# Теория

## Naive Bayes Classifier

**Байесовский классификатор** — широкий класс алгоритмов классификации, основанный на принципе максимума апостериорной вероятности. Для классифицируемого объекта вычисляются функции правдоподобия каждого из классов, по ним вычисляются апостериорные вероятности классов. Объект относится к тому классу, для которого апостериорная вероятность максимальна.

**Наивный байесовский классификатор** - частный случай байесовского классификатора, основывающийся на том, что все признаки, описывающие объект, независимы друг от друга.

Пусть X — множество описаний объектов, Y — множество номеров (или наименований) классов. На множестве пар «объект, класс» X \times Yопределена вероятностная мера \mathsf P. Имеется конечная обучающая выборка независимых наблюдений X^m = \{(x_1,y_1),\ldots,(x_m,y_m)\}, полученных согласно вероятностной мере \mathsf P.

**Задача классификации** заключается в том, чтобы построить алгоритм a:\; X\to Y, способный классифицировать произвольный объект x \in X.

В байесовской теории классификации эта задача разделяется на две.

* Построение оптимального классификатора при известных плотностях классов. Эта подзадача имеет простое и окончательное решение.
* Восстановление плотностей классов по обучающей выборке. В этой подзадаче сосредоточена основная сложность байесовского подхода к классификации.

### **Построение классификатора при известных плотностях классов**

Пусть для каждого класса y \in Y известна *априорная вероятность* P_y того, что появится объект класса y, и плотности распределения p_y(x)каждого из классов, называемые также *функциями правдоподобия* классов. Требуется построить алгоритм классификации a(x), доставляющий минимальное значение функционалу *среднего риска*.

*Средний риск* опредеяется как математическое ожидание ошибки:

R(a) = \sum_{y\in Y} \sum_{s\in Y} \lambda_{y} P_y \mathsf{P}_{(x,y)}\bigl\{a(x)=s|y\bigr\},

где \lambda_{y} — *цена ошибки* или штраф за отнесение объекта класса y к какому-либо другому классу.

**Теорема.** Решением этой задачи является алгоритм

a(x) = \mathrm{arg}\max_{y\in Y} \lambda_{y} P_y p_y(x).

Значение P\{y|x\} = P_y p_y(x) интерпретируется как апостериорная вероятность того, что объект x принадлежит классу y.

Если классы равнозначимы, \lambda_{y} P_y = \mathrm{const}(y), то объект x просто относится к классу с наибольшим значением плотности распределения в точке x.

### **Восстановление плотностей классов по обучающей выборке**

По заданной подвыборке объектов класса y построить эмпирические оценки априорных вероятностей P_y и функций правдоподобия p_y(x).

В качестве оценки априорных вероятностей берут, как правило, долю объектов данного класса в обучающей выборке.

Восстановление плотностей (функций правдоподобия каждого из классов) является наиболее трудной задачей. Наиболее распространены три подхода: параметрический, непараметрический и разделение смеси вероятностных распределений. Третий подход занимает промежуточное положение между первыми двумя, и в определённом смысле является наиболее общим.

* *Параметрическое* восстановление плотности при дополнительном предположении, что плотности нормальные (гауссовские), приводит к нормальному дискриминантному анализу и линейному дискриминанту Фишера.
* *Непараметрическое* восстановление плотности приводит, в частности, к методу парзеновского окна.
* *Разделение смеси распределений* может быть сделано с помощью EM-алгоритма. Дополнительное предположение, что плотности компонент смеси являются радиальными функциями, приводит к методу радиальных базисных функций. Обычно в качестве компонент смеси берут, опять-таки, гауссовские плотности.

Таким образом, формула байесовского классификатора приводит к большому разнообразию байесовских алгоритмов, отличающихся только способом восстановления плотностей.

## K Nearest Neighbor Classifier

**Метод ближайших соседей** — простейший метрический классификатор, основанный на оценивании [сходства](http://www.machinelearning.ru/wiki/index.php?title=%D0%A1%D1%85%D0%BE%D0%B4%D1%81%D1%82%D0%B2%D0%BE&action=edit) объектов. Классифицируемый объект относится к тому классу, которому принадлежат ближайшие к нему объекты обучающей выборки.

**Метод ближайшего соседа** является, пожалуй, самым простым алгоритмом классификации. Классифицируемый объект x относится к тому классу y_i, которому принадлежит ближайший объект обучающей выборки x_i.

**Метод k ближайших соседей**. Для повышения надёжности классификации объект относится к тому классу, которому принадлежит большинство из его *соседей* — k ближайших к нему объектов обучающей выборки x_i. В задачах с двумя классами число соседей берут нечётным, чтобы не возникало ситуаций неоднозначности, когда одинаковое число соседей принадлежат разным классам.

Пусть задана обучающая выборка пар «объект-ответ» X^m = \{(x_1,y_1),\dots,(x_m,y_m)\}.

Пусть на множестве объектов задана функция расстояния \rho(x,x'). Эта функция должна быть достаточно адекватной *моделью сходства* объектов. Чем больше значение этой функции, тем менее схожими являются два объекта x,x'.

Для произвольного объекта u расположим объекты обучающей выборки x_i в порядке возрастания расстояний до u:

\rho(u,x_{1; u}) \leq  \rho(u,x_{2; u}) \leq \cdots \leq \rho(u,x_{m; u}),

где через x_{i; u} обозначается тот объект обучающей выборки, который является i-м соседом объекта u. Аналогичное обозначение введём и для ответа на i-м соседе: y_{i; u}. Таким образом, произвольный объект u порождает свою перенумерацию выборки. В наиболее общем виде алгоритм ближайших соседей есть

a(u) = \mathrm{arg}\max_{y\in Y} \sum_{i=1}^m \bigl[ x_{i; u}=y \bigr] w(i,u),

где w(i,u) — заданная *весовая функция*, которая оценивает степень важности i-го соседа для классификации объекта u. Естественно полагать, что эта функция неотрицательна и не возрастает по i.

По-разному задавая весовую функцию, можно получать различные варианты метода ближайших соседей.

* w(i,u) = [i=1] — простейший метод ближайшего соседа;
* w(i,u) = [i\leq k] — метод k ближайших соседей;
* w(i,u) = [i\leq k] q^i — метод k экспоненциально взвешенных ближайших соседей, где предполагается q < 1;
* w(i,u) = K\biggl(\frac{\rho(u,x_{i; u})}{h}\biggr) — метод парзеновского окна фиксированной ширины h;
* w(i,u) = K\biggl(\frac{\rho(u,x_{i; u})}{\rho(u,x_{k+1; u})}\biggr) — метод парзеновского окна переменной ширины;
* w(i,u) = K\biggl(\frac{\rho(u,x_{i; u})}{h(x_{i; u})}\biggr) — метод потенциальных функций, в котором ширина окна h(x_i) зависит не от классифицируемого объекта, а от обучающего объекта x_i.

Здесь K(r) — заданная неотрицательная монотонно невозрастающая функция на [0,+\infty), ядро сглаживания.

# Задание

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

# Пример выполнения работы

#### **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: <http://archive.ics.uci.edu/ml/datasets/Zoo>

Предметная область: обитатели зоопарка

Задача: определить, к какому из 7ми типов относится каждое животное

Количество записей: 101

Количество атрибутов: 17

Атрибуты:

1. Название животного (строка, уникальный для каждого экземпляра)
2. Наличие волос (логический тип)
3. Наличие перьев (логический тип)
4. Яйца (логический тип)
5. Млекопитающий (логический тип)
6. Умеет летать (логический тип)
7. Водный (логический тип)
8. Хищник (логический тип)
9. Наличие зубов (логический тип)
10. Наличие позвоночника (логический тип)
11. Дышит воздухом (логический тип)
12. Ядовитость (логический тип)
13. Наличие плавников (логический тип)
14. Количество ног (набор целочисленных значений: {0,2,4,5,6,8})
15. Наличие хвоста (логический тип)
16. Является домашним (логический тип)
17. Catsize (логический тип)
18. Тип (целочисленные значения в диапазоне [1,7])

#### **3. Ход работы**

1. Реализация алгоритма Naive Bayes.

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('zoo.data.csv')

animal\_attr = dataset[:, 1:-1] # список атрибутов (признаков) для каждого животного

animal\_class = dataset[:, -1] # классы животных

animal\_class = animal\_class.astype(np.int64, copy=False)

data\_train, data\_test, class\_train, class\_test = train\_test\_split(animal\_attr, animal\_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))

Сравнение работы реализованного алгоритма с библиотечным:

def main():

data\_train, class\_train, data\_test, class\_test = split\_dataset(0.3)

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()

myNBClass Accuracy: 0.9354838709677419

sklNBClass Accuracy: 0.967741935484

1. Реализация алгоритма K Nearest Neighbours

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():

dataset = pd.read\_csv('zoo.data.csv', header=None).values

animal\_attr = dataset[:, 1:-1] # список атрибутов (признаков) для каждого животного

animal\_class = dataset[:, -1] # классы животных

animal\_class = animal\_class.astype(np.int64, copy=False)

return train\_test\_split(animal\_attr, animal\_class, 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))

# cортировка 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)

Сравнение работы реализованного алгоритма с библиотечным:

def main():

data\_train, data\_test, class\_train, class\_test = load\_data()

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()

myKNClass Accuracy: 0.75

sklKNClass Accuracy: 0.75