Федеральное государственное бюджетное образовательное учреждение высшего образования “Сибирский государственный университет телекоммуникаций и информатики”

Лабораторная работа по теме “Метод k ближайших соседей”

Выполнили:  
Студенты группы ИП-712  
Коротеев Максим Сергеевич

Бахарев Максим Евгеньевич

Работу проверил:

ассистент кафедры ПМиК  
Морозова К.И.

# Задание

Написать классификатор на основе метода k ближайших соседей. По варианту необходимо использовать метод парзеновского окна с относительным размером окна, используя квартическое ядро.

# Процесс выполнения работы

1. Данные были разбиты на обучающий и тестовый наборы с соотношением 2:1
2. Данные в обучающем наборе были нормализованы при помощи метода Z-оценки
3. Для обучающего набора было найдено оптимальное количество соседей, дающее больше всего правильных ответов.
4. Данные в тестовом наборе были нормализованы
5. Тестовый набор был проверен с полученным числом k. Точность составила 98.2%

# Вывод

Для данной задачи используемый метод даёт хорошие результаты.

# Листинг

|  |
| --- |
| import csv  import math  import random  import typing  from collections import defaultdict  def split\_by\_classes(raw\_data: typing.Iterable[typing.Sequence[int]]) -> typing.DefaultDict[int, typing.List[int]]:  """Разделяет данные на классы, к которым они относятся"""  splitted\_data = defaultdict(list)  for item in raw\_data:  splitted\_data[item[-1]].append(item)  return splitted\_data  def split\_by\_training\_and\_test\_sets(  raw\_data: typing.Iterable[typing.Sequence[int]],  ) -> typing.Tuple[typing.List[typing.List[int]], typing.List[typing.List[int]]]:  """Разделяет входной набор данные на обучающий и тестовый набор"""  splitted\_data = split\_by\_classes(raw\_data)  training\_set, test\_set = [], []  for data\_class, items in splitted\_data.items():  random.shuffle(items)  size\_of\_test\_set = len(items) // 3  test\_set += items[:size\_of\_test\_set]  training\_set += items[size\_of\_test\_set:]  return training\_set, test\_set  def quart\_core(r: float) -> float:  """Квартическое ядро"""  return (1 - r \*\* 2) \*\* 2 if abs(r) < 1 else 0  def distance(x1: float, y1: float, x2: float, y2: float) -> float:  """Возвращает расстояние между двумя точками на плоскости"""  return math.sqrt((x2 - x1) \*\* 2 + (y2 - y1) \*\* 2)  def expected\_value(data: typing.Collection[float]) -> float:  """Возвращает математическое ожидание случайной величины"""  return sum(data) / len(data)  def std\_deviation(data: typing.Collection[float], expected: float):  """Возвращает среднекрвадратическое отклонение"""  return math.sqrt(sum((elem - expected) \*\* 2 for elem in data) / len(data))  def normalize\_data(data: typing.Collection[float], expected: float, deviation: float) -> typing.List[float]:  """Z-score нормализация данных"""  return [(x - expected) / deviation for x in data]  def remove\_doubles(data):  return list(set(data))  raw\_data = csv.reader(open(**"data5.csv"**))  next(raw\_data)  raw\_data = list(raw\_data)  random.shuffle(raw\_data)  raw\_data = raw\_data[:500] # Берём лишь кусок исходного набора для ускорения работы  converted\_data = (tuple(map(int, line)) for line in raw\_data)  train\_set, t\_set = split\_by\_training\_and\_test\_sets(converted\_data)  train\_set = remove\_doubles(train\_set)  mrot\_in\_hour\_list = [x[0] for x in train\_set]  salary\_list = [x[1] for x in train\_set]  mrot\_in\_hour\_expected, salary\_expected = expected\_value(mrot\_in\_hour\_list), expected\_value(salary\_list)  mrot\_in\_hour\_deviation, salary\_deviation = (  std\_deviation(mrot\_in\_hour\_list, mrot\_in\_hour\_expected),  std\_deviation(salary\_list, salary\_expected),  )  mrot\_in\_hour\_normalized = normalize\_data(mrot\_in\_hour\_list, mrot\_in\_hour\_expected, mrot\_in\_hour\_deviation)  salary\_normalized = normalize\_data(salary\_list, salary\_expected, salary\_deviation)  normalized\_train\_set = [  [mrot\_in\_hour\_normalized[i], salary\_normalized[i], train\_set[i][-1]] for i in range(len(train\_set))  ]  def calc\_distances(dot, all\_set):  distances = []  for d in all\_set:  distances.append(distance(dot[0], dot[1], d[0], d[1]))  return distances  def calc\_hits(k: int, set):  hits = 0  for dot in set:  distances = calc\_distances(dot, set)  z = list(zip(distances, set))  z.sort(key=lambda x: x[0])  assert z[0][0] == 0.0  neighbours = z[1 : k + 2]  max\_range = neighbours[-1][0]  s = 0  for neigh in neighbours:  if neigh[-1][-1] == 1:  mult = 1  elif neigh[-1][-1] == 0:  mult = -1  qc = quart\_core(neigh[0] / max\_range) \* mult  s += qc  if s < 0 and dot[-1] == 0 or s > 0 and dot[-1] == 1:  hits += 1  return hits  max\_val = 0  max\_k = 0  for k in range(1, len(normalized\_train\_set) - 1):  a = calc\_hits(k, normalized\_train\_set)  print(**f"k =** {k}**, hits =** {a} **/** {len(normalized\_train\_set)}**"**)  if a > max\_val:  max\_val = a  max\_k = k  print(**"max k"**, max\_k)  test\_mrot\_in\_hour\_list = [x[0] for x in t\_set]  test\_salary\_list = [x[1] for x in t\_set]  test\_mrot\_in\_hour\_normalized = normalize\_data(test\_mrot\_in\_hour\_list, mrot\_in\_hour\_expected, mrot\_in\_hour\_deviation)  test\_salary\_normalized = normalize\_data(test\_salary\_list, salary\_expected, salary\_deviation)  def get\_class\_of\_dot(k, set, dot):  distances = calc\_distances(dot, set)  z = list(zip(distances, set))  z.sort(key=lambda x: x[0])  neighbours = z[: k + 2]  max\_range = neighbours[-1][0]  s = 0  for neigh in neighbours:  if neigh[-1][-1] == 1:  mult = 1  elif neigh[-1][-1] == 0:  mult = -1  qc = quart\_core(neigh[0] / max\_range) \* mult  s += qc  if s < 0:  return -1  else:  return 1  normalized\_test\_set = [  [test\_mrot\_in\_hour\_normalized[i], test\_salary\_normalized[i], t\_set[i][-1]] for i in range(len(t\_set))  ]  hits = 0  for dot in normalized\_test\_set:  dot\_class = -1 if dot[-1] == 0 else 1  if get\_class\_of\_dot(max\_k, normalized\_train\_set, dot) == dot\_class:  hits += 1  print(**f"Test set accuracy:** {hits}**/**{len(normalized\_test\_set)} **-** {hits / len(normalized\_test\_set) \* 100:**.3**}**%"**) |