Университет ИТМО
Кафедра ИПМ

Машинное обучение
Лабораторная работа 1
«Метрические алгоритмы классификации»

Выполнил:
Шаймарданов Руслан
группа P4117
Преподаватель:
Жукова Н. А.

Санкт-Петербург
2017

1.Постановка задачи

- На языке Python программно реализовать два метрических алгоритма классификации: Naive Bayes и K Nearest Neighbours
- Сравнить работу реализованных алгоритмов с библиотечными из scikit-learn
- Для тренировки, теста и валидации использовать один из предложенных датасетов
- Сформировать краткий отчет

2. Выбранный датасет: «Statlog (Shuttle) Data Set»

http://archive.ics.uci.edu/ml/datasets/Statlog+%28Shuttle%29

Количество записей: 14500

Количество атрибутов: 9

3. Алгоритм Naive Bayes (naiveBayes.py)

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

def split_dataset(test_size):
    dataset = pd.read_csv("shuttle.csv", header=None).values
    attr = dataset[:, :-1].astype(np.int32, copy=False) # атрибуты
    classes = dataset[:, -1].astype(np.int32, copy=False) # классы
        data_train, data_test, class_train, class_test = train_test_split(attr, classes, 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

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):
    summaries = [(mean(att_numbers), stand_dev(att_numbers)) for att_numbers in zip(*data_train)]
    return summaries

def summarize_by_class(data_train, class_train): # Обучение классификатора
    # 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 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 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
    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()
```

Вывод программы:

myNBClass Accuracy: 0.529362591431557

sklNBClass Accuracy: 0.835945663532

4. Алгоритм K Nearest Neighbours (neighbours.py)

```
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

def load_data(filename):
    dataset = pd.read_csv(filename, header=None).values
    attr = dataset[:, :-1].astype(np.int32, copy=False) # атрибуты
    classes = dataset[:, -1].astype(np.int32, copy=False) # классы
    return train_test_split(attr, classes, test_size=0.35)

# 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 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(neigbours):
    return Counter(neigbours).most_common()[0][0][1]

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

```
        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, data_test, class_train, class_test = load_data("shuttle.csv")
    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()
```

Вывод программы:

myKNClass Accuracy: 0.992315270936

sklKNClass Accuracy: 0.994876847291

5. Вывод

Результаты работы показали, что библиотечная реализация является более точной в обоих случаях, что логично, так как в ней наверняка продуманы многие нюансы. Не самая высокая точность обучения алгоритмом Naive Bayes объясняется тем, что из семи классов первый встречается в 80% случаев. Полагаю, из-за этой же особенности алгоритм K Nearest Neighbours показал точность, приближенную к единице.