Университет ИТМО

Машинное обучение
Лабораторная работа №1

Студент: Маскайкин А.В.
Группа: P4117

Санкт-Петербург
2017г

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 (либо найти самостоятельно и внести в таблицу). Сформировать краткий отчет (постановка задачи, реализация, эксперимент с данными, полученные характеристики, вывод).

2. Исходные данные

- Датасет: https://archive.ics.uci.edu/ml/datasets/Website+Phishing
- Предметная область: Фишинговые сайты
- Задача: определить, фишинговый, подозрительный или нормальный сайт
- Количество записей: 1353
- Количество атрибутов: 9
- Атрибуты:
  1. SFH {1,-1,0}
  2. Pop-up Window {1,-1,0}
  3. SSL final state {1,-1,0}
  4. Request URL {1,-1,0}
  5. URL of Anchor {1,-1,0}
  6. Web traffic {1,-1,0}
  7. URL Length {1,-1,0}
  8. Age of domain {1,-1}
  9. Having IP Address {1,-1}

Во всех характеристиках значение «-1» означает «фишнговый», «0» - подозрительный, «1» - нормальный.

2.1 Описание параметров

- SFH (Server from handler) — Представление пользовательской информации, которая передается из веб страницы на сервер. Если оно пустое — сайт фишинговый, если передача идет на другой домен — подозрительный.
- Pop-up Window — Наличие всплывающего окна. Если при окне не доступен правый клик, то сайт фишинговый.
- SSL final state — Подлинность SSL сертификата.
- Request URL — Количество запросов к веб странице. Если их много, то, вероятно, сайт подвергся атаке, которая заменяет содержимое (текст/картинки). Если количество запросов велико — сайт фишинговый.
- URL of Anchor — привязка к URL. Если при вводе адреса сайта в браузере происходит редирект на другой домен, то привязки нет. И если процент редиректов большой — сайт фишинговый.

- Web traffic — объем веб трафиика сайта. У нормальных сайтов объем высокий, у фишинговых — низкий.
- URL Length — Длина адреса сайта. Чем больше длина, тем выше вероятность, что в адрес встроен вредоносный код.
- Age of domain — Возраст сайта. Если сайт существует менее полугода, то его можно заподозрить как фишинговый.
- Having IP Address — Наличие IP адреса. Если адреса нет — сайт фишинговый.

3. Реализация алгоритмов.

Реализация алгоритма Naive Bayes.

```
# coding=utf-8
import math
import numpy as np
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.naive_bayes import GaussianNB
# 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('fs.dataset.csv')
    site_attr = dataset[:, :-1]  # список атрибутов для каждого сайта
    site_class = dataset[:, -1]  # класс (результат) сайта (норм,
подозрительный, фишинговый)
    site_class = site_class.astype(np.int64, copy=False)
    data_train, data_test, class_train, class_test = \
        train_test_split(site_attr, site_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 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(0.25)
    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))
if __name__ == '__main__':
    main()
```

Реализация алгоритма K Nearest Neighbours

```
# coding=utf-8

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('fs.dataset.csv')
    site_attr = dataset[:, :-1]  # список атрибутов для каждого сайта
    site_class = dataset[:, -1]  # класс (результат) сайта (норм,
подозрительный, фишинговый)
    site_class = site_class.astype(np.int64, copy=False)
    data_train, data_test, class_train, class_test = \
        train_test_split(site_attr, site_class, test_size=test_size,
random_state=55)
    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 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, class_train, data_test, class_test = split_dataset(0.3)
    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))
if __name__ == '__main__':
    main()
```

4. Результаты работы алгоритмов.

Naive Bayes

('myNBClass ', 'Accuracy: ', 0.7728613569321534)

('sklNBClass ', 'Accuracy: ', 0.79351032448377579)

K Nearest Neighbours

('myKNClass', 'Accuracy: ', 0.77832512315270941)

('sklKNClass', 'Accuracy: ', 0.79556650246305416)

По результатм работы оба алгоритма (Naive Bayes и K Nearest Neighbours) показали схожий результат на данном датасете. Разработанные алгоритмы оказались довольно точны: результаты их работы на доли сотых отличаются от библиотечных.