(40条消息) 概率论中高斯分布(正态分布)介绍及C++11中std::normal_distribution的使用 _fengbingchun的博客-CSDN博客 _normal_distribution

高斯分布：最常用的分布是正态分布(normal distribution)，也称为高斯分布(Gaussian distribution)：

$$N(x;\mu,\sigma^2)=\sqrt{\frac{1}{2\pi\sigma^2}}\exp(-\frac{1}{2\sigma^2}(x-\mu)^2)$$

正态分布$N(x;\mu,\sigma^2)$呈现经典的”钟形曲线”的形状，其中中心峰的x坐标由μ给出，峰的宽度受σ控制。

正态分布由两个参数控制，$\mu\in R$和$\sigma\in(0,\infty)$。参数μ给出了中心峰值的坐标，这也是分布的均值：$E[x]=\mu$。分布的标准差用σ表示，方差用σ^2表示。

当我们要对概率密度函数求值时，我们需要对σ平方并且取倒数。当我们需要经常对不同参数下的概率密度函数求值时，一种更高效的参数化分布的方式是使用参数$\beta\in(0,\infty)$，来控制分布的精度(precision)(或方差的倒数)：

$$N(x;\mu,\beta^{-1})=\sqrt{\frac{\beta}{2\pi}}\exp(-\frac{1}{2}\beta(x-\mu)^2)$$

采用正态分布在很多应用中都是一个明智的选择。当我们由于缺乏关于某个实数上分布的先验知识而不知道该选择怎样的形式时，正态分布是默认的比较好的选择，其中有两个原因：

(1)、我们想要建模的很多分布的真实情况是比较接近正态分布的。中心极限定理(central limit theorem)说明很多独立随机变量的和近似服从正态分布。这意味着在实际中，很多复杂系统都可以被成功地建模成正态分布的噪声，即使系统可以被分解成一些更结构化的部分。

(2)、在具有相同方差的所有可能的概率分布中，正态分布在实数上具有最大的不确定性。因此，我们可以认为正态分布是对模型加入的先验知识量最少的分布。

正态分布可以推广到R^n空间，这种情况下被称为多维正态分布(multivariate normal distribution)。它的参数是一个正定对称矩阵∑：

$$N(x;\mu,\Sigma)=\sqrt{\frac{1}{(2\pi)^n\det(\Sigma)}}\exp(-\frac{1}{2}(x-\mu)^T\Sigma^{-1}(x-$$

参数μ仍然表示分布的均值，只不过现在是向量值。参数∑给出了分布的协方差矩

阵。和单变量的情况类似，当我们希望对很多不同参数下的概率密度函数多次求值时，协方差矩阵并不是一个很高效的参数化分布的方式，因为对概率密度函数求值时需要对∑求逆。我们可以使用一个精度矩阵(precision matrix)β进行替代：

$$N(\mathrm{x};\mu,\beta^{-1})=\sqrt{\frac{\det(\beta)}{(2\pi)^n}}\exp\left(-\frac{1}{2}(\mathrm{x}-\mu)^T\beta(\mathrm{x}-\mu)\right)$$

我们常常把协方差矩阵固定成一个对角阵。一个更简单的版本是各向同性(isotropic)高斯分布，它的协方差矩阵是一个标量乘以单位阵。

正态分布(normal distribution)：又名高斯分布(Gaussian distribution，以德国数学家卡尔·弗里德里希·高斯的姓冠名)，是一个在数学、物理及工程等领域都非常重要的概率分布，由于这个分布函数具有很大非常漂亮的性质，使得其在诸多涉及统计科学离散科学等领域的许多方面都有着重大的影响力。比如图像处理中最常用的滤波器类型为Gaussian滤波器(也就是所谓的正态分布函数)。

若随机变量X服从一个位置参数为μ、尺度参数为σ的概率分布，记为：$X\backsim N(\mu,\sigma^2)$,则其概率密度函数为：

$$f(\mathrm{x})=\frac{1}{\sigma\sqrt{2\pi}}e^{-\frac{(\mathrm{x}-\mu)^2}{2\sigma^2}}$$

正态分布的数学期望值或期望值μ等于位置参数，决定了分布的位置；其方差σ^2的开平方或标准差σ等于尺度参数，决定了分布的幅度。

正态分布的概率密度函数曲线呈钟形，因此人们又经常称之为钟形曲线。我们通常所说的标准正态分布是位置参数μ=0，尺度参数σ^2=1的正态分布。

以上内容摘自： 《深度学习中文版》和 维基百科

以下是对C++11中的正态分布模板类std::normal_distribution的介绍：

C++11中引入了正态分布模板类std::normal_distribution，在头文件<random>中。

正态分布(normal distribution)又名高斯分布(Gaussian distribution)。若随机变量X服从一个数学期望为μ、方差为σ^2的高斯分布，记为$N(\mu,\sigma^2)$。其概率密度函数为正态分布的期望值μ决定了其位置，其标准差σ决定了分布的幅度。通常所说的标准正态分布是μ=0, σ=1的正态分布。

std::normal_distribution:Normal distribution, Random number distribution that produces floating-point values according to a normal distribution, which is described by the following probability density function:

$$p(x\mid\mu,\sigma)=\frac{1}{\sigma\sqrt{2\pi}}\cdot e^{-\frac{(\mathrm{x}-\mu)^2}{2\sigma^2}}$$

This distribution produces random numbers around the distribution mean (μ) with a specific standard deviation (σ).

The normal distribution is a common distribution used for many kind of processes,since it is the distribution that the aggregation of a large number of independent random variables approximates to, when all follow the same distribution (no matter which distribution).

The distribution parameters, mean (μ) and stddev (σ), are set on construction. To produce a random value following this distribution, call its member function operator().

normal distribution represents an unbounded distribution.

以下是std::normal_distribution的测试code：

```
#include "normal_distribution.hpp"
#include <iostream>
#include <random>
#include <string>
#include <chrono>
#include <map>
#include <iomanip>


int test_normal_distribution_1()
{
{
const int nrolls = 10000;
const int nstars = 100;

        std::default_random_engine generator;
std::normal_distribution<double> distribution(5.0, 2.0);

int p[10] = {};

for (int i = 0; i<nrolls; ++i) {
double number = distribution(generator);
if ((number >= 0.0) && (number<10.0)) ++p[int(number)];
        }

        std::cout << "normal_distribution (5.0,2.0):" << std::endl;

for (int i = 0; i<10; ++i) {
                std::cout << i << "-" << (i + 1) << ": ";
```

```
31.                 std::cout << std::string(p[i] * nstars / nrolls, '*') << std
32.         }
33. }
34.
35. {
36.
37.
38.
39.
40.
41.
42.
43.
44.
45. unsigned seed = std::chrono::system_clock::now().time_since_epoch().count();
46. std::default_random_engine generator(seed);
47.
48. std::normal_distribution<double> distribution(0.0, 1.0);
49.
50.         std::cout << "some Normal-distributed(0.0,1.0) results:" << std::end
51. for (int i = 0; i<10; ++i)
52.                 std::cout << distribution(generator) << std::endl;
53.         std::cout << "max: " << distribution.max() << std::endl;
54.         std::cout << "min: " << distribution.min() << std::endl;
55.         std::cout << "mean: " << distribution.mean() << std::endl;
56.         std::cout << "stddev: " << distribution.stddev() << std::endl;
57. }
58.
59. {
60.         std::default_random_engine generator;
61. std::normal_distribution<double> d1(0.0, 1.0);
62. std::normal_distribution<double> d2(d1.param());
63.
64.
65.         std::cout << d1(generator) << std::endl;
66.         std::cout << d2(generator) << std::endl;
67. }
68.
69. {
70.
71.         std::default_random_engine generator;
```

```
72. std::normal_distribution<double> distribution(0.0, 1.0);
73.
74.
75.         std::cout << distribution(generator) << std::endl;
76.         distribution.reset();
77.         std::cout << distribution(generator) << std::endl;
78. }
79.
80. return 0;
81. }
82.
83.
84.
85. int test_normal_distribution_2()
86. {
87.         std::random_device rd;
88. std::mt19937 gen(rd());
89.
90.
91.
92.         std::normal_distribution<> d(5, 2);
93.
94.         std::map<int, int> hist;
95. for (int n = 0; n<10000; ++n) {
96.                 ++hist[std::round(d(gen))];
97.         }
98. for (auto p : hist) {
99.                 std::cout << std::fixed << std::setprecision(1) << std::setw
100.                         << p.first << ' ' << std::string(p.second / 200, '*'
101.         }
102.
103. return 0;
104. }
105.
106.
107.
108. static void test(const double m, const double s, const int samples)
109. {
110. using namespace std;
111.
112.
```

```
113.
114. mt19937 gen(1701);
115.
116.         normal_distribution<> distr(m, s);
117.
118.         cout << endl;
119.         cout << "min() == " << distr.min() << endl;
120.         cout << "max() == " << distr.max() << endl;
121.         cout << "m() == " << fixed << setw(11) << setprecision(10) << distr.m
122.         cout << "s() == " << fixed << setw(11) << setprecision(10) << distr.s
123.
124.
125.         map<double, int> histogram;
126. for (int i = 0; i < samples; ++i) {
127.                 ++histogram[distr(gen)];
128.         }
129.
130.
131.         cout << "Distribution for " << samples << " samples:" << endl;
132. int counter = 0;
133. for (const auto& elem : histogram) {
134.                 cout << fixed << setw(11) << ++counter << ": "
135.                         << setw(14) << setprecision(10) << elem.first << end
136.         }
137.         cout << endl;
138. }
139.
140. int test_normal_distribution_3()
141. {
142. using namespace std;
143.
144. double m_dist = 0;
145. double s_dist = 1;
146. int samples = 10;
147.
148.         cout << "Use CTRL-Z to bypass data entry and run using default value
149.         cout << "Enter a floating point value for the 'mean' distribution pa
150.
```

```
151.         cout << "Enter a floating point value for the 'stddev' distribution ]
152.
153.         cout << "Enter an integer value for the sample count: ";
154.
155.
156. test(m_dist, s_dist, samples);
157.
158. return 0;
159. }
```

GitHub： https://github.com/fengbingchun/Messy_Test