Beijing Forest Studio

北京理工大学信息系统及安全对抗实验中心

超参数优化

硕士研究生 齐首华

2023年1月2日

内容提要

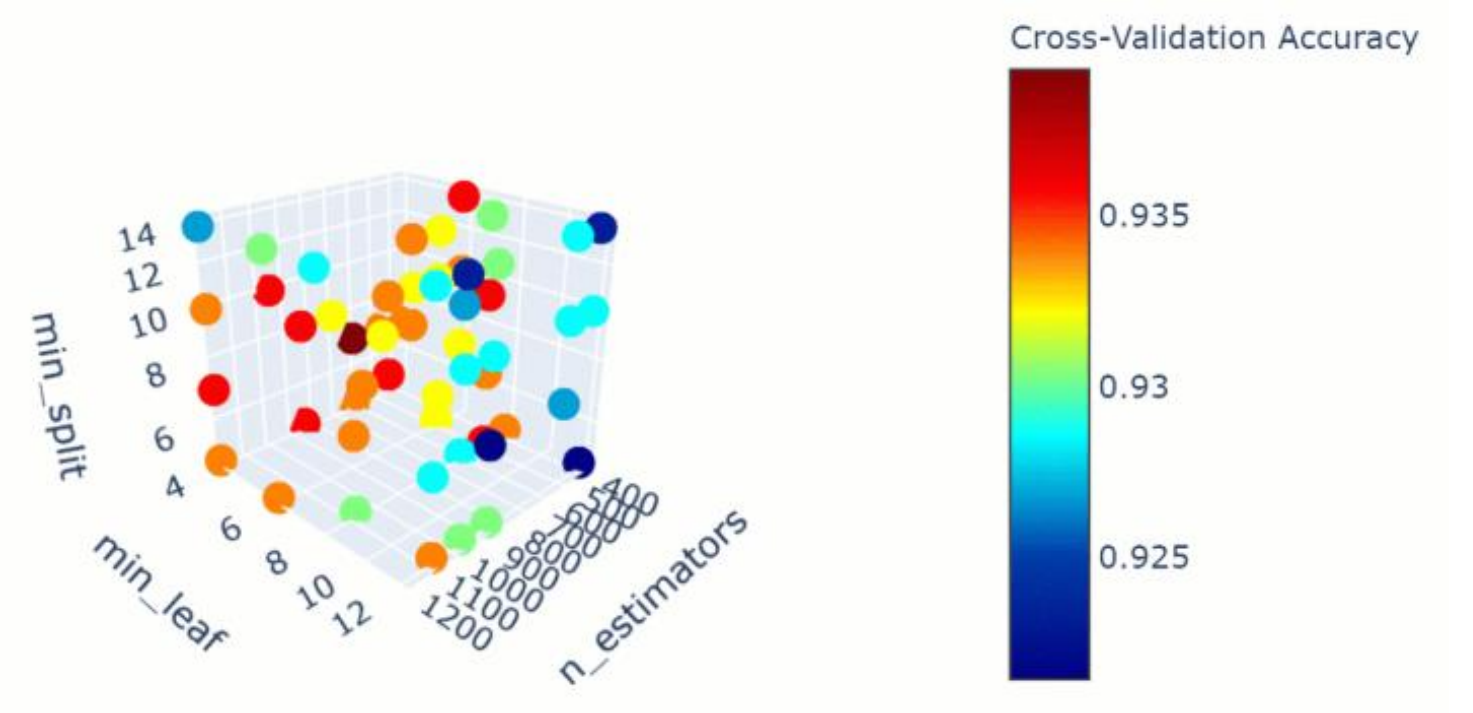
- 背景简介
- 基本概念
- 方法原理
- 总结
- 参考文献

背景简介

- 预期收获
  - 了解超参数的概念以及与普通参数的区别
  - 了解超参数对模型的影响
  - 理解超参数调优的基本方法

背景简介

- 超参数调优
  - 超参数调优是机器学习或深度学习中必不可少的步骤
  - 模型的性能与超参数选取直接相关
  - 调整超参数很难又乏味

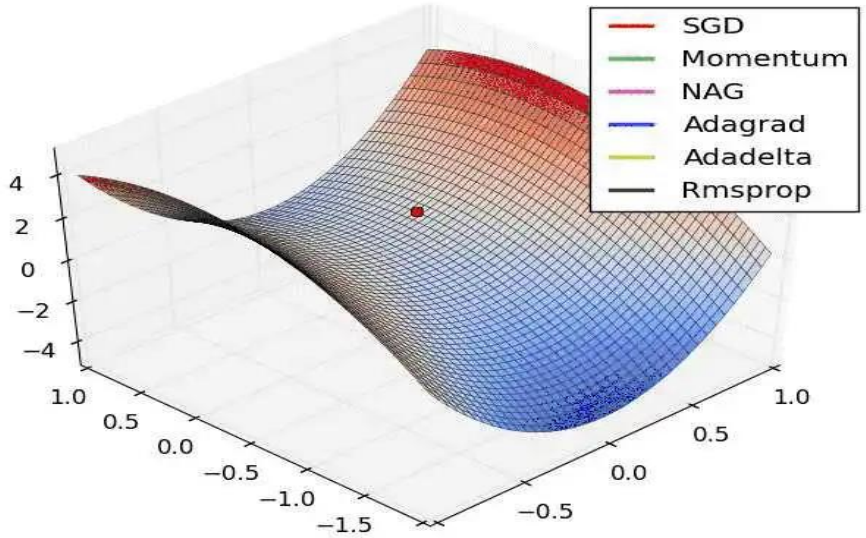


基本概念

- 超参数的概念
  - 超参数是在开始学习过程之前设置值的参数，而不是通过训练得到的参数数据
- 超参数的特点
  - 定义关于模型的更高层次的概念，如复杂性或学习能力。
  - 不能直接从标准模型培训过程中的数据中学习，需要预先定义。
  - 通过设置不同的值，训练不同的模型和选择更好的测试值来决定
- 模型参数与超参数的区别
  - 区分两者最大的一点就是是否通过数据来进行调整，模型参数通常是有数据来驱动调整，超参数则不需要数据来驱动，而是在训练前或者训练中人为的进行调整的参数

基本概念

- 超参数类型
  - 网络参数：可指网络层与层之间的交互方式（相加、相乘或者串接等）、卷积核数量和卷积核尺寸、网络层数和激活函数等。
  - 优化参数：一般指学习率、批样本数量、不同优化器的参数以及部分损失函数的可调参数。
  - 正则化参数：权重衰减系数，dropout比率



- 部分超参数对模型性能的影响

超参数	如何提升模型容量	原因	注意事项
学习率	调至最优	过高或过低的学习率，都会优化失败导致降低模型有效容量	学习率最优点，在训练的不同时间点都可能变化，所以需要一套有效的学习率衰减策略
损失函数参数	调至最优	不合适的超参数会降低模型有效容量	对于部分损失函数超参数其变化会对结果十分敏感，而有些则并不会太影响。在调整时，建议参考论文的推荐值，并在该推荐值数量级上进行最大最小值调试该参数对结果的影响
批样本数量	调至最优	大部分情况下，选择适合自身硬件容量的批样本数量，并不会对模型容限造成显著影响	在一些特殊的目标函数的设计中，如何选择样本是很可能影响到模型的有效容限的，例如度量学习（metric learning）中的N-pair loss。这类损失因为需要样本的多样性，可能会依赖于批样本数量
Dropout比率	降低	较少的丢弃参数意味着模型参数量的提升，参数间适应性提升，模型容量提升，但不一定能提升模型有效容限	
权重衰减系数	降低	权重衰减可以有效的起到限制参数变化的幅度，起到一定的正则作用	
优化器动量	调至最优	动量参数通常用来加快训练，同时更容易跳出极值点，避免陷入局部最优解	
网络层数	增加	增加深度意味着模型具有更多的参数，更强的拟合能力	深度越深意味着参数越多，需要的时间和硬件资源也越高
卷积核尺寸	增加	增加卷积核尺寸意味着参数量的增加，同条件下，模型参数也相应的增加	

基本概念

- 部分超参数合适的范围

超参数	建议范围	注意事项
学习率	SGD: [1e-2, 1e-1] momentum: [1e-3, 1e-2] Adagrad: [1e-3, 1e-2] Adadelta: [1e-2, 1e-1] RMSprop: [1e-3, 1e-2] Adam: [1e-3, 1e-2] Adamax: [1e-3, 1e-2] Nadam: [1e-3, 1e-2]	这些范围通常是指从头开始训练的情况。若是微调，初始学习率可在降低一到两个数量级。
损失函数超参数	多个损失函数之间，损失值之间尽量相近，不建议超过或者低于两个数量级	这是指多个损失组合的情况，不一定完全正确。单个损失超参数需结合实际情况。
批样本数量	[1:1024]	当批样本数量过大(大于6000)或者等于1时，需要注意学习策略或者内部归一化方式的调整
Dropout比率	[0, 0.5]	
权重衰减系数	[0, 1e-4]	
卷积核尺寸	[7x7],[5x5],[3x3],[1x1], [7x1,1x7]	

超参数调优

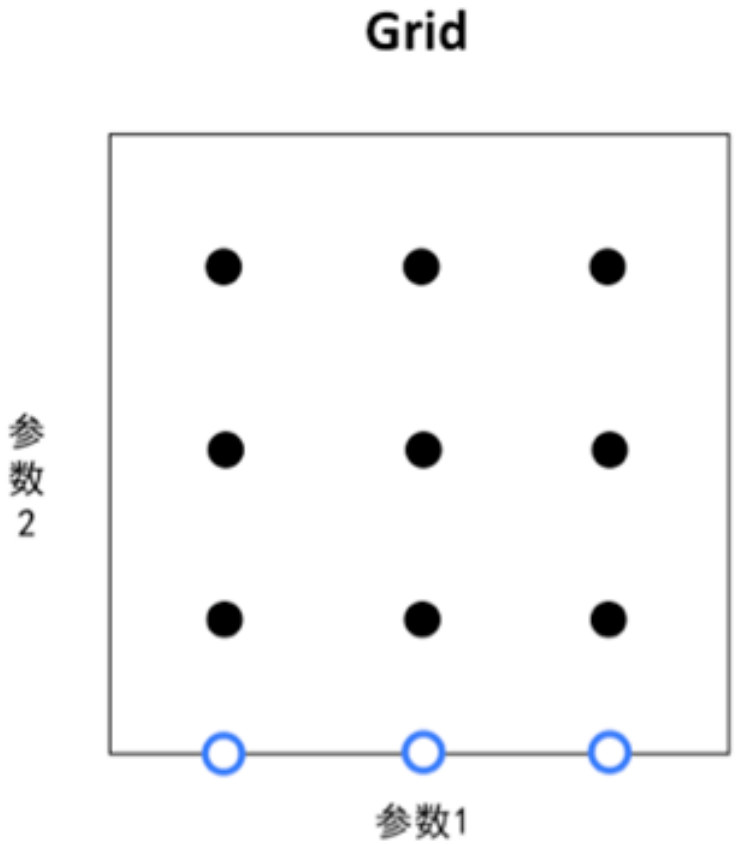
网格搜索

- 网格搜索就是遍历所有可能的超参数组合，找到能得到最佳性能的超参数组合，但是由于一次训练的计算代价很高，搜索区间通常只会限定于少量的离散数值

◆ 定义n维方格

◆ 每个方格对应一个超参数

◆ 一组一组参数尝试



网格搜索

- 优点：
  - 原理简单（穷举）
  - 可并行计算

- 缺点：
  - 当参数量很多时，非常耗费计算资源
  - 不能保证得到最佳的参数组合

```
from sklearn import svm, datasets
from sklearn.model_selection import GridSearchCV
import pandas as pd
# 导入数据
iris = datasets.load_iris()
# 定义超参搜索空间
parameters = {'kernel':('linear', 'rbf'), 'C':[1, 10]}
# 初始化模型
svc = svm.SVC()
# 网格搜索
clf = GridSearchCV(estimator = svc,
                   param_grid = parameters,
                                scoring = 'accuracy',
                                n_jobs = -1,
                                cv = 5)
clf.fit(iris.data, iris.target)
返回: GridSearchCV(cv=5, estimator=SVC(), n_jobs=-1,
                   param_grid={'C': [1, 10], 'kernel': ('linear', 'rbf')},scoring='accuracy')
# 打印结果
print('详细结果:\n', pd.DataFrame.from_dict(clf.cv_results_))
print('最佳分类器:\n', clf.best_estimator_)
print('最佳分数:\n', clf.best_score_)
print('最佳参数:\n', clf.best_params_)
```

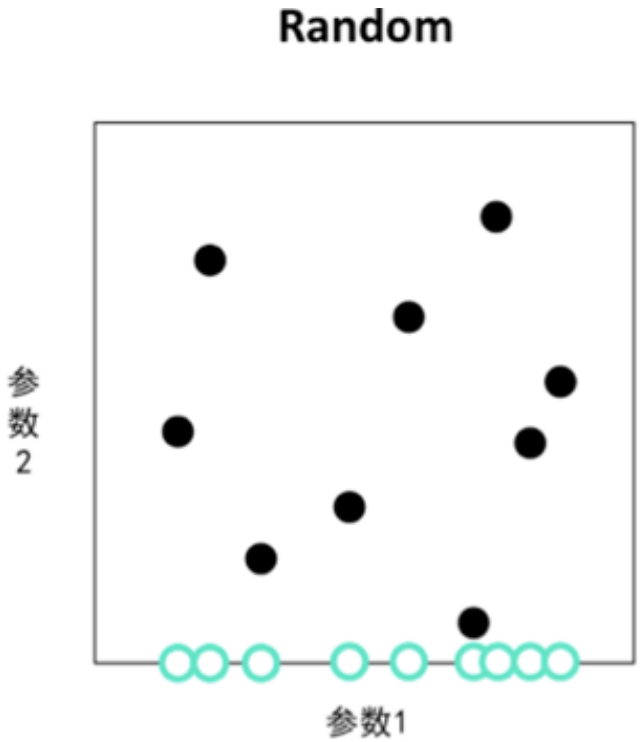
随机搜索

- 随机搜索的思想与网格搜索比较相似，只是不再测试上界和下界之间的所有值，而是在搜索范围中随机选取样本点。它的理论依据是，如果样本点集足够大，那么通过随机采样也能大概率地找到全局最优值，或其近似最优。

◆ 在n维空间中随机取点

◆ 每个点对应一组超参数组合

◆ 一组一组参数尝试



随机搜索

- 优点：
  - 随机搜索所搜索相比于网格搜索的参数空间更大，更可能获得更好的评估结果
- 缺点：
  - 忽略了计算历史，耗费计算资源依然较大

```
from sklearn import svm, datasets
from sklearn.model_selection import RandomizedSearchCV
import pandas as pd
from scipy.stats import uniform
# 导入数据
iris = datasets.load_iris()
# 定义超参搜索空间
distributions = {'kernel':['linear', 'rbf'], 'C':uniform(loc=1, scale=9)}
# 初始化模型
svc = svm.SVC()
# 网格搜索
clf = RandomizedSearchCV(estimator = svc,
                         param_distributions = distributions,n_iter = 4,scoring = 'accuracy',
                                               cv = 5,n_jobs = -1, random_state = 2021)
clf.fit(iris.data, iris.target)
返回: RandomizedSearchCV(cv=5, estimator=SVC(), n_iter=4, n_jobs=-1,
                         param_distributions={'C': <scipy.stats._distn_infrastructure.rv_frozen object
                                              'kernel': ['linear', 'rbf']},
                         random_state=2021, scoring='accuracy')
# 打印结果
print('详细结果:\n', pd.DataFrame.from_dict(clf.cv_results_))
print('最佳分类器:\n', clf.best_estimator_)
print('最佳分数:\n', clf.best_score_)
print('最佳参数:\n', clf.best_params_)
```

贝叶斯优化

- 基本思想
  - 贝叶斯优化就是一种基于先验的优化，一种根据历史信息来决定后面的路怎么走的优化方法，从而减少搜索空间，大大提升搜索效率。
  - 关键在于：用什么样的标准来判断下一步怎么走比较好。
- SMBO
  - SMBO，序列化基于模型的优化，所谓序列化，是指通过迭代的方式，通过一次试验来进行优化，是贝叶斯优化的一种具体实现形式

SMBO(f, M_0, T, S)

1 $\mathcal{H} \leftarrow \emptyset$,
2 For $t \leftarrow 1$ **to** T,
3 $x^* \leftarrow \text{argmin}_x \; S(x, M_{t-1})$,
4 Evaluate $f(x^*)$, ▷ *Expensive step*
5 $\mathcal{H} \leftarrow \mathcal{H} \cup (x^*, f(x^*))$,
6 Fit a new model M_t to $\mathcal{H}$.
7 **return** $\mathcal{H}$

Figure 1: The pseudo-code of generic Sequential Model-Based Optimization.

- SMOB流程

SMBO(f, M_0, T, S)

1 $\mathcal{H} \leftarrow \emptyset$,
2 For $t \leftarrow 1$ **to** T,
3 $x^* \leftarrow \text{argmin}_x \; S(x, M_{t-1})$,
4 Evaluate $f(x^*)$, ▷ *Expensive step*
5 $\mathcal{H} \leftarrow \mathcal{H} \cup (x^*, f(x^*))$,
6 Fit a new model M_t to $\mathcal{H}$.
7 **return** $\mathcal{H}$

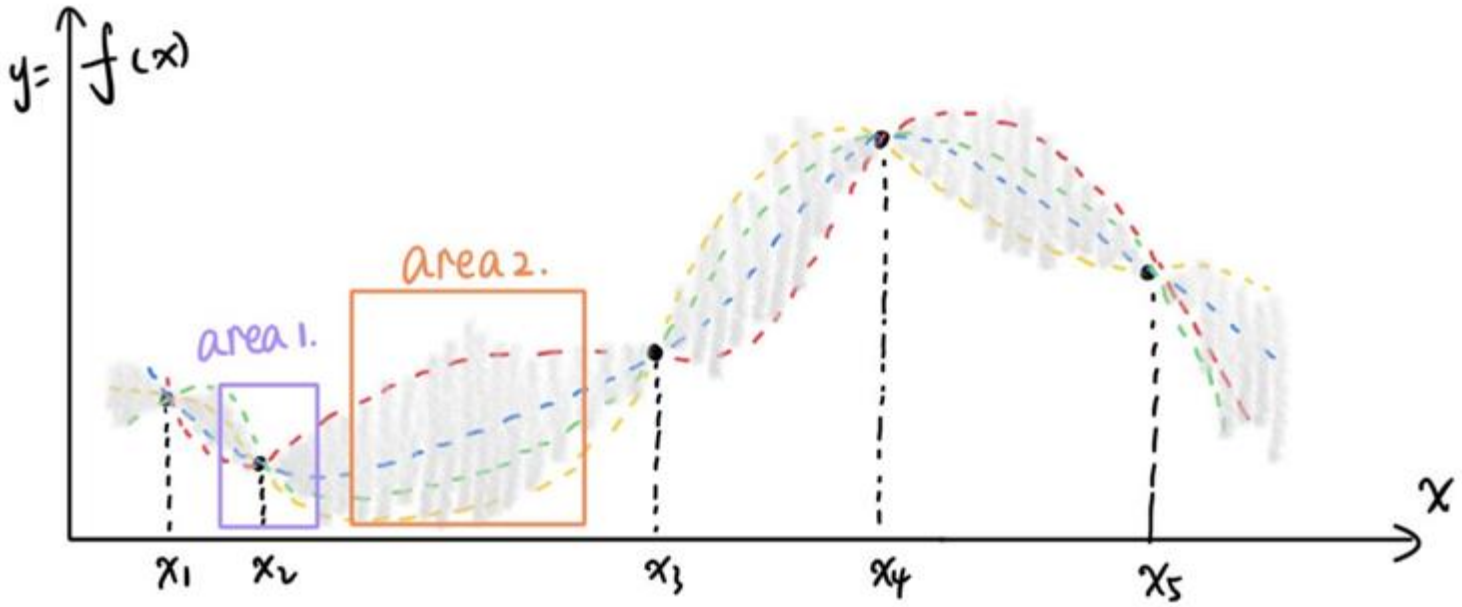
Figure 1: The pseudo-code of generic Sequential Model-Based Optimization.

- f是要去优化的函数
- x是超参数组合
- S为代理函数
- H是前面所有的{x, f(x)}的记录，要对H进行建模，得到它们的概率分布模型M

- 所以，不同的贝叶斯优化方法，主要区别在：
  - 用何种概率模型对历史进行建模
  - 如何挑选下一步超参数

贝叶斯优化

- 如何挑选下一步超参数
  - Exploitation：尽量选择靠近已知点的点为下一次用于迭代的参考点，即尽量挖掘已知点周围的点
  - Exploration：尽量选择远离已知点的点为下一次用于迭代的参考点，即尽量探索未知的区域，点的分布会尽可能的平均



贝叶斯优化

- 需要设计一个acquisition function来判断。一种最常用的方案就是Expected Improvement (EI)，它是对Exploration和Exploitation做了一个折中

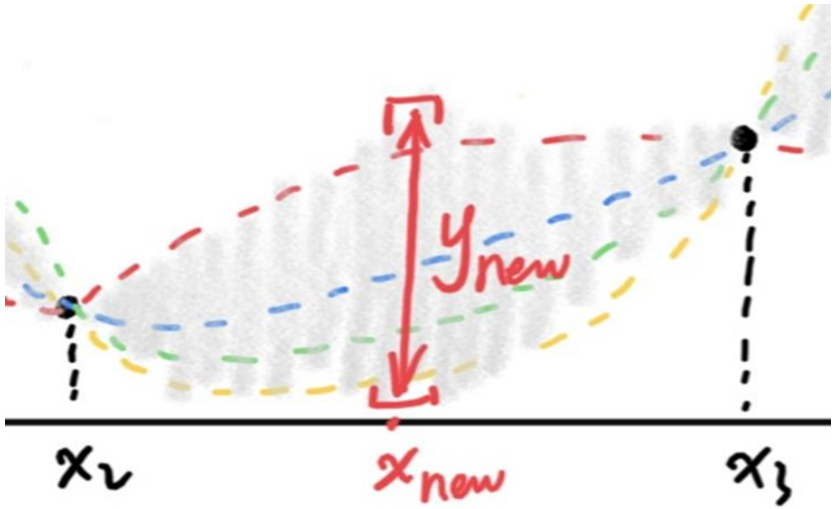
$$EI_{y^*}(x) = \int_{-\infty}^{+\infty} max(y^* - y, 0) p_M(y|x) dy$$

  - y*是某个阈值，EI就是一个期望，该期望是x的函数。当给定x的时候，EI(x)就是y相对于阈值y*平均提升了多少
- 通过最大化EI找到下一步要找的超参数

$$x_{new} = argmax_x EI_{y^*}(x)$$

贝叶斯优化

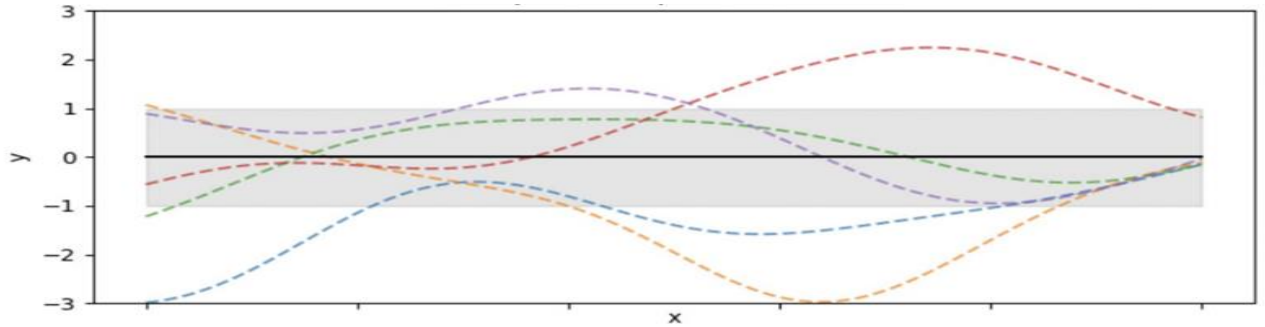
- 对历史观测进行建模
  - 对要优化的目标函数进行建模，得到该函数的分布 $p(y|x)$，从而了解该函数可能会在什么范围内波动
  - 当给定一个新的超参数x_{new}，y_{new}的取值可以根据$p(y|x_{new})$给出



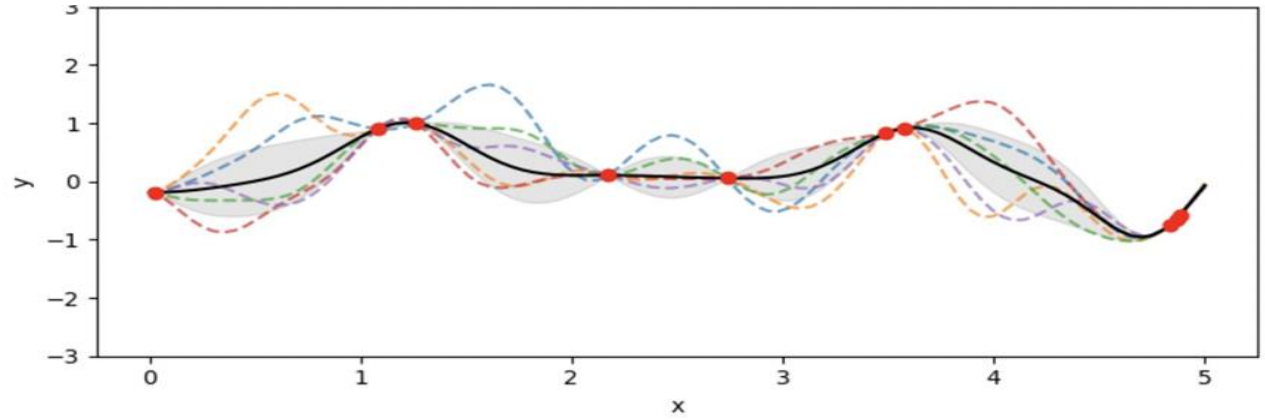
- 具体的建模方法，包括高斯过程回归（GPR）和Tree Parzen Estimator（TPE）

贝叶斯优化

- 基于GPR的贝叶斯优化
  - 对x和y做的一个先验假设：每一个x对应的y，都是一个高斯分布
  - 那么当我们还没有任何观测点时，y实际上服从一个无限维的高斯分布



  - 高斯过程回归，就是根据一些观测点，来进行一些推断



贝叶斯优化

在GPR中，我们选取当前观测结果中最好的y来作为y*，有了p(y|x)，对EI函数进行优化

$$maximize_x \ EI_{y^*}(x)$$

$$EI_{y^*}(x) = \int_{-\infty}^{+\infty} max(y^* - y, 0) p_M(y|x) dy$$

贝叶斯优化

- 基于TPE的贝叶斯优化
  - 根据贝叶斯定理:

$$p(y|x) = \frac{p(x|y)p(y)}{p(x)}$$

  - 把p(y|x)分解成p(x|y)和p(y)，TPE对p(x|y)进行了如下划分：

$$p(x|y) = \begin{cases} l(x), & y < y^* \\ g(x), & y > y^* \end{cases}$$

  - 在阈值y*两侧的观测点x，构造不一样的分布

$$p(x) = \gamma l(x) + (1-\gamma)g(x)$$

贝叶斯优化

- 把上述公式带入到EI公式中

$$EI_{y^*}(x) = \frac{\int_{-\infty}^{y^*}(y^* - y)p(y)dy}{\gamma + (1-\gamma)\frac{g(x)}{l(x)}}$$

- TPE跟GPR的区别就在于，其对历史观测结果的建模更加精细一些，对观测结果按照成绩进行了划分，分段进行概率分布建模。

贝叶斯优化

- 优点：
  - 考虑之前的参数信息，不断地更新先验数据
  - 贝叶斯调参迭代次数少，速度相对较快
- 缺点：
  - 由于初始化存在随机性，其效果不稳定，可能陷入局部点。

```python
from sklearn import svm, datasets
from sklearn.model_selection import cross_val_score
from hyperopt import hp, fmin, tpe, space_eval
import pandas as pd

# 导入数据
iris = datasets.load_iris()
# step1: 定义目标函数
def objective(params):
            # 初始化模型并交叉验证
            svc = svm.SVC(**params)
          cv_scores = cross_val_score(svc, iris.data, iris.target, cv=5)
            # 返回loss = 1 - accuracy （loss必须被最小化）
            loss = 1 - cv_scores.mean()
            return loss
# step2: 定义超参搜索空间
space = {'kernel':hp.choice('kernel', ['linear', 'rbf']),
                'C':hp.uniform('C', 1, 10)}
# step3: 在给定超参搜索空间下，最小化目标函数
best = fmin(objective, space, algo=tpe.suggest, max_evals=100)
# step4: 打印结果
print(best)
```

总结

总结

- 超参数概念
  - 超参数对模型性能的影响
  - 部分超参数合适的范围

- 超参数调优
  - 网格搜索
  - 随机搜索
  - 贝叶斯优化

现有项目

- Hyperopt
  - Hyperopt是一种通过贝叶斯优化来调整参数的工具，该方法较快的速度，并有较好的效果。此外，Hyperopt结合MongoDB可以进行分布式调参，快速找到相对较优的参数。
- OpenBox开源项目
  - 相较于现有的黑盒优化（超参数优化）系统，OpenBox支持更广泛的使用场景，包括多目标优化、带约束条件优化、多种参数类型、迁移学习、分布式并行验证、多精度优化等。
  - https://github.com/sponsors
- NNI
  - NNI 是微软开源的自动机器学习（AutoML）的工具包。它通过多种调优的算法来搜索最好的神经网络结构和超参，并支持单机、本地多机、云等不同的运行环境。
  - https://nni.readthedocs.io/zh/stable/

总结

- 上面介绍的超参数优化方法都是在固定（或变化比较小）的超参数空间中进行最优配置搜索，而最重要的神经网络架构一般还是需要由有经验的专家来进行设计。
- 神经架构搜索(neural architecture search,NAS)是一个新的比较有前景的研究方向通过神经网络来自动实现网络架构的设计。一个神经网络的架构可以用一个变长的字符串来描述。利用元学习的思想，神经架构搜索利用一个控制器来生成另一个子网络的架构描述。

参考文献

- Bergstra, J., Bardenet, R., Bengio, Y., & Kégl, B. Algorithms for hyper-parameter optimization. In 25th annual conference on neural information processing systems ,2011
- Bergstra, J., & Bengio, Y.Random search for hyper-parameter optimization. Journal of machine learning research, 2012.13(2).
- sklearn.model_selection.GridSearchCV. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.GridSearchCV.html#sklearn.model_selection.GridSearchCV
- sklearn.model_selection.RandomizedSearchCV. https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.RandomizedSearchCV.html#sklearn.model_selection.RandomizedSearchCV
- Hyperopt: Distributed Hyperparameter Optimization. https://github.com/hyperopt/hyperopt#getting-started

谢谢！

大成若缺，其用不弊。大盈若冲，其用不穷。大直若屈。大巧若拙。大辩若讷。静胜躁，寒胜热。清静为天下正。