Университет ИТМО

Кафедра информатики и прикладной математики

Машинное обучение

Лабораторная работа №1

Метрические алгоритмы классификации

Выполнили: Иппо Вера, группа P4117

Преподаватель:

Санкт-Петербург
2017

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/Letter+Recognition

Предметная область: буквы латинского алфавиита

Задача: определить, какой из букв латинского алфавита соответствует набор характеристик ее написания.

Количество записей: 20000

Количество атрибутов: 16

Атрибуты:

1. lettr capital letter (26 values from A to Z)
2. x-box horizontal position of box (integer)
3. y-box vertical position of box (integer)
4. width width of box (integer)
5. high height of box (integer)
6. onpix total # on pixels (integer)
7. x-bar mean x of on pixels in box (integer)
8. y-bar mean y of on pixels in box (integer)
9. x2bar mean x variance (integer)
10. y2bar mean y variance (integer)
11. xybar mean x y correlation (integer)
12. x2ybr mean of x * x * y (integer)
13. xy2br mean of x * y * y (integer)
14. x-ege mean edge count left to right (integer)
15. xegvy correlation of x-ege with y (integer)
16. y-ege mean edge count bottom to top (integer)

17. yegvx correlation of y-ege with x (integer)

3. Ход работы

Реализация алгоритма Naive Bayes.

```
import math
import pandas
import numpy
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
# разделение датасета на тестовую и обучающую выборку
def load_dataset(test_size):
	dataset = pandas.read_csv("letter-recognition.csv", header=None).values
	letter_attr = dataset[:,1:] # список атрибутов (признаков) для каждой буквы
	letter_class = dataset[:,0] # классы букв
	data_train, data_test, class_train, class_test = train_test_split(letter_attr, letter_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
	return sum(numbers) / float(len(numbers))
```

```
def stand_dev(numbers): # вычисление дисперсии
    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():
    # Среднее значение и среднеквадратичное отклонение атрибутов для каждого класса
    входных данных
    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 эпсилон, 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 classificate_one(summaries, instance_attr):
        # вычисление вероятности принадлежности объекта к каждому из классов
        probabilities = calc_class_probabilities(summaries, instance_attr)
        best_class = None
        max_probability = -1
        for class_name, probability in probabilities.items():
        if best_class is None or probability > max_probability:
                max_probability = probability
                best_class = class_name
        return best_class
# классификация тестовой выборки
def classificate(summaries, data_test):
        predictions = []
        for i in range(len(data_test)):
        result = classificate_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_answer = 0
        # классификация тестовой выборки
        predictions = classificate(summaries, data_test)
        for i in range(len(data_test)):
        if class_test[i] == predictions[i]:
                correct_answer += 1
        return correct_answer / float(len(data_test))
def main():
        data_train, class_train, data_test, class_test = load_dataset(4000)
        summary = summarize_by_class(data_train, class_train)
        accuracy = calc_accuracy(summary, 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()
```

Сравнение работы реализованного алгоритма с библиотечным:

myNBClass Accuracy: 0.63925

sklNBClass Accuracy: 0.638

Реализация алгоритма K Nearest Neighbours

```
from __future__ import division
import pandas
import numpy
```

```
import operator
from sklearn.model_selection import train_test_split
from math import sqrt
from collections import Counter
from sklearn.neighbors import KNeighborsClassifier
def load_dataset(test_size):
    dataset = pandas.read_csv("letter-recognition.csv", header=None).values
    letter_attr = dataset[:,1:] # список атрибутов (признаков) для каждой буквы
    letter_class = dataset[:,0] # классы букв
    data_train, data_test, class_train, class_test = train_test_split(letter_attr, letter_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))

# рассчет расстояний до всех объектов в датасете
def calc_neighbours_distance(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_most_common(neigbours):
	return Counter(neigbours).most_common()[0][0][1]


# классификация тестовой выборки
def get_predictions(data_train, class_train, data_test, k):
	predictions = []
	for j in data_test:
	neighbours = calc_neighbours_distance(j, data_train, class_train, k)
	most_common = get_most_common(neighbours)
	predictions.append(most_common)
	return predictions


# измерение точности
def calc_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)
#Сравнение работы реализованного алгоритма с библиотечным:
def main():
	data_train, class_train, data_test, class_test = load_dataset(4000)
	accuracy = calc_accuracy(data_train, class_train, data_test, class_test, 15)
	print('myKNeighboursClass ', 'Accuracy: ', accuracy)
	clf = KNeighborsClassifier(n_neighbors=15)
	clf.fit(data_train, class_train)
	print('sklKNeigboursClass ', 'Accuracy: ', clf.score(data_test, class_test))
main()
```

Сравнение работы реализованного алгоритма с библиотечным:

myKNeighboursClass Accuracy: 0.8665

sklKNeigboursClass Accuracy: 0.9425

4. Выводы

В ходе лабораторной работы были реализованы два алгоритма классификации: наивный алгоритм Байеса и метод ближайших соседей. Оба алгоритма были протестированы на датасете из 20000 элементов, из которых 4000 были тестовой выборкой, а 16000 – обучающей и была измерена точность классификации. Алгоритм ближайших соседей более точен, чем алгоритм Байеса. Библиотечный алгоритм ближайших соседей ожидаемо лучше реализованного самостоятельно, но наивный алгоритм Байеса из библиотеки scikit-learn показал немного меньшую точность, чем реализованный, что может быть вызвано небольшой погрешностью и недостаточностью обучающей выборки.