实践方法论

朱明超

Email: deityrayleigh@gmail.com

Github: github.com/MingchaoZhu/DeepLearning

1 实践方法论

到目前为止，我们一直在谈论很多内容，涉及深度学习的理论方面。但是，理论与实际可行之间还存在很大差距。人们可能需要做出各种选择，包括收集哪种类型的数据，在哪里找到该数据，是否应该收集更多数据，更改模型复杂性，更改 (添加/删除) 正则化，改进优化，调试软件实现等等。建议的实用设计过程如下：

1. 确定目标，决定使用什么样的度量指标 (即用一个单一的数字指标来评估你的模型)——这代表了最终目标。这个度量指标的选择取决于该系统旨在解决的问题。
2. 尽快建立**端到端的工作流程**，包括评估所需指标。也就是说，我们在设计一个系统时，要考虑 “数据”、“模型” 和 “分析” 三个模块的实现。要尽快保证系统可以正确接受输入并预处理 (“数据” 模块)，系统可以以正确的格式生成输出 (“模型” 模块)，系统可以计算度量指标并可视化结果 (“分析” 模块)。于是，我们最早在设计 “模型” 模块时，可以先用一个非常简单的模型。接下来，我们就可以只专注于改进 “模型” 模块中的模型，然后就可以立即获得最终结果，并检查该改进是否优化了度量指标。
3. 很好地对系统进行检测，以确定性能瓶颈，这需要**诊断哪些部分的性能比预期的差**，并了解导致性能不佳的原因——过拟合、欠拟合、建模问题、数据问题、软件实现错误等等。
4. 根据上述诊断，可以通过**添加更多数据、增加模型的容量、调整超参数或通过更好的标注来改善数据质量等等来不断地改进算法**。

为此，本章讨论的内容包括如下：

- 性能度量指标
- 默认的基准模型
- 确定是否收集更多数据
- 选择超参数
- 调试策略

2 性能度量指标

如上所述，决定使用哪种错误指标非常重要，因为这最终将指导你如何取得进展。它应该足以代表你要实现的最终目标。例如，我们考虑一个二分类问题–良/恶性乳腺癌肿瘤预测。现在，选择什么合理的指标来代表这里的最终目标呢?

2.1 错误率与准确性

首先考虑错误率 (`Error rate`) 和准确性 (`Accuracy`)。

- 错误率：分类错误的样本数占样本总数的比例 err。
- 准确性：分类正确的样本数占样本总数的比例 $acc = 1 - err$。

现在，我们考虑一种情况，如果我们有 1000 个样本，其中 995 个是良性的，只有 5 个人是恶性的。假设我们只关注预测的错误率与准确性，那么如何设计好的模型? 可以设计模型无论是什么样本输入，全部输出良性。此时的准确性是 99.5%，是不是很满意呢? 但这种模型根本没有用。所以说对于数据的类别存在不平衡的情况，不能只看错误率/准确性。

这便引出了查准率、查全率、ROC、AUC。

2.2 查准率、查全率与 $\mathbf{F}_1$ 值

2.2.1 混淆矩阵

首先定义**混淆矩阵** (`Confusion matrix`)：

	Positive Prediction	Negative Prediction
Positive Class	TP	FN
Negative Class	FP	TN

其中：

- TP (`True Positive`)：表示将正样本预测为正例的数目。即真实结果为 1，预测结果也为 1。
- TN (`True Negative`)：表示将负样本预测为负例的数目。即真实结果为 0，预测结果也为 0。
- FP (`False Positive`)：表示将负样本预测为正例的数目。即真实结果为 0，预测结果为 1。
- FN (`False Negative`)：表示将正样本预测为负例的数目。即真实结果为 1，预测结果为 0。

混淆矩阵的示例如图 1 所示，考虑 8 个样本的真实结果和预测结果，根据定义我们可以得到其混淆矩阵。

Class	1	1	1	1	0	0	0	0
Prediction	1	0	1	1	0	1	1	0

	Positive Prediction (1)	Negative Prediction (0)
Positive Class (1)	3	2
Negative Class (0)	1	2

图 1. 混淆矩阵示例。上方显示真实类别与预测类别，下方显示混淆矩阵。

通过混淆矩阵，我们可以获得前面描述的错误率和准确性的数学定义：

$$
\begin{aligned}
acc &= \frac{\mathrm{TP}+\mathrm{TN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}} \\
err &= \frac{\mathrm{FP}+\mathrm{FN}}{\mathrm{TP}+\mathrm{TN}+\mathrm{FP}+\mathrm{FN}}
\end{aligned}
\tag{1}
$$

2.2.2 查准率和查全率的定义与关联

现在我们定义查准率和查全率：

- 查准率 (`Precision`)：也叫精度，简记为 P 或 PPV，表示预测为正例的样本中（$\mathrm{TP}+\mathrm{FP}$）有多少是真正的正样本（TP）。

$$
\mathrm{P} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FP}} \tag{2}
$$

- 查全率 (`Recall`)：也叫召回率，简记为 R 或 TPR，表示在实际真正的正样本中（$\mathrm{TP}+\mathrm{FN}$），预测为正例的样本数（TP）所占的比例。

$$
\mathrm{R} = \frac{\mathrm{TP}}{\mathrm{TP}+\mathrm{FN}} \tag{3}
$$

为什么要引入这两个指标?

查准率表示**宁愿漏掉，不可错杀**。在识别垃圾邮件中偏向这种思路，因为我们不希望正常邮件 (对应为负样本，通常将占多数的类别视为负类) 被误杀，这样会造成严重的困扰。

查全率表示**宁愿错杀，不可漏掉**。在金融风控领域偏向这种思路，我们希望系统能够筛选出所有有风险的行为或用户 (对应为正样本)，然后交给人工鉴别，漏掉一个可能造成灾难性后果。

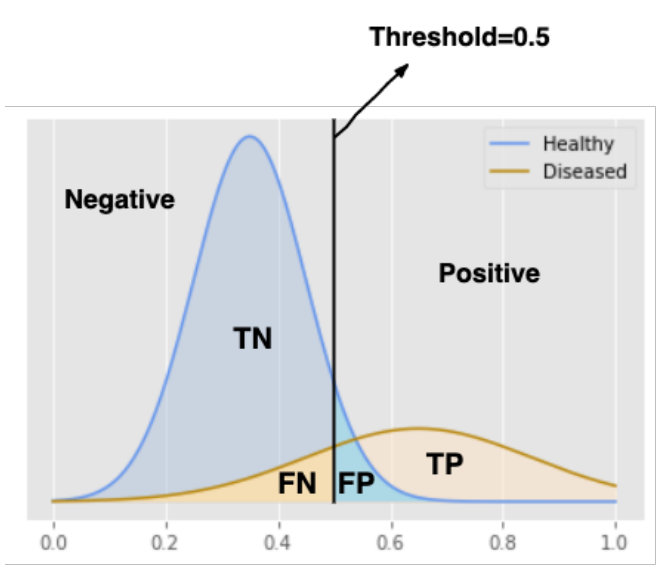


图 2. 肿瘤恶性/良性分类器示例。暗黄色高斯曲线表示真正的恶性肿瘤分布，淡蓝色高斯曲线表示真正的良性肿瘤分布。Positive 区域是大于阈值即肿瘤分类器预测为正例的样本。Negative 区域是肿瘤分类器预测为负例的样本。

查准率代表实际检测结果的百分比，而查全率则代表成功检测到的真实事件的比例。如图 2 所示，我们构造一个肿瘤恶性/良性分类器 (并非基于真实数据绘制)。暗黄色高斯曲线表示真正的恶性肿瘤分布，对应所有正例样本数 (暗黄色曲线下方面积) 为 $\mathrm{TP}+\mathrm{FN}$。淡蓝色高斯曲线表示真正的良性肿瘤分布，对应所有负例样本数 (淡蓝色曲线下方面积) 为 $\mathrm{FP}+\mathrm{TN}$。Positive 区域是大于阈值即肿瘤分类器预测为正例的样本 $\mathrm{TP}+\mathrm{FP}$。

Negative 区域是肿瘤分类器预测为负例的样本 TN + FN。于是，查准率 P 表示在 Positive 中，肿瘤分类器预测对的比重，所以越大越好。查全率 R 表示在暗黄色曲线中，落在 Positive 区域的比重，所以越大越好。

现在，回到最初的问题，考虑如果模型指出所有的肿瘤预测都不是恶性的，那么召回率为 0 (此时一个真正的正样本都没预测到)。查准率和查全率通常是一对矛盾的度量。例如，卖瓜的时候，如果希望查准率高，那就选择那些最有把握的瓜，但这样有些好瓜就不会被选到，查全率并不高；但如果希望查全率高，那就将有把握的和模棱两可的瓜都选上，这样虽然查全率高，但查准率又下降了。

2.2.3 F_1 值

而很多时候，我们实际上只希望有一个单一的指标来判断，而不是在两个指标之间进行权衡。F_1 值是 “查准率和查全率” 的调和平均值，它是一个广为接受的指标：

$$F_1 = \frac{2PR}{P + R} \tag{4}$$

但是，F_1 值会在查准率和查全率上给予同等的权重。在某些情况下，你可能想偏重一个，因此我们获得了更一般的 F_β 值：

$$F_\beta = (1 + \beta^2)\frac{PR}{\beta^2 P + R} \tag{5}$$

其中 β 用于调整权重，当 $\beta = 1$ 时两者权重相同，即为 F_1 值。如果认为查准率更重要，则减小 β；若认为查全率更重要，则增大 β。

2.3 PR 曲线

但如果你确实想讨论查准率/查全率的关系时，PR 曲线 (`Precision-Recall Curve`) 可以提供帮助。PR 曲线是针对不同阈值的查准率 P (y 轴) 和查全率 R (x 轴) 的图。算法对样本进行分类时都会有置信度，即表示该样本是正例的概率，比如 99% 的概率认为样本 A 是正例，或者 15% 的概率认为样本 B 是正例。通过选择合适的阈值 (比如 50%)，就对样本进行划分，概率大于阈值 (50%) 的就认为是正例，小于阈值 (50%) 的就是负例。

因此，我们考虑选择不同的阈值，并计算不同阈值情况下的查准率和查全率又是如何。具体做法是通过置信度对所有样本进行排序，再逐个样本的选择阈值 (以该样本的置信度作为阈值)，在该样本之前的都视作正例，该样本之后的都视作负例。将每一个样本分别作为划分阈值，并计算对应的查准率和查全率，就可以绘制 PR 曲线。

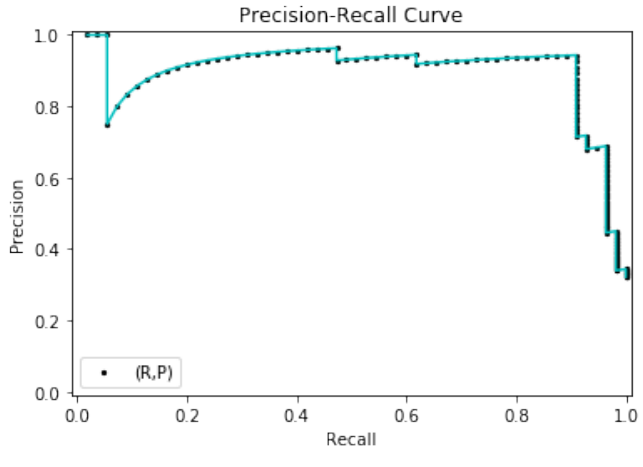


图 3. PR 曲线示意图。实现代码见后文代码实现部分。

PR 曲线示意图如图 3 所示 (实现代码见后文)。因为是经过排序后逐个样本作为阈值划分点，可以知道随着划分点的移动，被视作正例的样本逐渐增加，于是查全率是单调递增的 (分母考虑所有的，所以不会变化)。而查准率并非递减，越来越多的样本被视作正例，则 TP 和 FP 都可能增加，因此查准率可能会振荡，但整体趋势会下降。一个好的模型表现是给正样本高置信度，负样本低置信度。在此情况下，我们陆续选择置信度作为阈值，好的模型的表现是经过对置信度排序后排在前列的都是正样本，则对应于查全率由 0 到 1 的过程中，查准率一直等于 1 或接近 1。

PR 曲线的注意点：

1. 由于在最后所有的样本都会被视作正例，因此 FN=0，所以 R=TP/(TP+FN)=1。同时，FP= 所有的负样本数，因此 P=TP/(TP+FP)= 正样本占所有样本的比例，可以知道除非样本中负样本数很多，否则 P 不会为 0。如果看到 PR 曲线的终点接近 (1,0) 点，这可能是因为样本中负样本远远多于正样本。
2. 具有完美表现的模型会绘制坐标 (1,1) 的点，PR 曲线表现为一条 y 轴值为 1 的水平线段，外加在 x 轴值为 1 处的垂直线段 (线段终点由样本中正样本所占比例决定)，这表示经过置信度排序后正样本都排在了前列。一个模型的表现可以由一条向坐标 (1,1) 弯曲的曲线表示。无技能的分类器 (即该分类器没有学习) 将是图上的一条水平线，其查准率与数据集中正样本数量成比例，对于一个平衡的数据集而言将是 0.5。

PR 曲线集中在少数类上 (这里我们将少数类作为正样本)，使其成为不平衡二元分类模型的有效诊断。同时，当你需要考虑适合你业务需求的阈值时 (阈值可能非 0.5)，使用 PR 曲线也是不错的选择。

2.4 ROC 曲线与 AUC 值

对于二分类的度量，除了上面的查准率 P、查全率 R 以及引申的 PR 曲线，我们还可以通过以下度量方法。首先，需要定义真正例率 (`True Positive Rate`) 和假正例率 (`False Positive Rate`)：

- 真正例率：简记为 TPR，表示当前被预测为正例的样本中，真正的正样本（TP）占实际所有的正样本（TP+FN）的比例。其实也就是查全率或召回率 (表示召回的正样本比例)。

$$\mathrm{TPR} = \frac{\mathrm{TP}}{\mathrm{TP} + \mathrm{FN}} \tag{6}$$

- 假正例率：简记为 FPR，表示当前被错误预测为正例的样本中，真正的负样本（FP）占实际所有的负样本（FP+TN）的比例。

$$\mathrm{FPR} = \frac{\mathrm{FP}}{\mathrm{FP} + \mathrm{TN}} \tag{7}$$

回到图中，真正例率 TPR 意义等同于前面描述的查全率 R。假正例率 FPR 表示淡蓝色曲线中，落在预测正例 Positive 区域的比重，所以越小越好。

2.4.1 ROC 曲线

如果我们以假正例率 FPR 为 x 轴，真正例率 TPR 为 y 轴，并且随着与在 PR 曲线中相同思路的阈值改变，我们将得到 ROC 曲线 (`Receiver Operating Characteristic Curve`)。示意图如图 4 所示 (实现代码见后文)。

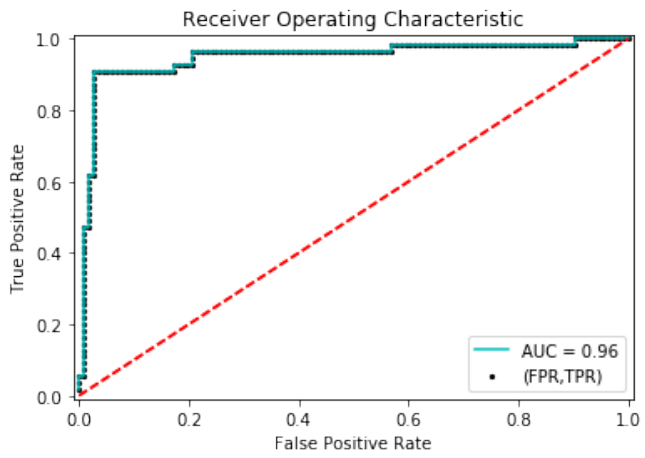


图 4. ROC 曲线示意图。实现代码见后文代码实现部分。

ROC 曲线的注意点：

1. 初始时所有样本均视为负例，此时 TP 和 FP 均为 0，故曲线必然经过 (0,0)。最后所有的样本都会被视作正例，此时 FN 和 TN 均为 0，故曲线必然经过 (1,1)。由于阈值点的移动，TP 和 FP 会不断增加，于是 FPR 和 FPR 都是单调递增的 (分母考虑所有的，所以不会变化)。
2. 可以想象，如果一个模型的性能较好，自然是 FPR 越小、TPR 越大，这样越好。所以，理想情况下的 ROC 曲线经过坐标 (0,1) 的点。如果模型的表现较好，我们自然希望在 FPR 很小的时候 TPR 就比较大 (可以参考图理解)，因此一个模型的表现可以由一条向坐标 (0,1) 弯曲的曲线表示。再考虑无技能的分类器 (即该分类器没有学习)，将是图上的一条 $y = x$ 的直线段，因为 TPR 和 FPR 都明显与被视作正例的数目成比例 (分母是除以各自对应的总数，因此没有学习的情况下都是从 0 到 1 沿相同斜率增加)。

2.4.2 AUC 值的计算方法

尽管 ROC 曲线是一种有用的诊断工具，但根据两个或多个分类器的曲线比较它们可能会具有挑战性。为了得到一个描述曲线的数字，我们可以计算 ROC 曲线下的面积或称 ROC AUC 值。显然，ROC 曲线的左上方程度越多，面积越大，ROC AUC 值越高。

那么如何 计算 AUC 值 呢?

梯形法

采用面积的积分公式，计算曲线下各个小矩形的面积之和。

概率法

现在，我们考虑数据集 D：$(\boldsymbol{x}_1, y_1), \ldots, (\boldsymbol{x}_m, y_m) \in \mathbb{R}^n \times \{0, 1\}$，其中 $\boldsymbol{x}_i$ 是第 i 个 n 维特征的样本。y_i 是第 i 个样本的真实类别 (0 或 1)。

对于一个新的观测样本 $\boldsymbol{x} \in \mathbb{R}^n$，我们通过训练后的分类模型为其得到预测概率 $\hat{p}(\boldsymbol{x})$，表示模型对 $\boldsymbol{x}$ 的标签 $y = 1$ 的置信度。接下来，我们可以选择不同的阈值，然后对每个样本根据置信度和阈值的比较将其分类到类别 0 或类别 1。更进一步，我们计算真正例率和假正例率，并绘制 ROC 曲线。

假设最终的类别 (0 或 1) 是由阈值 $t \in [0, 1]$ 决定，那么真正例率 TPR 可以写作条件概率：

$$T(t) := P[\hat{p}(\boldsymbol{x}) > t \mid \boldsymbol{x} \text{属于类别 } 1] \tag{8}$$

假正例率 FPR 也可以写作：

$$F(t) := P[\hat{p}(\boldsymbol{x}) > t \mid \boldsymbol{x} \text{不属于类别 } 1] \tag{9}$$

简化起见，我们用 $y(\boldsymbol{x}) = 1$ 表示 $\boldsymbol{x}$ 属于类别 1，$y(\boldsymbol{x}) = 0$ 表示 $\boldsymbol{x}$ 不属于类别 1。

ROC 曲线绘制 t 由 1 至 0 时 $T(t)$ 和 $F(t)$ 的关系，于是我们将 T 视作 F 的函数：

$$\begin{aligned}
\text{AUC} &= \int_0^1 T(F_0)\,\mathrm{d}F_0 \\
&= \int_0^1 P(\,\hat{p}(\boldsymbol{x}) > F^{-1}(F_0) \mid y(\boldsymbol{x}) = 1)\,\mathrm{d}F_0 \\
&= \int_1^0 P(\,\hat{p}(\boldsymbol{x}) > F^{-1}(F(t)) \mid y(\boldsymbol{x}) = 1) \cdot \frac{\partial F(t)}{\partial t}\,\mathrm{d}t \\
&= \int_0^1 P(\,\hat{p}(\boldsymbol{x}) > t \mid y(\boldsymbol{x}) = 1) \cdot P(\hat{p}(\boldsymbol{x'}) = t \mid y(\boldsymbol{x'}) = 0)\,\mathrm{d}t \\
&= \int_0^1 P(\,\hat{p}(\boldsymbol{x}) > \hat{p}(\boldsymbol{x'}), \hat{p}(\boldsymbol{x'}) = t \mid y(\boldsymbol{x}) = 1, y(\boldsymbol{x'}) = 0)\,\mathrm{d}t \\
&= P(\,\hat{p}(\boldsymbol{x}) > \hat{p}(\boldsymbol{x'}) \mid y(\boldsymbol{x}) = 1, y(\boldsymbol{x'}) = 0)
\end{aligned}$$

其中，从第三步到第四步是用到累积分布函数 $P[\hat{p}(\boldsymbol{x'}) \leq t \,|\, y(\boldsymbol{x'}) = 0] = 1 - F(t)$ 关于 t 的导数为概率密度函数 $P[\hat{p}(\boldsymbol{x'}) = t \,|\, y(\boldsymbol{x'}) = 0]$。所以，我们可以看出，如果给定一个随机选择的观测样本 $\boldsymbol{x}$ 属于类别 1，以及另一个随机选择的观测样本 $\boldsymbol{x'}$ 属于类别 0，AUC 是分类模型给 $\boldsymbol{x}$ 的打分要高于给负例 $\boldsymbol{x'}$ 的打分的概率，即 $\hat{p}(\boldsymbol{x}) > \hat{p}(\boldsymbol{x'})$ 的条件概率。

这便得到了概率法的做法：随机抽出一对样本 (一个正样本，一个负样本)，然后用训练好的分类模型对这两个样本进行预测，预测得到正样本的概率大于负样本概率的概率。在实现时，我们可以先对预测概率升序排序，然后统计有多少正负样本对满足：正样本预测值 > 负样本预测值, 再除以总的正负样本对的数目。统计的过程为：构造初始的当前负样本数目 $Count_{Neg} = 0$，如果样本 $\boldsymbol{x'}$ 的标签 (真实类别) 为 0，则计入当前的负样本数目 $Count_{Neg}$；如果样本 $\boldsymbol{x}$ 的标签为 1，则满足条件的正负样本对数目加上 $Count_{Neg}$，这是因为我们是对概率升序排序，所以一定满足 $\hat{p}(\boldsymbol{x}) > \hat{p}(\boldsymbol{x'})$，于是正样本 $\boldsymbol{x}$ 和 $Count_{Neg}$ 计数的负样本都是满足条件的。这样做的时间复杂度为 $O(m \log m)$，m 为样本数。

2.5 覆盖

在一些应用中，机器学习系统可能会拒绝做出判断。如果机器学习算法能够估计所作判断的置信度，这就会非常有用，可以帮助机器决定是否要做出判断。如果错误判断会导致严重危害，而且操作人员又可以偶尔地人工接管，那么当机器学习系统认为某个样本或数据无法操作时，最好的办法是由人来做。但是，只有当机器学习系统确实能够大量降低需要人工操作处理的工作时，它才是有用的。这便是覆盖 (Coverage) 的度量指标：

- 覆盖 (Coverage)：表示机器学习系统能够产生响应的比例。

2.6 指标性能的瓶颈

在大多数应用中，即便拥有了无限量的数据，也可能无法实现绝对的零误差，这可能是由于特征的表示能力不足或者系统本质上的随机性。系统可能的最小误差量称为系统的贝叶斯误差。

指标性能的主要瓶颈通常是训练数据有限。现在，如果从 MNIST 之类的标准数据集转入更多实际问题，你就会意识到，获取准确数据要比最初看起来要困难得多，而且在大多数情况下，这并不是免费的。

因此，重要的事情是你需要事先确定好应用程序的实际期望误差率，并用来指导接下来的设计决策。

自定义实现度量指标

[1]:
```
from abc import ABC, abstractmethod
import pandas as pd
import numpy as np
from sklearn.model_selection import train_test_split
from sklearn.preprocessing import StandardScaler
from chapter5 import Sigmoid, LogisticRegression
import matplotlib.pyplot as plt
%matplotlib inline
import itertools
import time
import re
from scipy.stats import norm
```

[2]:
```
def cal_conf_matrix(labels, preds):
    """
    计算混淆矩阵。

    参数说明:
    labels: 样本标签 (真实结果)
    preds: 预测结果
    """
    n_sample = len(labels)
```

```
    result = pd.DataFrame(index=range(0,n_sample),columns=('probability','label'))
    result['label'] = np.array(labels)
    result['probability'] = np.array(preds)
    cm = np.arange(4).reshape(2,2)
    cm[0,0] = len(result[result['label']==1][result['probability']>=0.5]) # TP，注意这里是以 0.5 为阈值
    cm[0,1] = len(result[result['label']==1][result['probability']<0.5])  # FN
    cm[1,0] = len(result[result['label']==0][result['probability']>=0.5]) # FP
    cm[1,1] = len(result[result['label']==0][result['probability']<0.5])  # TN
    return cm
```

[3]:
```
def cal_PRF1(labels, preds):
    """
    计算查准率 P，查全率 R，F1 值。
    """
    cm = cal_conf_matrix(labels, preds)
    P = cm[0,0]/(cm[0,0]+cm[1,0])
    R = cm[0,0]/(cm[0,0]+cm[0,1])
    F1 = 2*P*R/(P+R)
    return P, R, F1
```

[4]:
```
def cal_PRcurve(labels, preds):
    """
    计算 PR 曲线上的值。
    """
    n_sample = len(labels)
    result = pd.DataFrame(index=range(0,n_sample),columns=('probability','label'))
    y_pred[y_pred>=0.5] = 1
    y_pred[y_pred<0.5] = 0
    result['label'] = np.array(labels)
    result['probability'] = np.array(preds)
    result.sort_values('probability',inplace=True,ascending=False)
    PandR = pd.DataFrame(index=range(len(labels)),columns=('P','R'))
    for j in range(len(result)):
        # 以每一个概率为分类的阈值，统计此时正例和反例的数量
        result_j = result.head(n=j+1)
        P = len(result_j[result_j['label']==1])/float(len(result_j))  # 当前实际为正的数量/当前预测为正的数量
        R = len(result_j[result_j['label']==1])/float(len(result[result['label']==1]))  # 当前真正例的数量/实际为正的数量
        PandR.iloc[j] = [P,R]
    return PandR
```

[5]:
```
def cal_ROCcurve(labels, preds):
    """
    计算 ROC 曲线上的值。
    """
    n_sample = len(labels)
    result = pd.DataFrame(index=range(0,n_sample),columns=('probability','label'))
    y_pred[y_pred>=0.5] = 1
    y_pred[y_pred<0.5] = 0
    result['label'] = np.array(labels)
    result['probability'] = np.array(preds)
    # 计算 TPR,FPR
    result.sort_values('probability',inplace=True,ascending=False)
    TPRandFPR=pd.DataFrame(index=range(len(result)),columns=('TPR','FPR'))
    for j in range(len(result)):
        # 以每一个概率为分类的阈值，统计此时正例和反例的数量
        result_j=result.head(n=j+1)
        TPR=len(result_j[result_j['label']==1])/float(len(result[result['label']==1]))  # 当前真正例的数量/实际为正的数量
        FPR=len(result_j[result_j['label']==0])/float(len(result[result['label']==0]))  # 当前假正例的数量/实际为负的数量
        TPRandFPR.iloc[j]=[TPR,FPR]
    return TPRandFPR
```

[6]:
```
def timeit(func):
    """
    装饰器，计算函数执行时间
    """
    def wrapper(*args, **kwargs):
        time_start = time.time()
        result = func(*args, **kwargs)
        time_end = time.time()
        exec_time = time_end - time_start
        print("{function} exec time: {time}s".format(function=func.__name__,time=exec_time))
        return result
    return wrapper

@timeit
def area_auc(labels, preds):
    """
    AUC 值的梯度法计算
    """
    TPRandFPR = cal_ROCcurve(labels, preds)
    # 计算 AUC，计算小矩形的面积之和
    auc = 0.
    prev_x = 0
    for x, y in zip(TPRandFPR.FPR,TPRandFPR.TPR):
        if x != prev_x:
            auc += (x - prev_x) * y
            prev_x = x
    return auc

@timeit
def naive_auc(labels, preds):
    """
    AUC 值的概率法计算
    """
    n_pos = sum(labels)
    n_neg = len(labels) - n_pos
    total_pair = n_pos * n_neg  # 总的正负样本对的数目
    labels_preds = zip(labels, preds)
    labels_preds = sorted(labels_preds,key=lambda x:x[1])  # 对预测概率升序排序
    count_neg = 0  # 统计负样本出现的个数
    satisfied_pair = 0   # 统计满足条件的样本对的个数
    for i in range(len(labels_preds)):
        if labels_preds[i][0] == 1:
            satisfied_pair += count_neg  # 表明在这个正样本下，有哪些负样本满足条件
        else:
            count_neg += 1
    return satisfied_pair / float(total_pair)
```

用自定义的度量指标，乳腺癌数据集测试，第五章描述的逻辑回归用于模型训练

[7]:
```
column_names = ['Sample code number','Clump Thickness',
                'Uniformity of Cell Size','Uniformity of Cell Shape',
                'Marginal Adhesion','Single Epithelial Cell Size',
                'Bare Nuclei','Bland Chromatin','Normal Nucleoli','Mitoses','Class']
data = pd.read_csv('../data/cancer/breast-cancer-wisconsin.data',names=column_names)
data = data.replace(to_replace='?',value=np.nan)  # 非法字符替代
data = data.dropna(how='any')  # 去掉空值，any；出现空值行则删除
print(data.shape)
# 随机采样 25% 的数据用于测试，剩下 75% 用于构建训练集
X_train,X_test,y_train,y_test = train_test_split(data[column_names[1:10]], data[column_names[10]],
                                                 test_size=0.25, random_state=1111)
```

```
# 再查看训练样本的数量和类别分布
print(y_train.value_counts())
# 修改标签为 0 和 1
print(y_train.shape)
y_train[y_train==2] = 0
y_train[y_train==4] = 1
y_test[y_test==2] = 0
y_test[y_test==4] = 1
# 再查看训练样本的数量和类别分布
print(y_train.value_counts())
# 数据标准化预处理，保证每个维度特征均值为 0，方差为 1
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)
```

```
(683, 11)
2    328
4    184
Name: Class, dtype: int64
(512,)
0    328
1    184
Name: Class, dtype: int64
```

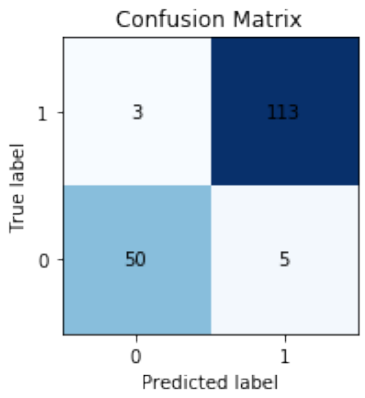
[8]:
```
model = LogisticRegression()
# 使用逻辑回归训练
model.fit(X_train,y_train)
# 预测测试集
y_pred = model.predict(X_test)
```

自定义混淆矩阵测试

[9]:
```
# 计算混淆矩阵
cm = cal_conf_matrix(y_test, y_pred)
print(cm)
# 绘制混淆矩阵
classes = [0,1]
plt.figure(figsize=(3,3))
plt.imshow(cm, interpolation='nearest', cmap=plt.cm.Blues)
plt.title('Confusion Matrix')
tick_marks = np.arange(len(classes))
plt.xlim(-0.5,1.5)
plt.ylim(-0.5,1.5)
plt.xticks(tick_marks, classes)
plt.yticks(tick_marks, classes)
for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
    plt.text(j, i, cm[i, j], horizontalalignment="center", verticalalignment='center')
    plt.tight_layout()
plt.ylabel('True label')
plt.xlabel('Predicted label')
```

```
[[ 50   5]
 [  3 113]]
```

[9]: Text(0.5, 6.800000000000011, 'Predicted label')



用 sklearn 实现混淆矩阵测试

[10]:
```
from sklearn.metrics import confusion_matrix

y_pred_cla = y_pred.copy()
y_pred_cla[y_pred_cla>=0.5] = 1
y_pred_cla[y_pred_cla<0.5] = 0
cm = confusion_matrix(y_test, y_pred_cla)
print(cm)
```

```
[[113   3]
 [  5  50]]
```

自定义 P、R、F_1 值测试

[11]:
```
P, R, F1 = cal_PRF1(y_test, y_pred)
print(P, R, F1)
```

```
0.9433962264150944 0.9090909090909091 0.9259259259259259
```

用 sklearn 实现 P、R、F_1 值测试

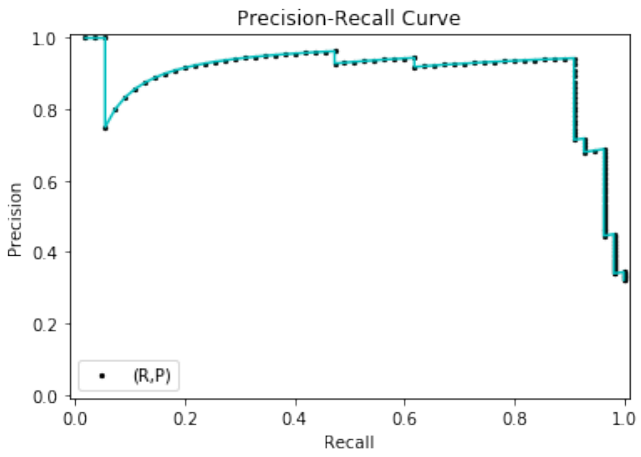
[12]:
```
from sklearn.metrics import precision_score, recall_score, f1_score

y_pred_cla = y_pred.copy()
y_pred_cla[y_pred_cla>=0.5] = 1
y_pred_cla[y_pred_cla<0.5] = 0
print(precision_score(y_test, y_pred_cla))
print(recall_score(y_test, y_pred_cla))
print(f1_score(y_test, y_pred_cla))
```

```
0.9433962264150944
0.9090909090909091
0.9259259259259259
```

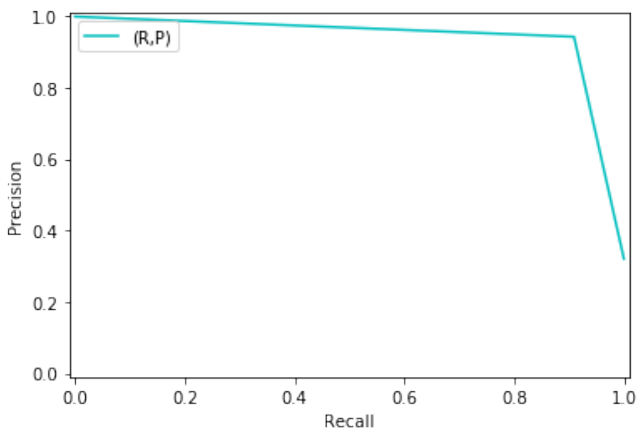
自定义 PR 曲线测试

[13]:
```
# 计算 PR 曲线并绘制
PandR = cal_PRcurve(y_test, y_pred)
plt.scatter(x=PandR['R'],y=PandR['P'],label='(R,P)',color='k',s=5)
plt.plot(PandR['R'], PandR['P'],color='c')
plt.title('Precision-Recall Curve')
plt.xlim([-0.01,1.01])
plt.ylim([-0.01,1.01])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
plt.show()
```



用 sklearn 实现 PR 曲线测试

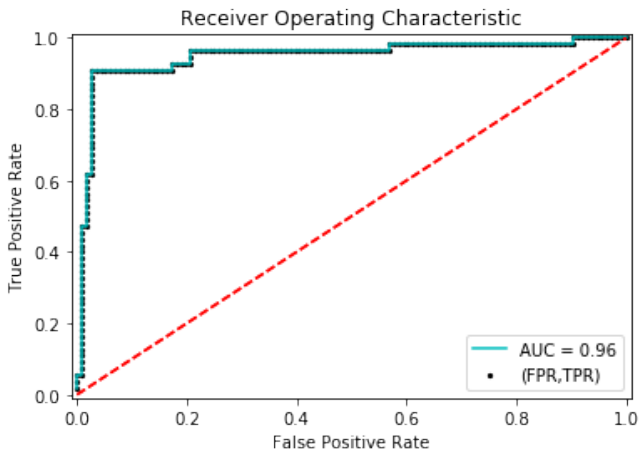
[14]:
```
from sklearn.metrics import precision_recall_curve
precision, recall, _ = precision_recall_curve(y_test, y_pred)
plt.plot(recall, precision, color='c', label='(R,P)')
plt.xlim([-0.01,1.01])
plt.ylim([-0.01,1.01])
plt.xlabel('Recall')
plt.ylabel('Precision')
plt.legend()
plt.show()
```



自定义 ROC 曲线和 AUC 值测试

[15]:
```
# 绘制 ROC 曲线和 AUC 值
area_AUC= area_auc(y_test, y_pred)
naive_AUC = naive_auc(y_test, y_pred)
TPRandFPR = cal_ROCcurve(y_test, y_pred)
plt.scatter(x=TPRandFPR['FPR'],y=TPRandFPR['TPR'],label='(FPR,TPR)',color='k',s=5)
plt.plot(TPRandFPR['FPR'], TPRandFPR['TPR'], 'c',label='AUC = %0.2f'% naive_AUC)
plt.legend(loc='lower right')
plt.title('Receiver Operating Characteristic')
plt.plot([(0,0),(1,1)],'r--')
plt.xlim([-0.01,1.01])
plt.ylim([-0.01,01.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```

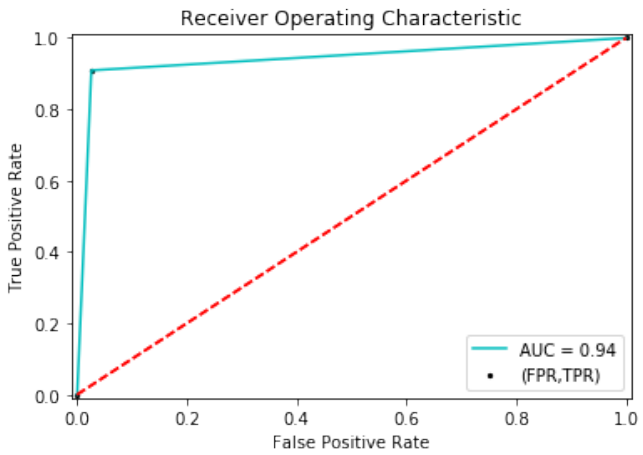
```
area_auc exec time: 0.4615659713745117s
naive_auc exec time: 0.00011801719665527344s
```



用 sklearn 实现 ROC 曲线和 AUC 值测试

[16]:
```
from sklearn.metrics import roc_curve
from sklearn.metrics import roc_auc_score

fpr, tpr, _ = roc_curve(y_test, y_pred)
roc_auc = roc_auc_score(y_test, y_pred)
plt.scatter(x=fpr,y=tpr,label='(FPR,TPR)',color='k',s=5)
plt.plot(fpr, tpr, 'c',label='AUC = %0.2f'% roc_auc)
plt.legend(loc='lower right')
plt.title('Receiver Operating Characteristic')
plt.plot([(0,0),(1,1)],'r--')
plt.xlim([-0.01,1.01])
plt.ylim([-0.01,01.01])
plt.ylabel('True Positive Rate')
plt.xlabel('False Positive Rate')
plt.show()
```



3 默认基准模型

如开始时提到的，尽快建立一个有效的端到端系统非常重要。根据问题的复杂性，我们甚至可能选择从一个非常简单的模型开始，例如逻辑回归。但是，如果你要解决的问题属于 “完全 AI” 的，例如图像分类，语音识别等，那么从深度学习模型入手几乎总是更好。

首先，你要根据数据的结构选择适合的基准模型。如果你的数据由固定大小的向量组成，并且你打算执行监督学习任务，那么可以使用多层感知器。如果你的数据具有固定的拓扑结构，则使用卷积神经网络可能是最好的方法。同样，如果你的数据具有顺序模式，则循环神经网络将是理想的起点。然而，深度学习在不断发展，默认的算法可能会改变。例如，3-4 年前，AlexNet 将是基于图像任务的理想起点。但是，现在 ResNets 是广泛接受的默认选择。

为了训练模型，一个合理的起点是使用 Adam 优化器。除此之外，具有动量和学习率衰减的 SGD 也被广泛使用，其中，学习率呈指数衰减直至一个点，然后每次验证误差稳定时，线性减小 2-10 倍。通常，批标准化可以提供稳定性并允许使用更大的学习率帮助更快地达到收敛，从而提高性能。

随着模型复杂度的增加，由于训练数据有限，最终将变得易于过拟合。因此，建议也为模型添加一些正则化。常见的选择包括损失函数的 L^2 正则化，Dropout，提前终止和批标准化。使用批标准化可以不用 Dropout。(请详见第七章和第八章)

如果你的任务与已有的先前工作的一些其他任务相似，建议你仅从后者复制模型 (以及权重)，然后将其用作任务的初始值。这种训练方式称为迁移学习 (`Transfer Learning`)。例如，在 Kaggle 上的 Dogs Vs Cats 图像分类挑战中，使用 ImageNet 上预训练的包含相似图像的模型作为获得最佳性能的起点，而不是从头开始训练模型。

最后，某些领域 (如自然语言处理 (`NLP`)) 在初始化过程中使用无监督的学习方法会极大地受益。在当前应用于 NLP 的深度学习趋势中，通常将每

个单词表示为词嵌入 (向量)，并且存在诸如 word2vec 和 GLoVe 之类的无监督学习方法来学习这些词嵌入 (`Word Embedding`)。

4 确定是否收集更多数据

很多人容易犯的错误是，他们不断尝试使用不同的算法来提高其模型的性能，但**简单地改善他们拥有的数据或收集更多数据则可能是最好的改进方法**。由于数据是使 AI 方案正常工作的最重要部分，因此我们现在将对此进行更详细的探讨。

那么，你如何决定何时获取更多数据? 首先，如果模型在训练集上的性能很差，则说明它没有充分利用数据中存在的信息，在这种情况下，你需要通过增加层数或增加每层中隐藏单元的数量来增加模型的复杂性。同样，**调整超参数**是要执行的重要步骤。你可能会惊讶于选择正确的超参数对使模型正常工作会产生多么大的影响。例如，学习率是你最重要的也是最需要调整的超参数。为你的问题设置正确的学习率值可以节省大量时间。但是，如果你的模型相当复杂并且对优化进行了仔细的调整，但性能仍未达到所需的水平，则问题可能出在数据的质量上，你必须收集更干净的数据。

为了强调数据在现代深度网络中的重要性，对于那些可能不知道的人来说，深度学习之所以开始受到关注的原因是 ImageNet 竞赛，其中在 2012 年，深度学习模型以显着优势远远超过了以前的最佳模型。ImageNet 由数百万个带标签的图像组成，并且创建类似的大标签数据集是当今诸如对象检测之类的极为复杂的问题已成为解决问题的原因。

训练误差通常会随着数据集大小的增加而增加。这是因为该模型会发现现在很难准确地适合所有数据点。而且，通过增加数据集的大小，由于模型现在将变得更加通用，因此你的验证误差 (`dev`) 或者测试误差将减少。另外一种情况，考虑模型，训练误差低但测试误差高的特定情况称为过拟合，是训练深度模型中最常见的问题之一，在这种情况下，正则化可能会有所帮助。

5 选择超参数

大多数深度学习算法具有许多需要正确选择的超参数。不同的超参数控制模型的不同方面。有些影响内存成本，例如要使用的层数，而另一些影响性能，如 Dropout 的保留概率，学习率，动量等。广泛地，有两种选择这些超参数的方法。第一个是手动选择它们，这需要了解超参数的作用以及它们如何影响训练和泛化。另一种方法是自动选择超参数，这大大降低了复杂性，但以计算能力为代价。现在，我们将更详细地讨论这两种方法：

5.1 手动超参数调整

手动超参数调整需要大量领域知识，并对训练误差，泛化误差，学习理论等有基本的了解。手动超参数调整的主要目的是通过平衡内存和运行时间，实现与任务复杂度匹配的有效容量。影响有效容量的因素是模型的表示能力，表示能力指最小化训练模型的代价函数以及正则化程度的学习算法的能力。

许多超参数以不同的方式影响过拟合 (或欠拟合)。例如，增加某些超参数 (如隐藏单元的数量) 会增加过拟合的可能，而增加其他超参数 (如权重衰减) 会减少过拟合的可能。它们中的一些是离散的，例如隐藏单元的数量，而另一些则可能是二进制的，例如是否使用批标准化。一些超参数具有隐式限制它们的界限，例如权重衰减只能减少容量。因此，如果模型欠拟合，则无法通过改变权重衰减使其过拟合。

如前所述，如果你只能调整一个超参数，请调整学习率。在正确的学习率下，模型的有效容量是最高的，这里学习率既不要不太高也不要太低。将学习率设置得太低会降低训练速度，甚至可能导致算法陷入局部极小值；设置得太高可能会由于剧烈振荡而导致训练不稳定。

如果训练误差高，通常的方法是添加更多的层或更多的隐藏单元以增加容量。如果训练误差低但测试误差高，则需要减小训练误差和测试误差之间的差距，而又不要过多增加训练误差。通常，一个充分大的且经过良好正则化的模型 (例如，通过使用 Dropout，批处理归一化，权重衰减等) 效果最佳。实现低泛化的最终目标的两种主要方法是：向模型添加正则化和增加数据集大小。下表显示了每个超参数如何影响容量：

超参数	容量何时增加	原因	注意事项
隐藏单元数量	增加	增加隐藏单元数量会增加模型的表示能力。	几乎模型每个操作所需的时间和内存代价都会随隐藏单元数量的增加而增加。
学习率	调至最优	不正确的学习速率，不管是太高还是太低都会由于优化失败而导致低有效容量的模型。	
卷积核宽度	增加	增加卷积核宽度会增加模型的参数数量。	较宽的卷积核导致较窄的输出尺寸，除非使用隐式零填充减少此影响，否则会降低模型容量。较宽的卷积核需要更多的内存存储参数，并会增加运行时间，但较窄的输出会降低内存代价。
隐式零填充	增加	在卷积之前隐式添加零能保持较大尺寸的表示。	大多数操作的时间和内存代价会增加。
权重衰减系数	降低	降低权重衰减系数使得模型参数可以自由地变大。	
Dropout 比率	降低	较少地丢弃单元可以更多地让单元彼此 “协力” 来适应训练集。	

5.2 自动超参数优化算法

超参数调整可以看作是优化过程本身，它可以优化目标函数 (例如验证)，有时还受到训练时间、内存限制等约束。因此，我们可以设计超参数优化 (Hyperparameter Optimization) 算法包装学习算法并选择其超参数。不幸的是，这些 HO 算法有它们自己的一组超参数，但是这些超参数通常更容易选择，我们现在将讨论这些 HO 算法：

5.2.1 网格搜索 (Grid Search)

对于网格搜索，首先选择一个你认为适合每个超参数的值范围。然后，为超参数值的每种可能组合训练模型。为简化起见，如果你有 2 个超参数并为每个参数选择一个 N 个值的范围，则需要针对所有可能的 N^2 的组合训练模型。通常，你会根据自己的理解 (或经验) 来设置范围的最大值和最小值，然后通常以对数刻度在两者之间选择一个值。例如，学习率的可能值：{0.1, 0.01, 0.001}，隐藏单位数：{50, 100, 200, 400} 等。

此外，网格搜索在重复执行时效果最佳。例如，如果初始设置的范围是 {0.1, 0, 1}，而性能最好的时候值是 1，则可能是该范围设置错误。应该再次检查较高的范围，例如 {1, 2, 3}。如果此时效果最好的值为 0，则应在 {-0.1, 0, 0.1} 之间进行更精细的搜索。

网格搜索的主要问题是计算成本。如果要调整 m 个超参数，并且每个参数都可以取 N 值，则训练和评估试验的次数将以 $O(N^m)$ 形式增长。

5.2.2 随机搜索 (Random Search)

一种更好、更快的方法是随机搜索。在这种情况下，你可以在值的选择上定义一些分布，例如，对于二进制的用二项分布，对于离散的用多项分布，在对数刻度上用均匀分布 (学习率)：

然后，对于每次运行，根据每个超参数的分布对其值进行随机抽样。事实证明，这比网格搜索更有效。下图对此进行了解释：

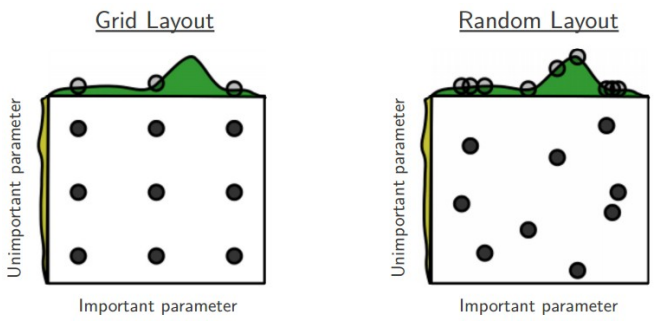


图 5. 左图为网格搜索，右图为随机搜索，横轴表示重要超参数的值，纵轴表示不重要超参数的值。

如图 5 所示，左边为网格搜索，横轴表示重要超参数的值，纵轴表示不重要超参数的值。当有超参数对性能度量没有显著影响时 (纵轴)，随机搜索比网格搜索高效。原因体现在**没有浪费的实验**。例如左图中给定横轴的一个超参数值，网格会对纵轴超参数的 3 个不同值给出相同结果，其实造成了浪费，这样的结果是虽然做了 9 次实验但本质上只有 3 次有用的。但是在右图的随机搜索中，每次超参数通常具有不同值，每次都是独立的探索和实验，这样就实质上做了 9 次实验。所以，更清楚地说，随机搜索比网格搜索更快的主要原因就是它不执行任何浪费的计算。

5.2.3 基于模型的超参数优化 (Model-based Hyperparameter Optimization)

如上所述，超参数调整可以看作是一个优化过程。在简化的设置中，可以针对超参数在验证集上采用一些可微分的误差度量的梯度，再使用梯度下降即可。但是，在大多数实际设置中，这种梯度是不可用的，可能是因为高额的计算代价和存储成本，也可能是压根儿就不可导。为了弥补这一点，你可以对验证误差建模并对该模型执行优化。一种通用方法是建立贝叶斯回归模型以估计验证误差的期望值以及该估计的不确定性。但贝叶斯超参数优化 (Bayesian Hyperparameter Optimization) 仍处于新生阶段，不够可靠。其基本想法是为了找到目标函数 ($f(\theta)$) 的最大值，这里的 θ 代表一组超参数，$f(\theta)$ 是超参数放入黑盒模型 (目标函数) 后的输出 (比如这组超参对应的神经网络的验证集的准确率)，即 $\theta_t = \arg\max f(\theta)$。第 t 次迭代我们生成一个 θ_t，将 $(\theta_t, f(\theta_t))$ 加到我们已有的观测到的数据集合里，然后进行下一次迭代，得到下一个样本 θ_{t+1}。现在问题的核心是在每次迭代里如何选择要观测哪个 θ_t。

在贝叶斯优化中 θ_t 是通过优化另一个函数来选择的：采集函数 (Acquisition Function) α_t。即 $\theta_t = \arg\max \alpha_t(\theta)$。在选下一个超参数点 θ_t 的时候，我们既想要去尝试那些我们之前没有探索过的区域的点 (探索，Exploration)，又想要去根据我们目前已经观测到的所有点的预测选择预测值可能比较大的点 (开发，Exploitation)。为了能很好地平衡两个需求，对于域中里面任意一个点 θ_t，我们既需要预测对应的 $f(\theta_t)$ 的值 (针对开发)，又需要知道对应的 $f(\theta_t)$ 的不确定性程度 (Uncertainty) (针对探索)。我们通常使用代理函数 (Surrogate Function) 同时得到这两个需求，首选方法是高斯过程回归。再通过采集函数基于这两个需求采样下一个点。如此迭代。

那么，这里我们就需要详细说明高斯过程回归，再描述采集函数的方法。为了阐述高斯过程回归，下面会依贝叶斯线性回归、核函数到高斯过程回归的顺序进行描述。

贝叶斯线性回归

在第七章介绍过，线性回归在当噪声服从高斯分布的时候，最小二乘损失最后导出的结果相当于对概率模型应用 MLE。而进一步引入参数的先验时，如果先验分布是高斯分布，那么 MAP 的结果相当于岭回归的正则化，如果先验是拉普拉斯分布，那么相当于 Lasso 的正则化。这两种方案都是点估计方法。我们希望利用贝叶斯方法来求解参数的后验分布。

这里线性回归的模型假设为：

$$\begin{aligned} f(\boldsymbol{x}) &= \boldsymbol{w}^\top \boldsymbol{x} \\ y &= f(\boldsymbol{x}) + \epsilon \\ \epsilon &\sim N(0, \sigma^2) \end{aligned} \tag{10}$$

在贝叶斯方法中，需要解决推断和预测两个问题。

推断

引入高斯先验：

$$p(\boldsymbol{w}) = N(\mathbf{0}, \boldsymbol{\Sigma}_w) \tag{11}$$

对参数的后验分布进行推断：

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) = \frac{p(\boldsymbol{w}, \boldsymbol{y} \mid \boldsymbol{X})}{p(\boldsymbol{y} \mid \boldsymbol{X})} = \frac{p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X})\, p(\boldsymbol{w} \mid \boldsymbol{X})}{\int p(\boldsymbol{y} \mid \boldsymbol{w}, \boldsymbol{X})\, p(\boldsymbol{w} \mid \boldsymbol{X}) d\boldsymbol{w}} \tag{12}$$

分母和参数无关，而 $p(\boldsymbol{w} \mid \boldsymbol{X}) = p(\boldsymbol{w})$，代入先验得到：

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) \propto \prod_{i=1}^{m} p(y^{(i)} \mid \boldsymbol{w}^\top \boldsymbol{x}^{(i)}) N(\mathbf{0}, \boldsymbol{\Sigma}_w) \tag{13}$$

高斯分布取高斯先验的共轭分布依然是高斯分布，于是最终得到的后验分布也会是高斯分布。当然，我们首先看一下这里的第一项：

$$\begin{aligned} \prod_{i=1}^{m} p(y^{(i)} \mid \boldsymbol{w}^\top \boldsymbol{x}^{(i)}) &= \frac{1}{(2\pi)^{m/2}\sigma^m} \exp\left(-\frac{1}{2\sigma^2} \sum_{i=1}^{m} (y^{(i)} - \boldsymbol{w}^\top \boldsymbol{x}^{(i)})\right) \\ &= \frac{1}{(2\pi)^{m/2}\sigma^m} \exp\left(-\frac{1}{2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\top (\sigma^{-2}\boldsymbol{I})(\boldsymbol{Y} - \boldsymbol{X}\boldsymbol{w})\right) \\ &= N(\boldsymbol{X}\boldsymbol{w}, \sigma^{-2}\boldsymbol{I}) \end{aligned} \tag{14}$$

代回原式：

$$p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) \propto \exp\left(-\frac{1}{2}(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w})^\top (\sigma^{-2}\boldsymbol{I})(\boldsymbol{y} - \boldsymbol{X}\boldsymbol{w}) - \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_w^{-1} \boldsymbol{w}\right) \tag{15}$$

假定最后得到的高斯分布记为 $N(\boldsymbol{\mu}, \boldsymbol{\Sigma})$。于是对于上面的分布，采用配方的方式来得到 $\boldsymbol{\mu}$ 和 $\boldsymbol{\Sigma}$。

- 首先分析指数上面的二次项：

$$-\frac{1}{2\sigma^2}\boldsymbol{w}^\top \boldsymbol{X}^\top \boldsymbol{X}\boldsymbol{w} - \frac{1}{2}\boldsymbol{w}^\top \boldsymbol{\Sigma}_w^{-1}\boldsymbol{w} \quad \Rightarrow \quad \boldsymbol{\Sigma}^{-1} = \sigma^{-2}\boldsymbol{X}^\top \boldsymbol{X} + \boldsymbol{\Sigma}_w^{-1} \triangleq \boldsymbol{A} \tag{16}$$

- 首先分析指数上面的一次项：

$$\begin{aligned} \frac{1}{2\sigma^2} 2\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{w} = \sigma^{-2}\boldsymbol{y}^\top \boldsymbol{X}\boldsymbol{w} \quad &\Rightarrow \quad \boldsymbol{\mu}\boldsymbol{\Sigma}^{-1} = \sigma^{-2}\boldsymbol{y}^\top \boldsymbol{X} \\ &\Rightarrow \quad \boldsymbol{\mu} = \sigma^{-2}\boldsymbol{A}^{-1}\boldsymbol{y}^\top \boldsymbol{X} \end{aligned} \tag{17}$$

预测

给定一个 $\boldsymbol{x}^*$，求解 y^*，此时是有 $f(\boldsymbol{x}^*) = \boldsymbol{w}^\top \boldsymbol{x}^* = \boldsymbol{x}^{*\top} \boldsymbol{w}$。代入参数后验，有 $\boldsymbol{x}^{*\top}\boldsymbol{w} = N(\boldsymbol{x}^{*\top}\boldsymbol{\mu}, \boldsymbol{x}^{*\top}\boldsymbol{\Sigma}\boldsymbol{x}^*)$。再考虑噪声，得到 $y^* = f(\boldsymbol{x}^*) + \epsilon$：

$$\begin{aligned} p(y^* \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*) &= \int p(y^* \mid \boldsymbol{w}, \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*) p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*) d\boldsymbol{w} \\ &= \int p(y^* \mid \boldsymbol{w}, \boldsymbol{x}^*) p(\boldsymbol{w} \mid \boldsymbol{X}, \boldsymbol{y}) d\boldsymbol{w} \\ &= N(\boldsymbol{x}^{*\top}\boldsymbol{\mu}, \boldsymbol{x}^{*\top}\boldsymbol{\Sigma}\boldsymbol{x}^* + \sigma^2) \end{aligned} \tag{18}$$

核函数

在第五章介绍支持向量机时我们简单提过核方法，在分类问题中，核方法是将低维空间中的严格不可分的数据转化为高维空间中线性可分的数据(核函数的应用之一)。我们用一个特征转换函数作用到低维空间的数据。而不可分数据在通过特征变换后，往往需要求得变换后的内积。但是将数据从低维变到高维后，我们其实是很难求得变换函数的内积。

现在，我们直接引入内积的变换函数：

$$\forall \boldsymbol{x}, \boldsymbol{x'} \in \mathcal{X}, \exists \phi \in \mathcal{H} : x \to z, \quad k(\boldsymbol{x}, \boldsymbol{x'}) = \phi(\boldsymbol{x})^\top \phi(\boldsymbol{x'}) \tag{19}$$

称 $k(\boldsymbol{x}, \boldsymbol{x'})$ 为核函数，ϕ 为特征转换函数，其中 $\mathcal{H}$ 是 Hilbert 空间 (完备的线性内积空间)。我们直接求解变换后的内积复杂难算，在实际中，通常都是使用核函数求解内积。

例如，考虑 RBF 核 $k(\boldsymbol{x}, \boldsymbol{x'}) = \exp\left(-\frac{(\boldsymbol{x}-\boldsymbol{x'})^2}{2\sigma^2}\right)$，便有：

$$\begin{aligned} \exp\left(-\frac{(\boldsymbol{x} - \boldsymbol{x'})^2}{2\sigma^2}\right) &= \exp\left(-\frac{\boldsymbol{x}^2}{2\sigma^2}\right) \exp\left(-\frac{\boldsymbol{x}\boldsymbol{x'}}{\sigma^2}\right) \exp\left(-\frac{\boldsymbol{x'}^2}{2\sigma^2}\right) \\ &= \exp\left(-\frac{\boldsymbol{x}^2}{2\sigma^2}\right) \sum_{n=0}^{\infty} \frac{\boldsymbol{x}^n \boldsymbol{x'}^n}{\sigma^{2n} n!} \exp\left(-\frac{\boldsymbol{x'}^2}{2\sigma^2}\right) \\ &= \exp\left(-\frac{\boldsymbol{x}^2}{2\sigma^2}\right) \psi^\top(\boldsymbol{x}) \psi(\boldsymbol{x'}) \exp\left(-\frac{\boldsymbol{x'}^2}{2\sigma^2}\right) \\ &= \phi^\top(\boldsymbol{x}) \phi(\boldsymbol{x'}) \end{aligned} \tag{20}$$

这里，我们的复杂的特征转换函数 $\phi(\boldsymbol{x}) = \exp\left(-\frac{\boldsymbol{x}^2}{2\sigma^2}\right)\psi(\boldsymbol{x})$。

我们常说的核函数其实是正定核函数。正定核函数要求核函数满足两个条件：

- 对称性，即 $k(\boldsymbol{x}, \boldsymbol{x'}) = k(\boldsymbol{x'}, \boldsymbol{x})$。
- 正定性，即 $\forall m, \boldsymbol{x}^{(1)}, ..., \boldsymbol{x}^{(m)} \in \mathcal{X}$，对应的 Gram 矩阵 $K = [k(\boldsymbol{x}^{(i)}, \boldsymbol{x}^{(j)})]$ 是半正定的。

高斯过程回归 (核贝叶斯线性回归)

贝叶斯线性回归可以通过加入核函数的方法来解决非线性函数的问题，将 $f(\boldsymbol{x}) = \boldsymbol{x}^\top \boldsymbol{w}$ 这个函数变为 $f(\boldsymbol{x}) = \phi(\boldsymbol{x})^\top \boldsymbol{w}$，变换到更高维的空间，有：

$$f(\boldsymbol{x}^*) = N(\phi(\boldsymbol{x}^*)^\top \sigma^{-2} \boldsymbol{A}^{-1} \boldsymbol{\Phi}^\top \boldsymbol{y}, \phi(\boldsymbol{x}^*)^\top \boldsymbol{A}^{-1} \phi(\boldsymbol{x}^*)) \tag{21}$$

其中 $\boldsymbol{A} = \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \boldsymbol{\Sigma}_w^{-1}$，$\boldsymbol{\Phi} = (\phi(\boldsymbol{x}^1), \phi(\boldsymbol{x}^2)...\phi(\boldsymbol{x}^m))^\top$。为了求解 $\boldsymbol{A}^{-1}$，可以利用 Woodbury Formula，即：

$$(A + UCV)^{-1} = A^{-1} + A^{-1}U(C^{-1} + VA^{-1}U)^{-1}VA^{-1} \tag{22}$$

令 $A = \boldsymbol{\Sigma}_w^{-1}, C = \sigma^{-2}\boldsymbol{I}$，可以得到协方差矩阵 $\boldsymbol{\Sigma}$：

$$\boldsymbol{A}^{-1} = \boldsymbol{\Sigma}_w - \boldsymbol{\Sigma}_w\boldsymbol{\Phi}(\sigma^2\boldsymbol{I} + \boldsymbol{\Phi}^\top\boldsymbol{\Sigma}_w\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}_w \tag{23}$$

另一方面，我们可以对 $\boldsymbol{A} = \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \boldsymbol{\Sigma}_w^{-1}$ 两边变换 (同时左乘/右乘矩阵)，得到均值表达式 $\boldsymbol{\mu}$：

$$\begin{aligned}
& \boldsymbol{A} = \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi} + \boldsymbol{\Sigma}_w^{-1} \\
\Rightarrow \quad & \boldsymbol{A}\boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top = \sigma^{-2}\boldsymbol{\Phi}^\top\boldsymbol{\Phi}\boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top + \boldsymbol{\Phi}^\top = \sigma^{-2}\boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top + \sigma^2\boldsymbol{I}) \\
\Rightarrow \quad & \boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top = \sigma^{-2}\boldsymbol{A}^{-1}\boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top + \sigma^2\boldsymbol{I}) \\
\Rightarrow \quad & \sigma^{-2}\boldsymbol{A}^{-1}\boldsymbol{\Phi}^\top = \boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top(\boldsymbol{\Phi}\boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top + \sigma^2\boldsymbol{I})^{-1} \\
\Rightarrow \quad & \phi(\boldsymbol{x}^*)^\top\sigma^{-2}\boldsymbol{A}^{-1}\boldsymbol{\Phi}^\top\boldsymbol{y} = \phi(\boldsymbol{x}^*)^\top\boldsymbol{\Sigma}_w^\top\boldsymbol{\Phi}(\boldsymbol{\Phi}\boldsymbol{\Sigma}_w\boldsymbol{\Phi}^\top + \sigma^2\boldsymbol{I})^{-1}\boldsymbol{y}
\end{aligned} \tag{24}$$

我们同时将 $\boldsymbol{A}^{-1}$ 代回预测过程：

$$\phi(\boldsymbol{x}^*)^\top\boldsymbol{A}^{-1}\phi(\boldsymbol{x}^*) = \phi(\boldsymbol{x}^*)^\top\boldsymbol{\Sigma}_w\phi(\boldsymbol{x}^*) - \phi(\boldsymbol{x}^*)^\top\boldsymbol{\Sigma}_w\boldsymbol{\Phi}(\sigma^2\boldsymbol{I} + \boldsymbol{\Phi}^\top\boldsymbol{\Sigma}_w\boldsymbol{\Phi})^{-1}\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}_w\phi(\boldsymbol{x}^*) \tag{25}$$

我们可以看到，在均值向量和协方差中，由这四项构成：$\phi(\boldsymbol{x}^*)^\top\boldsymbol{\Sigma}_w^\top\boldsymbol{\Phi}$，$\phi(\boldsymbol{x}^*)^\top\boldsymbol{\Sigma}_w\phi(\boldsymbol{x}^*)$，$\phi(\boldsymbol{x}^*)^\top\boldsymbol{\Sigma}_w\boldsymbol{\Phi}$，$\boldsymbol{\Phi}^\top\boldsymbol{\Sigma}_w\phi(\boldsymbol{x}^*)$。将 $\boldsymbol{\Phi}$ 的表达式代入展开，那么它们是有着通式：$k(\boldsymbol{x}, \boldsymbol{x'}) = \phi(\boldsymbol{x})^\top\boldsymbol{\Sigma}_w\phi(\boldsymbol{x'}) = \sqrt{\boldsymbol{\Sigma}_w}\phi(\boldsymbol{x}) \cdot \sqrt{\boldsymbol{\Sigma}_w}\phi(\boldsymbol{x'})$。其中由于 $\boldsymbol{\Sigma}_w$ 是正定对称的方差矩阵，所以，这是一个核函数。

$$\begin{aligned}
\phi(\boldsymbol{x}^*)^\top\boldsymbol{\mu} &= k(\boldsymbol{x}^*, \boldsymbol{X})[k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\boldsymbol{I}]^{-1}\boldsymbol{y} \\
\phi(\boldsymbol{x}^*)^\top\boldsymbol{\Sigma}\phi(\boldsymbol{x}^*) &= k(\boldsymbol{x}^*, \boldsymbol{x}^*) - k(\boldsymbol{x}^*, \boldsymbol{X})[k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\boldsymbol{I}]^{-1}k(\boldsymbol{X}, \boldsymbol{x}^*) + \sigma^2\boldsymbol{I}
\end{aligned} \tag{26}$$

这样我们就可以和贝叶斯线性回归中进行一样的预测步骤。

函数空间的角度

现在，我们再换个角度看高斯过程回归。在刚刚的视角中，我们是基于概率分布的权重。那现在，我们上升一下，直接到函数级别。回顾一下最开始的函数定义 (式 10)，那么预测就可以写作：

$$p(y^* \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*) = \int_f p(y^* \mid f, \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*)p(f \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{x}^*)df \tag{27}$$

这里可以对比一下 (式 18)。就数据集来说，取 $f(\boldsymbol{X}) \sim N(\mu(\boldsymbol{X}), k(\boldsymbol{X}, \boldsymbol{X}))$，$\boldsymbol{y} = f(\boldsymbol{X}) + \epsilon \sim N(\mu(\boldsymbol{X}), k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\boldsymbol{I})$。这里体现核函数另一个作用，平滑。以 RBF 核为例，当 $\boldsymbol{x}^* = \boldsymbol{x}$ 时，核函数值最大；当两个输入样本变得越来越远时，曲线逐渐呈平滑下降趋势。为了实现平滑度，我们会希望这两个样本的协方差就等于核函数。

现在预测任务的目的是给定一个新数据样本序列 $\boldsymbol{X}^*$，得到对应的预测 $\boldsymbol{y}^* = f(\boldsymbol{X}^*) + \epsilon$。然后，我们便可以写出：

$$\begin{pmatrix} \boldsymbol{y} \\ f(\boldsymbol{X}^*) \end{pmatrix} \sim N\left(\begin{pmatrix} \mu(\boldsymbol{X}) \\ \mu(\boldsymbol{X}^*) \end{pmatrix}, \begin{pmatrix} k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2\boldsymbol{I} & k(\boldsymbol{X}, \boldsymbol{X}^*) \\ k(\boldsymbol{X}^*, \boldsymbol{X}) & k(\boldsymbol{X}^*, \boldsymbol{X}^*) \end{pmatrix}\right) \tag{28}$$

这里回顾一下高斯分布：

$$\begin{pmatrix} x_a \\ x_b \end{pmatrix} \sim N\left(\begin{pmatrix} \mu_a \\ \mu_b \end{pmatrix}, \begin{pmatrix} \Sigma_{aa} & \Sigma_{ab} \\ \Sigma_{ba} & \Sigma_{bb} \end{pmatrix}\right) \tag{29}$$

则会有：

$$\begin{aligned}
& x_b \mid x_a \sim N(\mu_{b|a}, \Sigma_{b|a}) \\
& \mu_{b|a} = \Sigma_{ba}\Sigma_{aa}^{-1}(x_a - \mu_a) + \mu_b \\
& \Sigma_{b|a} = \Sigma_{bb} - \Sigma_{ba}\Sigma_{aa}^{-1}\Sigma_{ab}
\end{aligned} \tag{30}$$

于是，我们可以直接写出：

$$\begin{aligned} p(f(\boldsymbol{X}^*) \mid \boldsymbol{X}, \boldsymbol{y}, \boldsymbol{X}^*) &= p(f(\boldsymbol{X}^*) \mid \boldsymbol{y}) \\ &= N\left(k(\boldsymbol{X}^*, \boldsymbol{X})[k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}]^{-1}(\boldsymbol{y} - \mu(\boldsymbol{X})) + \mu(\boldsymbol{X}^*), k(\boldsymbol{X}^*, \boldsymbol{X}^*) - k(\boldsymbol{X}^*, \boldsymbol{X})[k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}]^{-1} k(\boldsymbol{X}, \boldsymbol{X}^*)\right) \end{aligned} \tag{31}$$

我们通常会设均值为 $\mu(\boldsymbol{X}) = \mu(\boldsymbol{X}^*) = 0$：

$$p(f(\boldsymbol{X}^*) \mid \boldsymbol{y}) = N\left(k(\boldsymbol{X}^*, \boldsymbol{X})[k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}]^{-1} \boldsymbol{y}, k(\boldsymbol{X}^*, \boldsymbol{X}^*) - k(\boldsymbol{X}^*, \boldsymbol{X})[k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}]^{-1} k(\boldsymbol{X}, \boldsymbol{X}^*)\right) \tag{32}$$

如果再有 $\boldsymbol{y}^* = f(\boldsymbol{X}^*) + \epsilon$：

$$p(\boldsymbol{y}^* \mid \boldsymbol{y}) = N\left(k(\boldsymbol{X}^*, \boldsymbol{X})[k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}]^{-1} \boldsymbol{y}, k(\boldsymbol{X}^*, \boldsymbol{X}^*) - k(\boldsymbol{X}^*, \boldsymbol{X})[k(\boldsymbol{X}, \boldsymbol{X}) + \sigma^2 \boldsymbol{I}]^{-1} k(\boldsymbol{X}, \boldsymbol{X}^*) + \sigma^2 \boldsymbol{I}\right) \tag{33}$$

于是，我们同样可以做到预测。但可以看到，函数空间的角度更加简单易于求解。那么我们看一下高斯进程回归的基本步骤：

- 选择适当的均值函数 μ 和核函数 k，以及噪声 σ。其中核函数的选择尤其重要，根据不同的应用选择不同的核，一般选法是 RBF 高斯核。
- 对于已有的训练样本 D，计算核矩阵 $K = k(\boldsymbol{X}, \boldsymbol{X})$。再考虑待预测的样本 $\boldsymbol{x}^*$，计算 $K_* = k(\boldsymbol{x}^*, \boldsymbol{X})$ 和 $K_{**} = k(\boldsymbol{x}^*, \boldsymbol{x}^*)$。
- 计算出 $\boldsymbol{\mu}$ 和 $\boldsymbol{\Sigma}$，代入预测公式 (高斯函数) 中，得到预测值的均值、标准差、置信区间。通常，我们可以将均值就作为预测结果。

最后，我们再考虑一个问题，同样是已有一些样本点再对新样本做出预测，这和我们在第五章讨论的最小二乘回归有什么区别？最小二乘回归最后对新样本点 $\boldsymbol{x}^*$ 的预测 $f(\boldsymbol{x}^*)$ 是一个点估计，而在这里的 $f(\boldsymbol{x}^*)$ 其实是个后验概率分布。既然得到了分布，那么我们同样可以得到预测值的点估计 (均值)，但还可以知道这个估计有多少信心在里面。

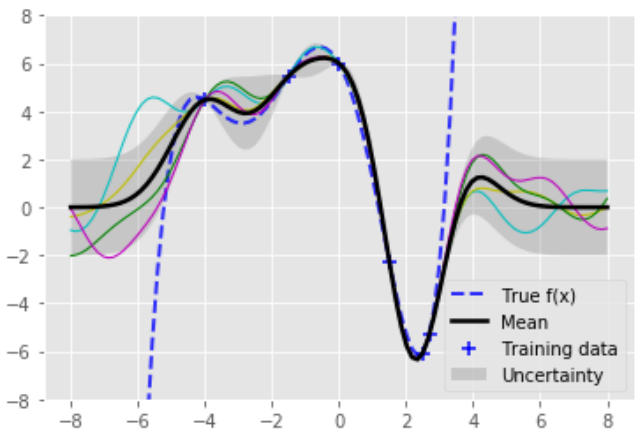


图 6. 高斯过程回归用于指导采样。已知点为图中蓝色点，真实曲线为图中蓝色虚线。对其采样 4 次，每次采样构造 100 个样本的测试集，基于“样本-预测” 可以绘制一条曲线。黑色曲线为进行多次采样平均后的曲线。

其另一个作用可以指导采样。如图 6 所示 (实现代码见下文)，训练集 D 中就 6 个点 (蓝色点)，真实曲线为图中蓝色虚线。我们可以构造一个 100 个样本的测试集。于是，我们便可以得到后验分布，我们从后验分布中随机采样得到每个 $\boldsymbol{x}^*$ 的预测结果 (视作标签)。再根据这些样本 (训练集 + 测试集) 以及它们的标签拟合一条曲线。在图中我们随机采样 4 次，得到 4 个拟合函数。如果是采样很多次，再取所有的拟合函数的平均值就得到均值曲线 (图中黑线)。含有已知数据 (训练集数据) 的地方，这些函数都很接近 (方差很低)；而没有数据的时候，变化范围就比较大 (灰色区域表示均值上下 2 个标准差)。这样，我们可以在预测值和方差之间折中，选择下一个采样点。

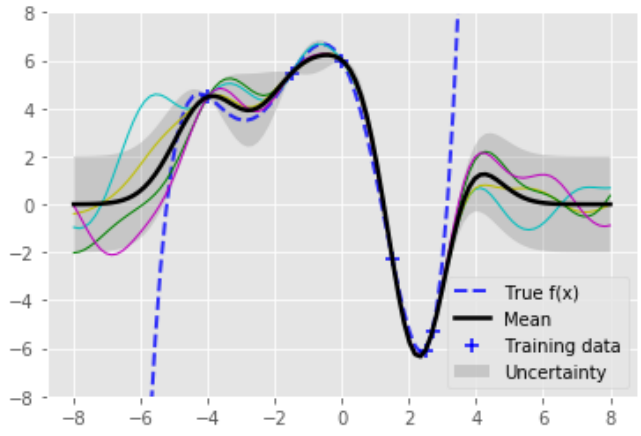
[18]:
```python
def k(xs, xt, sigma=1):
    """
    协方差函数。
    """
    dx = np.expand_dims(xs, 1) - np.expand_dims(xt, 0)
    return np.exp(-(dx**2) / (2*sigma**2))

def m(x):
    """
    均值函数。
    """
    return np.zeros_like(x)

def f(x):
    """
    目标函数。
    """
    coefs = [6, -2.5, -2.4, -0.1, 0.2, 0.03]
    y = 0
    for i, coef in enumerate(coefs):
            y += coef * (x ** i)
```

```
    return y

x = np.array([-4, -1.5, 0, 1.5, 2.5, 2.7])
y = f(x)
x_star = np.linspace(-8, 8, 100)
K = k(x, x)
K_star = k(x, x_star)
K_star_star = k(x_star, x_star)
mu = m(x_star) + K_star.T@np.linalg.pinv(K)@(y-m(y))
Sigma = K_star_star - K_star.T@np.linalg.pinv(K)@K_star
y_true = f(x_star)
plt.style.use('ggplot')
plt.plot(x_star, y_true, linewidth=2, color='b', alpha=0.8,
         linestyle='dashed', label='True f(x)')
plt.scatter(x, y, s=70, c='b', marker='+', label='Training data')
stds = np.sqrt(Sigma.diagonal())
up_lower = mu + 2*stds
plt.fill_between(x_star, mu+2*stds, mu-2*stds, facecolor='grey', alpha=0.3,
                 label='Uncertainty')
Colors = ['c', 'y', 'g', 'm']
for color in Colors:
    y_star = np.random.multivariate_normal(mu, Sigma)
    plt.plot(x_star, y_star, linewidth=1, color=color)
plt.ylim(-8, 8)
plt.plot(x_star, mu, linewidth=2.5, color='k', label='Mean')
plt.legend()
plt.show()
```



采集函数

前面我们通过高斯过程回归得到均值 $\boldsymbol{\mu}$ 和协方差 $\boldsymbol{\Sigma}$，这两个可以分别理解为用来做开发和探索的信息。然后基于这两个信息我们可以设计不同的采集函数。我们这里看一个简单的方法 Thompson Sampling。

Thompson Sampling (TS)

第一步先从当前的高斯过程后验里面采样得到一个函数 (例如在上图中从后验分布中采样了 4 个函数)，不妨记为 $\hat{f}_t$。

第二步我们要新生成的点就是：

$$\boldsymbol{x}^* = \arg\max \hat{f}_t(\boldsymbol{x}^*) \tag{34}$$

[19]:
```
class KernelBase(ABC):

    def __init__(self):
        super().__init__()
        self.params = {}
        self.hyperparams = {}

    @abstractmethod
    def _kernel(self, X, Y):
        raise NotImplementedError
```

```
    def __call__(self, X, Y=None):
        return self._kernel(X, Y)

    def __str__(self):
        P, H = self.params, self.hyperparams
        p_str = ", ".join(["{}={}".format(k, v) for k, v in P.items()])
        return "{}({})".format(H["op"], p_str)

    def summary(self):
        return {
            "op": self.hyperparams["op"],
            "params": self.params,
            "hyperparams": self.hyperparams,
        }


class RBFKernel(KernelBase):

    def __init__(self, sigma=None):
        """
        RBF 核。
        """
        super().__init__()
        self.hyperparams = {"op": "RBFKernel"}
        self.params = {"sigma": sigma}  # 如果 sigma 未赋值则默认为 np.sqrt(n_features/2)，n_features 为特征数。

    def _kernel(self, X, Y=None):
        """
        对 X 和 Y 的行的每一对计算 RBF 核。如果 Y 为空，则 Y=X。

        参数说明：
        X：输入数组，为 (n_samples, n_features)
        Y：输入数组，为 (m_samples, n_features)
        """
        X = X.reshape(-1, 1) if X.ndim == 1 else X
        Y = X if Y is None else Y
        Y = Y.reshape(-1, 1) if Y.ndim == 1 else Y
        assert X.ndim == 2 and Y.ndim == 2, "X and Y must have 2 dimensions"
        sigma = np.sqrt(X.shape[1] / 2) if self.params["sigma"] is None else self.params["sigma"]
        X, Y = X / sigma, Y / sigma
        D = -2 * X @ Y.T + np.sum(Y**2, axis=1) + np.sum(X**2, axis=1)[:, np.newaxis]
        D[D < 0] = 0
        return np.exp(-0.5 * D)


class KernelInitializer(object):

    def __init__(self, param=None):
        self.param = param

    def __call__(self):
        r = r"([a-zA-Z0-9]*)=([^,)]*)"
        kr_str = self.param.lower()
        kwargs = dict([(i, eval(j)) for (i, j) in re.findall(r, self.param)])
        if "rbf" in kr_str:
            kernel = RBFKernel(**kwargs)
        else:
            raise NotImplementedError("{}".format(kr_str))
```

```
        return kernel
```

[20]:
```
class GPRegression:
    """
    高斯过程回归
    """
    def __init__(self, kernel="RBFKernel", sigma=1e-10):
        self.kernel = KernelInitializer(kernel)()
        self.params = {"GP_mean": None, "GP_cov": None, "X": None}
        self.hyperparams = {"kernel": str(self.kernel), "sigma": sigma}

    def fit(self, X, y):
        """
        用已有的样本集合得到 GP 先验。

        参数说明:
        X: 输入数组，为 (n_samples, n_features)
        y: 输入数组 X 的目标值，为 (n_samples)
        """
        mu = np.zeros(X.shape[0])
        Cov = self.kernel(X, X)
        self.params["X"] = X
        self.params["y"] = y
        self.params["GP_cov"] = Cov
        self.params["GP_mean"] = mu

    def predict(self, X_star, conf_interval=0.95):
        """
        对新的样本 X 进行预测。

        参数说明:
        X_star: 输入数组，为 (n_samples, n_features)
        conf_interval: 置信区间，浮点型 (0, 1)，default=0.95
        """
        X = self.params["X"]
        y = self.params["y"]
        K = self.params["GP_cov"]
        sigma = self.hyperparams["sigma"]
        K_star = self.kernel(X_star, X)
        K_star_star = self.kernel(X_star, X_star)
        sig = np.eye(K.shape[0]) * sigma
        K_y_inv = np.linalg.pinv(K + sig)
        mean = K_star @ K_y_inv @ y
        cov = K_star_star - K_star @ K_y_inv @ K_star.T
        percentile = norm.ppf(conf_interval)
        conf = percentile * np.sqrt(np.diag(cov))
        return mean, conf, cov
```

[21]:
```
class BayesianOptimization:

    def __init__(self):
        self.model = GPRegression()

    def acquisition_function(self, Xsamples):
        mu, _, cov = self.model.predict(Xsamples)
        mu = mu if mu.ndim==1 else (mu.T)[0]
        ysample = np.random.multivariate_normal(mu, cov)
        return ysample
```

```
    def opt_acquisition(self, X, n_samples=20):
        # 样本搜索策略，一般方法有随机搜索、基于网格的搜索，或局部搜索
        # 我们这里就用简单的随机搜索，这里也可以定义样本的范围
        Xsamples = np.random.randint(low=1,high=50,size=n_samples*X.shape[1])
        Xsamples = Xsamples.reshape(n_samples, X.shape[1])
        # 计算采集函数的值并取最大的值
        scores = self.acquisition_function(Xsamples)
        ix = np.argmax(scores)
        return Xsamples[ix, 0]

    def fit(self, f, X, y):
        # 拟合 GPR 模型
        self.model.fit(X, y)
        # 优化过程
        for i in range(15):
            x_star = self.opt_acquisition(X)  # 下一个采样点
            y_star = f(x_star)
            mean, conf, cov = self.model.predict(np.array([[x_star]]))
            # 添加当前数据到数据集合
            X = np.vstack((X, [[x_star]]))
            y = np.vstack((y, [[y_star]]))
            # 更新 GPR 模型
            self.model.fit(X, y)
        ix = np.argmax(y)
        print('Best Result: x=%.3f, y=%.3f' % (X[ix], y[ix]))
        return X[ix], y[ix]
```

用自定义的度量指标，乳腺癌数据集测试，第七章描述的随机森林用于模型训练

[22]:
```
from chapter7 import RandomForest
```

[23]:
```
column_names = ['Sample code number','Clump Thickness',
                'Uniformity of Cell Size','Uniformity of Cell Shape',
                'Marginal Adhesion','Single Epithelial Cell Size',
                'Bare Nuclei','Bland Chromatin','Normal Nucleoli','Mitoses','Class']
data = pd.read_csv('../data/cancer/breast-cancer-wisconsin.data',names=column_names)
data = data.replace(to_replace='?',value=np.nan)  # 非法字符替代
data = data.dropna(how='any')  # 去掉空值，any；出现空值行则删除
print(data.shape)
# 随机采样 25% 的数据用于测试，剩下 75% 用于构建训练集
X_train,X_test,y_train,y_test = train_test_split(data[column_names[1:10]], data[column_names[10]],
                                                 test_size=0.25, random_state=1111)
# 再查看训练样本的数量和类别分布
print(y_train.value_counts())
# 修改标签为 0 和 1
print(y_train.shape)
y_train[y_train==2] = 0
y_train[y_train==4] = 1
y_test[y_test==2] = 0
y_test[y_test==4] = 1
# 再查看训练样本的数量和类别分布
print(y_train.value_counts())
# 数据标准化预处理，保证每个维度特征均值为 0，方差为 1
ss = StandardScaler()
X_train = ss.fit_transform(X_train)
X_test = ss.transform(X_test)
y_train = y_train.as_matrix()
y_test = y_test.as_matrix()
```

```
(683, 11)
```

```
2    328
4    184
Name: Class, dtype: int64
(512,)
0    328
1    184
Name: Class, dtype: int64
```

[24]:
```
def func_black_box(k):
    model = RandomForest(n_estimators=k)
    model.fit(X_train, y_train)
    return model.score(X_test, y_test)

X0 = np.array([[4]])
y0 = np.array([func_black_box(4)])
print("initial n_estimators={}, score={}".format(4, y0[0]))
BO = BayesianOptimization()
X1, y1 = BO.fit(func_black_box, X0, y0)
print("Best n_estimators={}, score={}".format(X1, y1))
```

```
initial n_estimators=4, score=0.9122807017543859
Best Result: x=19.000, y=0.965
Best n_estimators=[19], score=[0.96491228]
```

[25]:
```
import numpy
import matplotlib
import re
import pandas
import sklearn
import itertools
import scipy

print("numpy:", numpy.__version__)
print("matplotlib:", matplotlib.__version__)
print("pandas:", pandas.__version__)
print("re:", re.__version__)
print("sklearn:", sklearn.__version__)
print("scipy:", scipy.__version__)
```

```
numpy: 1.14.5
matplotlib: 3.1.1
pandas: 0.25.1
re: 2.2.1
sklearn: 0.21.3
scipy: 1.3.1
```