05/vkb_10_01.pdf
2. Урбанович, П. П. Защита информации методами криптографии, стеганографии и обфускации: учеб.-метод. пособие для студ./ П.П. Урбанович. - Минск: БЕТУ, 2016. - 220 с.
3. LW-криптография: шифры для RFID-систем. [Электронный ресурс]. – URL: <https://habr.com/ru/post/119700/>
4. Урбанович, П. П. Компьютерные сети: учебное пособие для студентов высших учебных заведений по техническим специальностям / П. П. Урбанович, Д. М. Романенко, Е. В. Кабак. - Минск: БГТУ, 2011. - 399 с.

Шавтикова Л.М.

Реализация алгоритма kNN на языке программирования Python

Северо-Кавказская государственная академия
(Россия, Черкесск)

doi: 10.18411/lj-09-2020-05

idsp: ljjournal-09-2020-05

Аннотация

В данной статье рассматривается реализация одного из простых алгоритмов машинного обучения - k-nearest neighbors (кближайших соседей) для задачи классификации на Python с использованием библиотек. Важной частью машинного обучения является предсказание значений на новых данных.

Ключевые слова: машинное обучение, алгоритм, выборка, диаграмма, графика, визуализация.

Abstract

This article discusses the implementation of one of the simple machine learning algorithms - k-nearest neighbors for the classification problem in Python using libraries. An important part of machine learning is predicting values on new data.

Keywords: machine learning, algorithm, selection, diagram, graph, visualization.

Важной частью машинного обучения является предсказание значений на новых данных. Нас интересует создание такого алгоритма, который на тестовой выборке (новых данных) выдавал бы наилучшие результаты. Насколько они хороши - определяют метрики качества. Мы разделим все данные на обучающую (обычно берут 80-85% от всех данных) и тестовую выборки.

Стадия подготовки данных:

1. Формируем выборку и строим графическое представление.
2. Делим выборку на две группы в заданном отношении (80% и 20%): 80% - обучающаяся выборка, 20% - тестовая выборка.
3. Строим графическое представление полученных групп выборок.

```
import numpy as np
import pandas as pd
from matplotlib import pyplot as plt
from matplotlib.pyplot import figure
import seaborn as sns
def showData(X, y):
    """Графическое представление выборки
    X - параметры выборки
    y - параметры классов"""
    # Формируем DataFrame из словаря, составленного из списка
    # x, y, Cod_class - названия столбцов
    df = pd.DataFrame(dict(x=X[:,0], y=X[:,1], Cod_class=y))
    # print(df) # Тестовый вывод
    # Определяем словарь для цветов классов: Признак класса --> Цвет
    colors = {0: 'red',
              1: 'b',
              2: '#00CC00'} # способы задания цвета
    # Определяем параметры графической области (графического изображения fig и
    # оси - ax)
    # В нашем случае параметры определяются автоматом
    fig, ax = plt.subplots()
    grouped = df.groupby('Cod_class') # Группировка по столбцу 'Cod_class'
    for key, group in grouped:      # выделяем признаки класса (код класса и коорди
    на ты)
        # рисуем
        group.plot(ax=ax,
                   kind = 'scatter', # Способ формирования изображения
                                   # 'scatter' - для построения графика разбросов
                                   # Можно изображать бары, гистограммы, боксы, ...
                   x='x', y='y',
                   label=key,
                   color=colors[key])
    plt.show() # Показываем результат
# Создаем выборку из 10000 элементов.
# Элементы выборки разделены на 3 класса с 2-я признаками
from sklearn.datasets import make_blobs
# Создание изотропных гауссовских капель для кластеризации
N_sampl = 10000 # размер выборки, равномерно распределённой
# между кластерами
N_class = 3 # число центров выборок - число классов объектов
X, y = make_blobs(n_samples=N_sampl,
                  centers=N_class,
```

```

n_features=2,    # количество характеристик для каждого образца
random_state=0) # для создания набора формируются случайные числа
# Если равно целому числу, то результат генерации
# будет повторяться для следующих вызовов функции

```

Визуализируем выборку с помощью точечной диаграммы. Столбец 1 взять за x, столбец 2 - за y. В качестве цвета должна выступать метка класса. Цвет можно оставить по умолчанию или задать свой, главное, чтобы разные классы (метки в y) различались по цвету.

```

showData(X, y)
# Всего два признака - x1 и x2 - точки на графике, ответ y -
цвет точки на графике.
# Выведем признаки для одной точки из каждой группы
# Значения могут отличаться при последующих вызовах,
# см. выше аргумент random_state = 0 при вызове make_blobs()
cod_sampl = []      # список кодов классов
for i in range(N_sampl):    # Для всех строк данных
    if y[i] not in cod_sampl:    # с таким кодом данные не выводились
        cod_sampl.append(y[i])  # Добавим код в список
        print('i = %5d x = %9.6f y = %9.6f cod = %3d'%(i, X[i][0], X[i][1], y[i]))
    if len(cod_sampl) == N_class: # У нас только три класса
        break

```

Разделим выборку в соотношении 80 на 20 для X и y на обучающую и тестовую. Общепринятые обозначения X_train, y_train (обучающая выборка - предикаты и ответы), X_test, y_test (тестовая выборка - предикаты и ответы, не участвует в этапе обучения).

```

#Разделение выборки на обучающую (получить предикаты и ответы:X_train, y_train)
in)
# и тестовую (предикаты и ответы: X_test, y_test)
# test_size = 0.20 - размер тестовой выборки в долях: 20%
from sklearn.model_selection import train_test_split
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size = 0.20)
# отобразить обучающую выборку на точечной диаграмме по аналогии с заданием
м выше
showData(X_train, y_train)
# отобразить тестовую выборку на точечной диаграмме по аналогии с заданием
выше
showData(X_test, y_test)

```

Стадия обработки полученных выборок. Алгоритм kNN

1. Реализуем алгоритм, который принимает на вход а) группу объектов (массив точек), для которой нужно получить предсказание (определить класс) б) обучающую выборку - группу объектов с известными параметрами (+ номер класса) в) число ближайших соседей, по которым надо выполнить классификацию тестовой выборки г) возвращается список ответы для тестовой группы: класс тестовых точек.
2. Реализуем алгоритм для подсчёта доли верно угаданных классов (мы знаем класс тестовых точек) $q = N_{\text{угаданных}} / N_{\text{ответов}}$.
3. Пишем скрипт для демонстрации работы алгоритма при разных значениях k - числа ближайших соседей.

```

def kNN(X_test, X_train, y_train, k = 5):
    global N_class
    """Алгоритм:

```

1. Вычисляем расстояния до каждого из объектов обучающей выборки от точки из тестовой выборки
2. Отбираем k точек с минимальным расстоянием: $k = 5$
3. Возвращаем класс классифицируемой точки, как класс наиболее часто встречающихся точек в выборке (в нашем случае из пяти точек).

Пример вызова:

```
answers = knn(X_test, X_train, y_train, 5)
```

X_test - характеристики тестовой группы

X_train, y_train - характеристики обучающей группы

k = 5 - число возвращаемых объектов"

```
def dist(a, b):
```

```
    """ Определение характеристики расстояния между точками a и b
```

```
    Поскольку квадрат большего числа больше, то можно не извлекать
    квадратный корень, что ускоряет вычисления."
```

```
    length = (a[0] - b[0])**2 + (a[1] - b[1])**2
```

```
    return length
```

```
tstL = []          # список для возвращаемых значений
```

```
for tst_point in X_test: # Для каждой точки из группы X_test
```

```
    # Вычисляем расстояние между этой точкой
```

```
    # и всеми точками обучающей выборки
```

```
    # Получаем список с расстояниями и классами
```

```
    testDist = [[dist(tst_point, X_train[i]), y_train[i]] for i in range(len(X_train))]
```

```
    # Как много точек каждого класса среди ближайших k точек
```

```
    stat = [0] * N_class          # список с нулевыми элементами
```

```
    for d in sorted(testDist)[0:k]: # сортируем по расстоянию и выбираем первые
```

к точек

```
        stat[d[1]] += 1    # d[1] - это код класса, и в соответствующем элементе
```

списка накапливается число ближайших точек с кодом класса

Назначаем класс для тестовых точек на основе класса большинства ближайших соседей

```
        tstL.append(sorted(zip(stat, range(N_class)), reverse = True)[0][1])
```

```
# возвращаем список прогнозов
```

```
    return tstL
```

```
answers = kNN(X_test, X_train, y_train, k = 5)
```

считаем долю верно угаданных ответов

```
def calculate_accuracy(preds, y_test):
```

```
    acc = sum(list(map(lambda a, b: 0 if a - b else 1,
                        preds,
```

```
                        y_test))) / len(y_test)
```

```
    return acc
```

```
for k in range(1, 100, 10):
```

```
    preds = kNN(X_test, X_train, y_train, k)
```

```
    r = calculate_accuracy(preds, y_test)
```

```
    print('k = %4d Доля угаданных = %5f% (k,r))
```

Вывод после запуска кода:

k = 1 Доля угаданных = 0.884500

k = 11 Доля угаданных = 0.920500

k = 21 Доля угаданных = 0.919500

k = 31 Доля угаданных = 0.920500

k = 41 Доля угаданных = 0.918500

$k = 51$ Доля угаданных = 0.917500
 $k = 61$ Доля угаданных = 0.919000
 $k = 71$ Доля угаданных = 0.920000
 $k = 81$ Доля угаданных = 0.920000
 $k = 91$ Доля угаданных = 0.919000

Рисунок 1 – диаграмма алгоритма kNN

kNN — один из простейших алгоритмов классификации. Как kNN понимает какие точки находятся ближе всего? Для непрерывных данных kNN использует метрику, например, Евклидову дистанцию. Выбор метрики зависит от типа данных.

1. Бастиан Шарден, Лука Массарон, Альберто Боскетти. «Крупномасштабное машинное обучение вместе с Python». Издательство ДМК Пресс. 2017 год.
2. Свейгарт, Эл. Автоматизация рутинных задач с помощью Python: практическое руководство для начинающих. Пер. с англ. — М.: Вильямс, 2016. — 592 с.
3. Себастьян Рашка. «Python и машинное обучение». Издательство ДМК Пресс. 2017 год.
4. Федоров, Д. Ю. Программирование на языке высокого уровня Python : учебное пособие для прикладного бакалавриата / Д. Ю. Федоров. — 2-е изд., перераб. и доп. — Москва : Издательство Юрайт, 2019. — 161 с. — (Бакалавр. Прикладной курс). — ISBN 978-5-534-10971-9. — Текст: электронный // ЭБС Юрайт [сайт]. — URL: <https://urait.ru/bcode/437489> (дата обращения: 13.02.2020).
5. Шелудько, В. М. Язык программирования высокого уровня Python. Функции, структуры данных, дополнительные модули: учебное пособие / В. М. Шелудько. — Ростов-на-Дону, Таганрог: Издательство Южного федерального университета, 2017. — 107 с. — ISBN 978-5-9275-2648-2. — Текст: электронный // Электронно-библиотечная система IPR BOOKS: [сайт]. — URL: <http://www.iprbookshop.ru/87530.html> (дата обращения: 13.02.2020). — Режим доступа: для авторизир. Пользователей