Федеральное государственное бюджетное образовательное учреждение высшего образования

«Сибирский государственный университет телекоммуникаций и информатики»

(СибГУТИ)

Кафедра прикладной математики и кибернетики

Отчет

по лабораторной работе №2

Логические классификаторы. Решающие деревья

Вариант 5

Выполнил:

студент гр. ИП-612

Быков А. С.

Проверил:

Ассистент кафедры  
Бочкарев Б. В.

Новосибирск, 2019 г.

Оглавление

[Теоретический базис 3](#_Toc21114431)

[Результаты 6](#_Toc21114432)

[Код программы 8](#_Toc21114433)

# Теоретический базис

Пусть — некоторый предикат, определённый на множестве объектов X. Говорят, что предикат ϕ выделяет или покрывает (cover) объект x, если . Предикат называют закономерностью, если он выделяет достаточно много объектов какого-то одного класса c, и практически не выделяет объекты других классов. Особую ценность представляют закономерности, которые описываются простой логической формулой. Их называют правилами (rules). Процесс поиска правил по выборке называют извлечением знаний из данных (knowledge discovery). К знаниям предъявляется особое требование — они должны быть интерпретируемы, то есть понятны людям. На практике логические закономерности часто ищут в виде конъюнкций небольшого числа элементарных высказываний. Именно в такой форме люди привыкли выражать свой житейский и профессиональный опыт.

Понятие информативности

Введём следующие обозначения:

* — число объектов класса в выборке ;
* — из них число объектов, для которых выполняется ;
* — число объектов всех остальных классов в выборке ;
* — из них число объектов, для которых выполняется .

Итак, задача построения информативного предиката сводится к оптимизации по двум критериям: и . Введём обозначение для доли негативных среди всех выделяемых объектов, и для доли выделяемых позитивных объектов:

**Опр.** Предикат будем называть логической -закономерностью для класса , если и при заданных достаточно малом и достаточно большом из отрезка [0, 1].

Если , то закономерность называется чистой или непротиворечивой. Если , то закономерность называется частичной.

Статистическое определение информативности

Адекватную скалярную характеристику информативности даёт техника проверки статистических гипотез. Пусть — вероятностное пространство, выборка — простая, то есть случайная, независимая, одинаково распределённая (independent, identically distributed — i.i.d.), и — случайные величины. Допустим, справедлива гипотеза о независимости событий . Тогда вероятность реализации пары подчиняется гипергеометрическому распределению:

где — биномиальные коэффициенты, . Если вероятность мала, и тем не менее пара реализовалась, то гипотеза о независимости должна быть отвергнута.

**Опр.** Информативность предиката относительно класса по выборке есть

Предикат будем называть статистической закономерностью для класса , если при заданном достаточно большом . Порог информативности выбирается так, чтобы ему соответствовало достаточно малое значение вероятности, называемое уровнем значимости. Для каждой задачи он должен выбираться индивидуально.

Энтропийное определение информативности

Ещё один способ определения информативности вытекает из теории информации. Энтропия определяется как матожидание количества информации:

.

Будем считать появление объекта класса c исходом , а появление объекта любого другого класса исходом . Тогда, подставляя вместо вероятностей частоты, можно оценить энтропию выборки :

Допустим, стало известно, что предикат выделил объектов из принадлежащих классу , и объектов из , не принадлежащих классу . Тогда энтропия выборки есть . Вероятность появления объекта из этой выборки оценивается как . Аналогично, энтропия выборки есть , а вероятность появления объекта из неё оценивается как . Таким образом, энтропия всей выборки после получения информации становится равна

.

В итоге уменьшение энтропии составляет

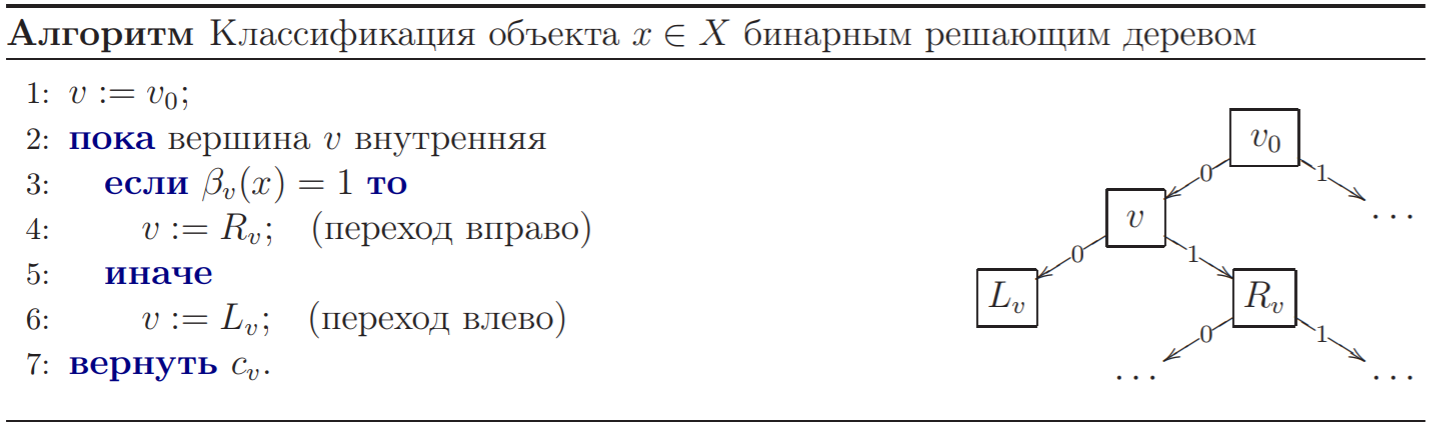
.

Это и есть информационный выигрыш (information gain) — количество информации об исходном делении выборки на два класса «c» и «не c», которое содержится в предикате . Таким образом, появляется ещё одно, альтернативное, определение закономерности.

Опр. Предикат является закономерностью по энтропийному критерию информативности, если при некотором достаточно большом .

Решающие деревья

**Опр.** Бинарное решающее дерево — это алгоритм классификации, задающийся бинарным деревом, в котором каждой внутренней вершине приписан предикат , каждой терминальной вершине приписано имя класса . При классификации объекта он проходит по дереву путь от корня до некоторого листа, в соответствии с Алгоритмом



Объект доходит до вершины v тогда и только тогда, когда выполняется конъюнкция , составленная из всех предикатов, приписанных внутренним вершинам дерева на пути от корня до вершины . Пусть — множество всех терминальных вершин дерева. Множества объектов , выделяемых терминальными конъюнкциями , попарно не пересекаются, а их объединение совпадает со всем пространством (это легко доказывается индукцией по числу вершин дерева). Отсюда следует, что решающее дерево никогда не отказывается от классификации, в отличие от решающего списка. Отсюда также следует, что алгоритм классификации , реализуемый бинарным решающим деревом, можно представить в виде простого голосования конъюнкций:

причём для любого одно и только одно слагаемое во всех этих суммах равно единице. Вместо суммирования можно было бы использовать и дизъюнкцию. Естественное требование максимизации информативности конъюнкций означает, что каждая из них должна выделять как можно больше обучающих объектов, допуская при этом как можно меньше ошибок. Для повышения обобщающей способности решающего дерева число листьев должно быть как можно меньше, и они должны покрывать подвыборки примерно одинаковой мощности .

Случайный лес (Random Forest)

Random forest — алгоритм машинного обучения, заключающийся в использовании комитета (ансамбля) решающих деревьев. Основная идея заключается в использовании большого ансамбля решающих деревьев, каждое из которых само по себе даёт очень невысокое качество классификации, но за счёт их большого количества результат получается хорошим.

*Алгоритм обучения классификатора*

Пусть обучающая выборка состоит из N образцов, размерность пространства признаков равна M, и задан параметр m (в задачах классификации обычно ) как неполное количество признаков для обучения.

Наиболее распространённый способ построения деревьев комитета следующий:

* Сгенерируем случайную подвыборку с повторениями размером N из обучающей выборки. (Таким образом, некоторые образцы попадут в неё два или более раза, а в среднем , а примерно N/e образцов не войдут в неё вообще).
* Построим решающее дерево, классифицирующее образцы данной подвыборки, причём в ходе создания очередного узла дерева будем выбирать набор признаков, на основе которых производится разбиение (не из всех M признаков, а лишь из m случайно выбранных). Выбор наилучшего из этих m признаков может осуществляться различными способами. В оригинальном коде Бреймана используется критерий Джини, применяющийся также в алгоритме построения решающих деревьев CART. В некоторых реализациях алгоритма вместо него используется критерий прироста информации.
* Дерево строится до полного исчерпания подвыборки и не подвергается процедуре прунинга
* Классификация объектов проводится путём голосования: каждое дерево комитета относит классифицируемый объект к одному из классов, и побеждает класс, за который проголосовало наибольшее число деревьев.

Оптимальное число деревьев подбирается таким образом, чтобы минимизировать ошибку классификатора на тестовой выборке. В случае её отсутствия, минимизируется оценка ошибки out-of-bag: тех образцов, которые не попали в обучающую подвыборку за счёт повторений (их примерно N/e).

# Результаты

В качестве логического классификатора был выбран случайный лес, состоящий из 15 решающих деревьев.

В таблице представлены результаты для 10 независимых запусков программы, выполняющей на старте случайное разбиение входных данных на обучающую и тестовую выборки (с минимизацией разницы между количеством представленных в обучающей выборке объектов разных классов).

Значения в ячейках – отношение количества правильно классифицированных объектов к размеру соответствующей выборки.

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| № запуска  выборка | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
| обучающая | 0.92 | 0.91 | 0.85 | 0.91 | 0.88 | 0.93 | 0.95 | 0.90 | 0.92 | 0.94 |
| тестовая | 0.86 | 0.85 | 0.79 | 0.87 | 0.87 | 0.88 | 0.73 | 0.84 | 0.86 | 0.86 |

# Код программы

import csv

import math

import random

from collections import defaultdict

from collections import Counter

from typing import \*

import numpy as np

from sklearn import tree

from sklearn.impute import SimpleImputer

*class* DataItem:

*def* \_\_init\_\_(*self*, *row*: *list*):

        self.attributes = [None] \* 13

        for i in range(len(self.attributes)):

            self.attributes[i] = None if row[i] == "?" else row[i]

        self.clazz = row[len(row) - 1]

*def* \_\_str\_\_(*self*):

        return *str*(self.attributes) + ", " + *str*(self.clazz)

*class* DecisionTree:

*def* \_\_init\_\_(*self*, *train\_data*: List[DataItem]):

        train\_data = [random.choice(train\_data) for i in range(len(train\_data))]

        n\_attributes = len(train\_data[0].attributes)

        m = *int*(math.sqrt(n\_attributes))

        self.attr\_subset\_indexes = random.sample(range(n\_attributes), m)

        attribute\_subset = [subset\_by\_indexes(item.attributes, self.attr\_subset\_indexes) for item in train\_data]

        classes = [item.clazz for item in train\_data]

        self.sk\_dec\_tree = tree.DecisionTreeClassifier()

        self.sk\_dec\_tree.fit(attribute\_subset, classes)

*def* predict(*self*, *X*) -> *list*:

        X = [subset\_by\_indexes(attrs, self.attr\_subset\_indexes) for attrs in X]

        return self.sk\_dec\_tree.predict(X)

*class* Forest:

*def* \_\_init\_\_(*self*, *trees*: List[DecisionTree]):

        self.trees = trees

*def* predict(*self*, *X*) -> *list*:

        vote\_table = zip(\*[dec\_tree.predict(X) for dec\_tree in self.trees])

        return [one\_most\_common(votes) for votes in vote\_table]

*def* one\_most\_common(*l*):

        return Counter(l).most\_common(1)[0][0]

*def* subset\_by\_indexes(*l*: *list*, *indexes*: List[*int*]) -> *list*:

    return [l[i] for i in indexes]

*def* read\_csv(*file*: *str*) -> List[DataItem]:

    with (open(file)) as csv\_file:

        reader = csv.reader(csv\_file)

        next(reader)  # skip header

        return [DataItem(row) for row in reader]

*def* group\_by\_class(*data*: List[DataItem]) -> *dict*:

    result = defaultdict(*list*)

    for entry in data:

        result[entry.clazz].append(entry)

    return result

*def* split\_data(*data*: List[DataItem], *train\_data\_ratio*: *float*) -> Tuple[List[DataItem], List[DataItem]]:

    if train\_data\_ratio <= 0:

        return [], data

    if train\_data\_ratio >= 1:

        return data, []

    n = len(data)

    n\_training = *int*(n \* train\_data\_ratio)

    grouped\_by\_class = group\_by\_class(data)

    classes = *list*(grouped\_by\_class.keys())

    train\_data = []

    test\_data = []

    class\_i = 0

    for i in range(n):

        class\_i = (class\_i + 1) % len(classes)

        clazz = classes[class\_i]

        group = grouped\_by\_class[clazz]

        if len(group) == 0:

            classes.remove(clazz)

            continue

        group.pop

        item = group.pop(random.randint(0, len(group) - 1))

        target\_data = train\_data if i < n\_training else test\_data

        target\_data.append(item)

    return train\_data, test\_data

*def* fill\_missing\_values(*data*: List[DataItem]) -> List[DataItem]:

    imp = SimpleImputer(*missing\_values*=np.nan, *strategy*='mean')

    attributes = [item.attributes for item in data]

    imp = imp.fit(attributes)

    attributes\_imp = imp.transform(attributes)

    return [DataItem([\*attributes\_imp[i], data[i].clazz]) for i in range(len(data))]

*def* count\_matches(*forest*: Forest, *test\_data*: List[DataItem]) -> *float*:

    matches = 0

    predicted: *list* = forest.predict([item.attributes for item in test\_data])

    for i in range(len(predicted)):

        if predicted[i] == test\_data[i].clazz:

            matches += 1

    return matches / len(test\_data)

*def* main():

    all\_data = read\_csv('data.csv')

    all\_data = fill\_missing\_values(all\_data)

    (train\_data, test\_data) = split\_data(all\_data, 0.7)

    trees = [DecisionTree(train\_data) for i in range(15)]

    forest = Forest(trees)

    train\_matches = '{*:.4f*}'.format(count\_matches(forest, train\_data))

    test\_matches = '{*:.4f*}'.format(count\_matches(forest, test\_data))

    print("Train: " + *str*(train\_matches))

    print("Test: " + *str*(test\_matches))

if \_\_name\_\_ == '\_\_main\_\_':

    main()

    exit()