目录

简介

【参考】

- Bagging and Random Forest Ensemble Algorithms for Machine Learning

欲得到泛化性能强的集成，集成中的个体学习器应尽可能的相互独立，虽然独立在现实任务中很难做到，但可以设法使得基学习器尽可能的具有较大的差异。

给定一个训练数据集，一个可能的做法是对训练样本进行采样，产生若干个不同的子集，再从每个子集中训练一个基学习器。这样由于训练样本集的不同，可以让基学习器有较大的差异。但没有也希望学习器不能太差，如果采样的每个子集都完全不同，则每个基学习器只用到了一个小部分训练集，甚至不足以有效的学习，这就无法保证会产生比较好的学习器。为了解决这个问题可以采用交叠采样子集。

Bagging

概念：自主采样法（boostrap sampling）、投票法、平均法、包外估计（out-of-bag estimate）、

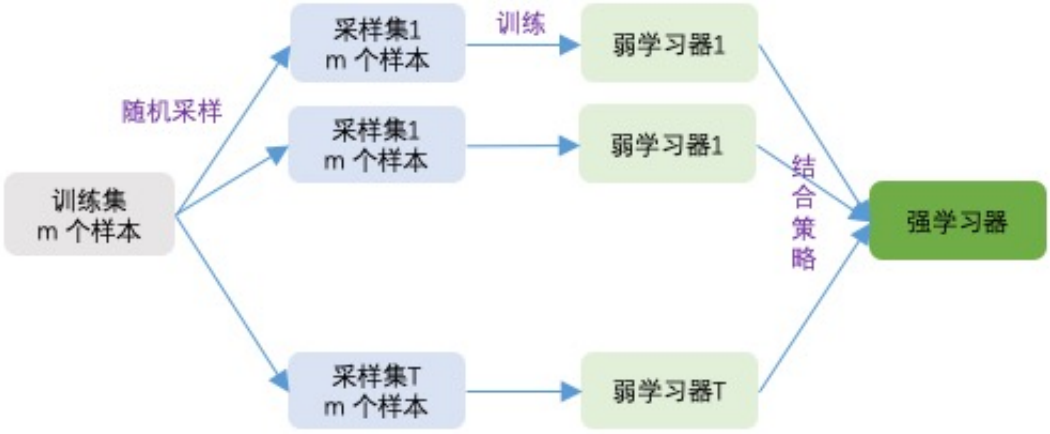
【参考】

- cnblogs - Bagging与随机森林算法原理小结

Bagging 是并行式集成学习方法最著名的代表。Bagging 是 Boosting AGGregatING 的简称。它基于**自主采样法（boostrap sampling）**。其操作过程如下：

给定包含 m 个样本的训练集，我们随机取出一个样本放入采样集中，再把该样本放回初始训练集中，使得下次采样时该样本仍有可能被选中，这样，经过 m 次随机采样操作，我们得到了一个含有 m 个样本的采样集，初始训练集中有的样本在采样集中多次出现，有的从未出现。最终，初始训练集中有63.2%的样本出现在采样集中。

之后我们采样出 T 个（T 轮）含 m 个训练样本的采样集，然后基于每个采样集训练出一个基学习器，再将这些基学习记性结合。这就是 Bagging 的基本流程，在进行预测输出时，Bagging 通常**对分类任务使用简单的投票法，对回归任务进行简单的平均法**。若分类预测时出现两个类收到同样的票数情形，则随机的从中选择一个，也可以进一步观察学习器投票的置信度来确定最终胜利者。



算法流程

输入：
训练集：$D=\{(\boldsymbol{x}_1,y_1),(\boldsymbol{x}_2,y_2),\cdots,(\boldsymbol{x}_m,y_m)\}$，含有 m 个样本
基学习算法：$\mathfrak{L}$
训练轮数：T

过程： for t=1,2,...T do
$h_t=\mathfrak{L}(D,\mathcal{D}_{bs})$
end for

输出：

$$H(\boldsymbol{x})=arg\max_{y\in Y}\sum_{t=1}^{T}\mathbf{I}(h_t(\boldsymbol{x})=y)$$

特点

与标准 AdaBoost 只适用于二分类任务不同，Bagging 能不经修改的用于多分类、回归任务。

自主采样也为 Bagging 带来了一个好处：由于每个基学习器只用了训练集约63.2%的样本，剩下的36.8%的样本可用于验证集来对泛化性能进行**包外估计（out-of-bag estimate）**。为此需要记录每个学习器所使用的训练样本，令 D_t 表示 h_t 实际使用的训练样本集，令 H^{oob} 表示对样本 $\boldsymbol{x}$ 的包外预测，即仅考虑那么未使用 $\boldsymbol{x}$ 训练的基学习器在 $\boldsymbol{x}$ 上预测，有：

$$H^{oob}(\boldsymbol{x}) = arg\max_{y \in y} \sum_{t=1}^{T} \mathbf{I}(h_t(\boldsymbol{x}) = y) \cdot \mathbf{I}(\boldsymbol{x} \notin D_t)$$

则 Bagging 的泛化误差的保外估计为：

$$\epsilon^{oob} = \frac{1}{|D|} \sum_{(\boldsymbol{x},\boldsymbol{y}) \in D} \mathbf{I}(H^{oob}(\boldsymbol{x}) \neq y)$$

包外样本还有很多其他用途，例如当基学习器是决策树，可以使用保外样本来辅助剪枝，或用于估计决策树中各节点的后验概率以辅助对零训练样本节点处理；但基学习器是神经网络的时候，可以使用包外样本来辅助早期停止以减少过拟合的风险。

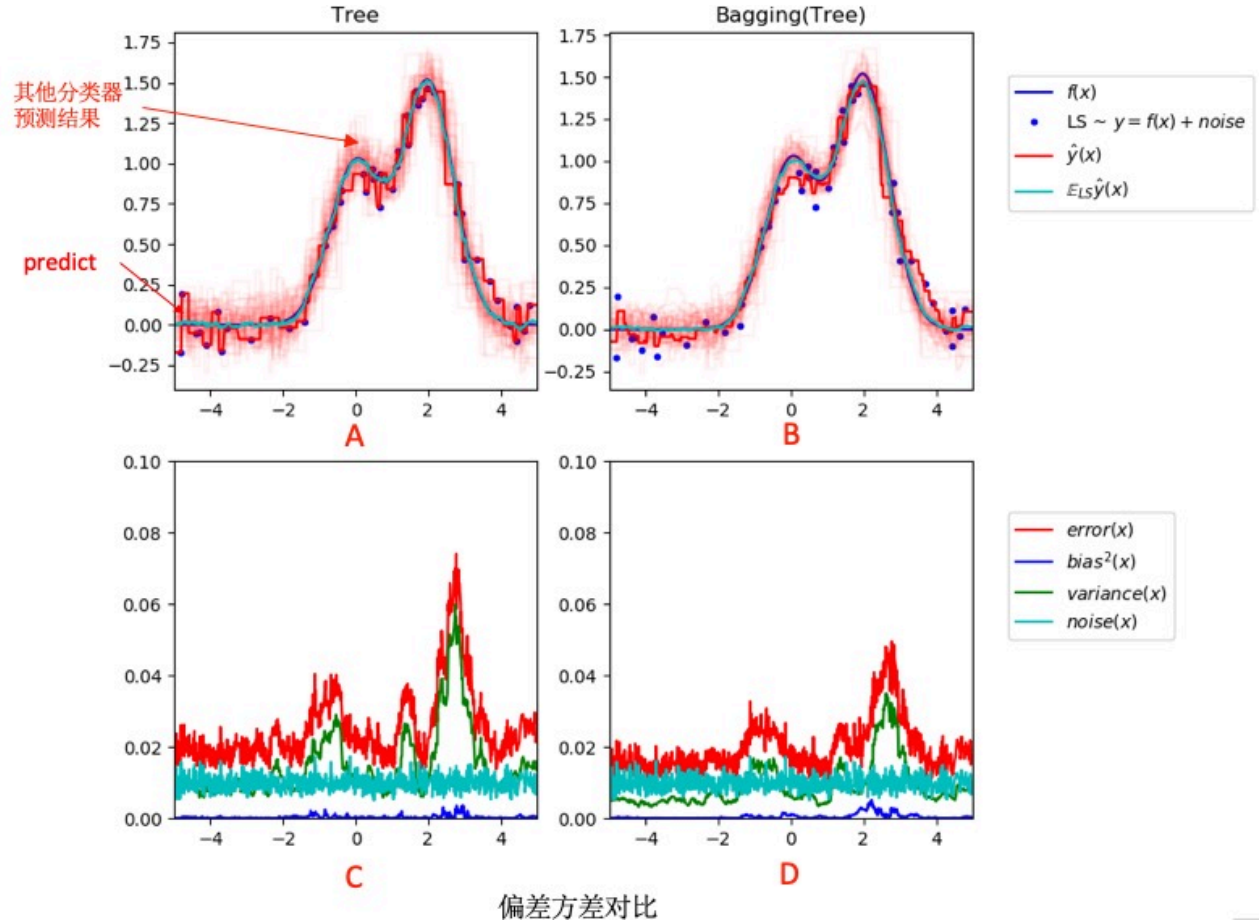
从偏差-方差分解的角度来看，Bagging 主要关注**降低方差**。由于Bagging算法每次都进行采样来训练模型，因此泛化能力很强，对于降低模型的方差很有作用。当然对于训练集的拟合程度就会差一些，也就是模型的偏差会大一些。因此他在不剪枝决策树、神经网络等易受样本扰动的学习器上效用更为明显。

Bagging对于弱学习器没有限制，这和Adaboost一样。但是最常用的一般也是决策树和神经网络。

偏差-方差分解分析

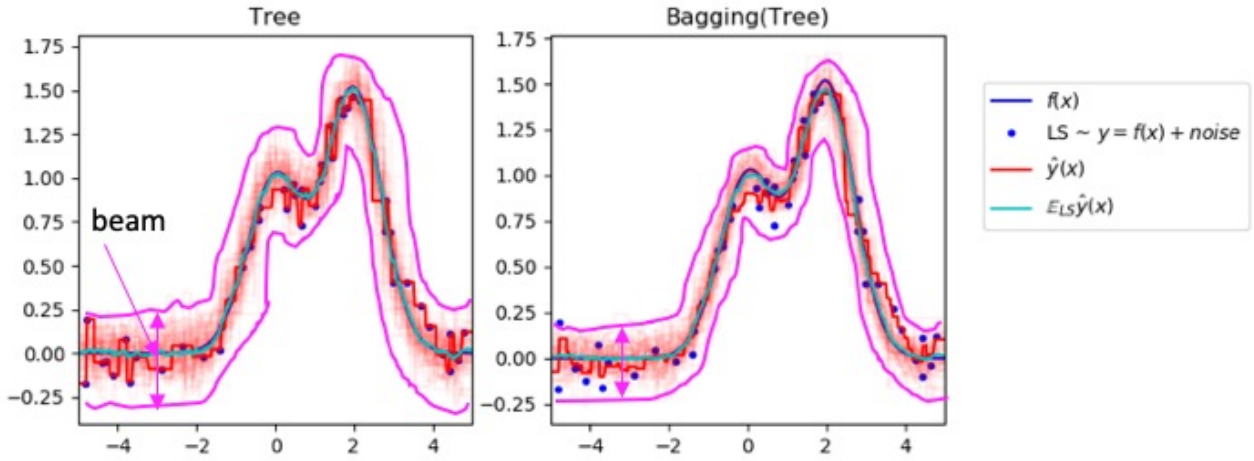
- Single estimator versus bagging: bias-variance decomposition

下图是单个评估器与Bagging 在偏差与方差上的对比，每个评估器均运行50次得出50个运行结果进行对比：

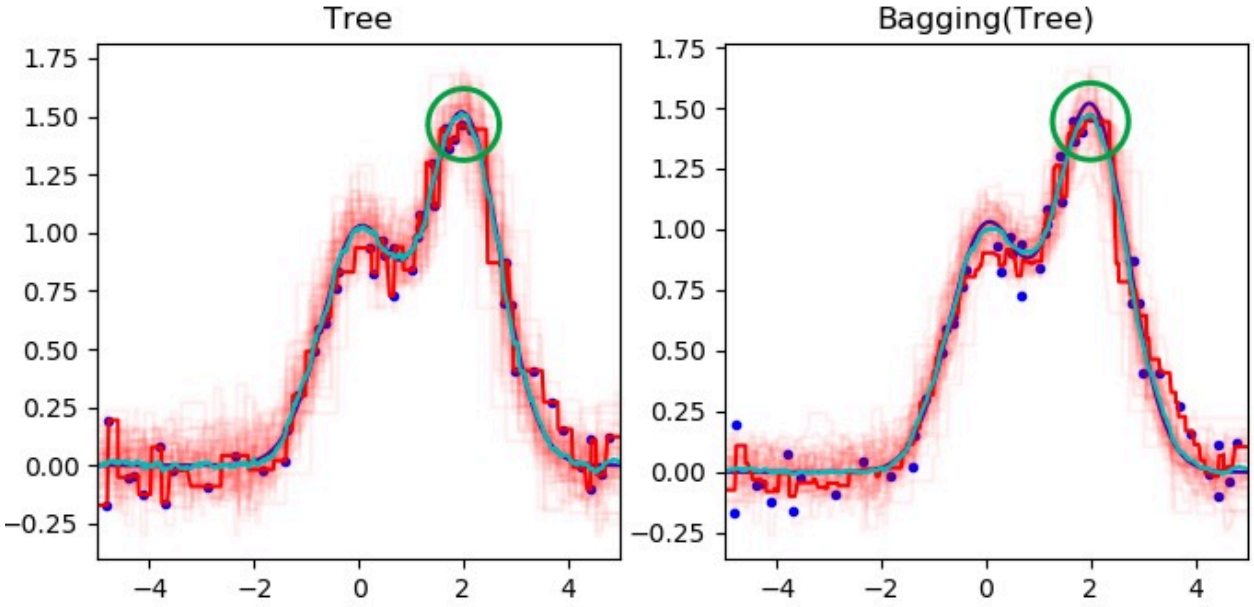


偏差方差对比

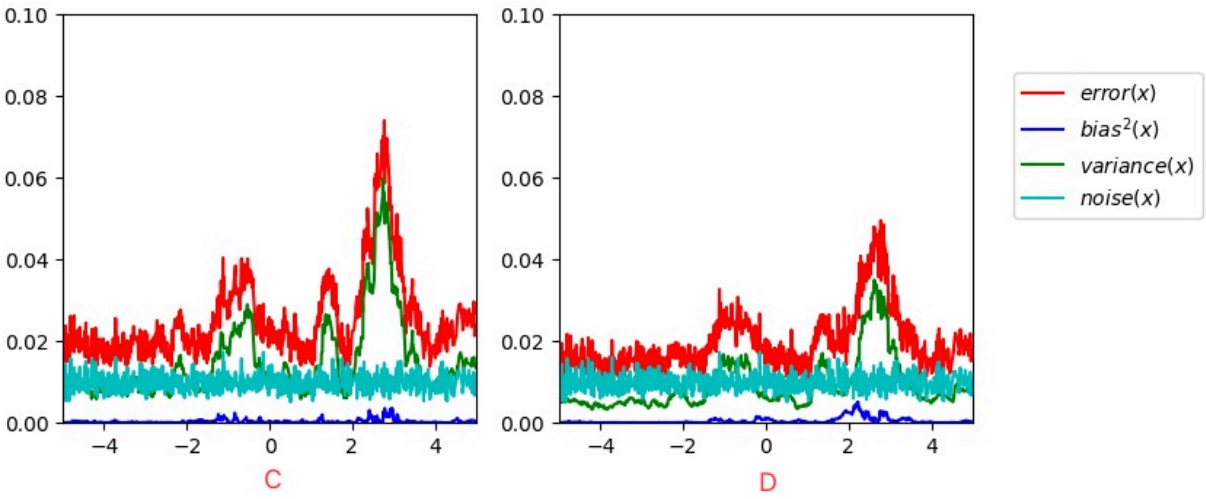
A 图是单个评估器(estimator) 得到的结果图，其中深红色的线是其中一个评估器预测结果，浅红色是其他评估器的预测结果。直觉上来说，方差对应于各个评估器预测的 beam 宽度，如下图：可以看到单个决策树的 beam 宽度要大于使用 Bagging 得到的结果，这在图 C 和 D 中也可以得到印证。



方差越大，那么预测结果对数据集中样本的小变化就越敏感。偏差则对应了模型（50个）的预测平均值（蓝绿色）与最好可能模型（深蓝色）之间的差距，可以看到 A 图的偏差是比较小的，基本上两条线重合，但方差比较大，但 B 图偏差较大，但方差较小：



C、D 对应了单个决策树逐点的均方误差(mean squared errow)，从图 C 和 D 中可以看到，C 中的偏差（蓝色线）小于 D 中的，但 D 中的方差（绿色线）明显下小于 C 中的；而 noise 在两图中保持一致，基本上维持在 0.01，因为 noise 至于数据集有关，两者使用的是同一个数据，因此在图中的值是一致的。



因此最终的误差，Bagging 要小于单个决策树，而这个差距主要在于 Bagging 减少了方差。

随机森林

【附加】

- arxiv - Understanding Random Forests: From Theory to Practice
- sklearn - OOB Errors for Random Forest

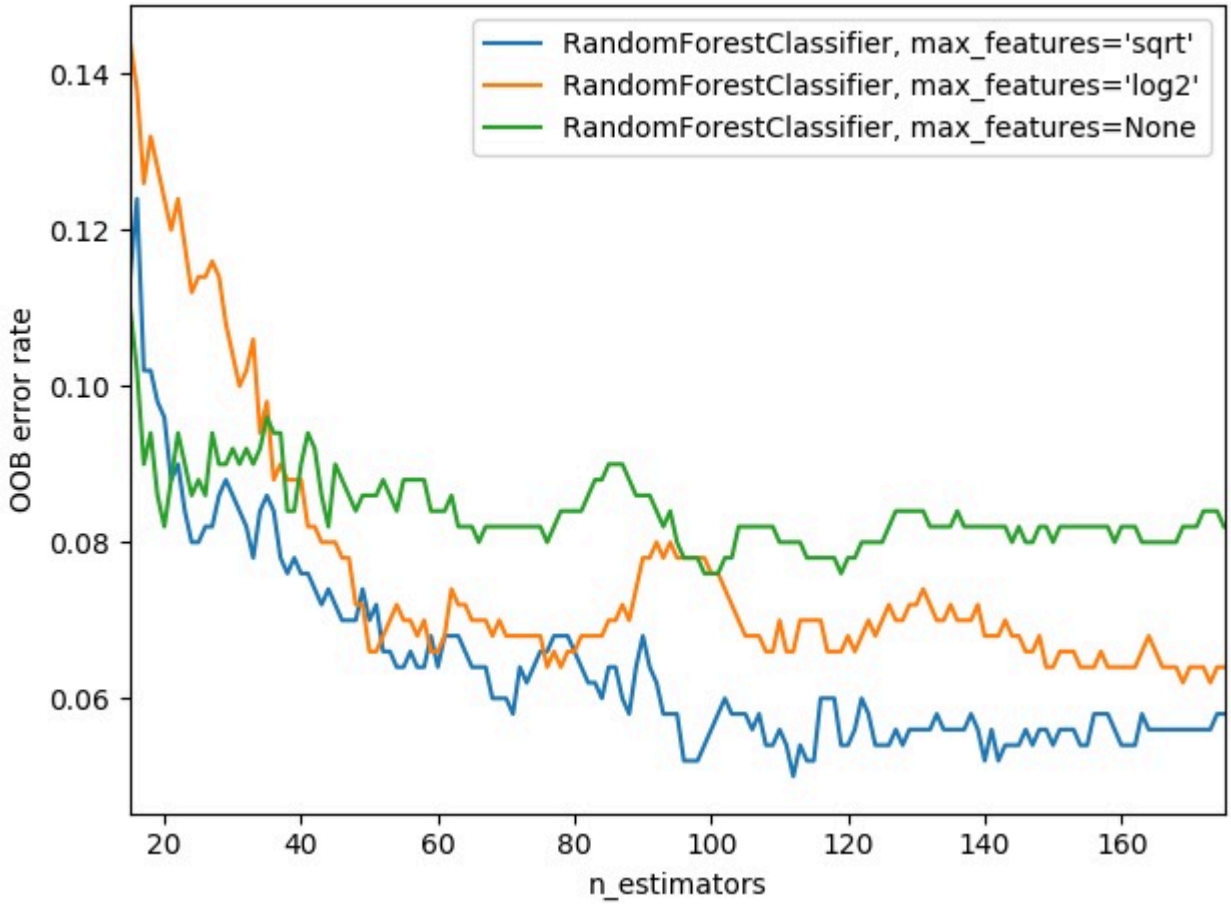
概念：确定型、随机型

`随机森林（Random Forest RF）`是 Bagging 的一个扩展体。RF 在以决策树为基学习器构建 Bagging 集成基础上，进一步在决策树训练过程中引入**随机属性选择**。

具体来说，传统决策树在划分属性时是在当前节点的属性集合中（假定有 d 个属性）中选择一个最优属性（通过信息增益或增益率）；而在 RF 中，对于基决策树的每一个节点，先从该节点的属性结合中**随机**选择一个包含 k 个属性的子集，然后再从这个子集中选择一个最优的属性进行划分。
参数 k 控制了随机性引入的程度：若令 $k = d$，则基决策树的构建与传统的决策树相同；若令 k=1，则是随机选择一个属性用于划分。

一般情况下，推荐的值为：$k = \log_2 d$

选择不同的 k 方法对于误差率的影响：



随机森林特点

随机深林简单、容易实现、计算开销小，而且他在很多现实任务中展现了强大的性能，被誉为“代表集成学习技术水平的方法”。

可以看出随机深林只是对Bagging 做了很小的改动，但是与 Bagging 中基学习器的多样性仅通过样本扰动（通过对初始训练集采样）而来不同，随机深林中基学习器的多样性来自样本的扰动，还来自属性的扰动，这就使得最终集成的泛化性能可通过这个学习器之间的差异进一步提升。

RF 的训练效率常优于 Bagging，因为在个体决策树构建过程中，Bagging 使用的是**确定型**决策树，在选择划分属性是要对节点的所有属性进行考察，而随机森林是**随机型**决策树则只考察一个属性子集。

RF的主要优点有：

- 1） 训练可以高度并行化，对于大数据时代的大样本训练速度有优势。个人觉得这是的最主要的优点。
- 2） 由于可以随机选择决策树节点划分特征，这样在样本特征维度很高的时候，仍然能高效的训练模型。
- 3） 在训练后，可以给出各个特征对于输出的重要性
- 4） 由于采用了随机采样，训练出的模型的方差小，泛化能力强。
- 5） 相对于Boosting系列的Adaboost和GBDT， RF实现比较简单。
- 6） 对部分特征缺失不敏感。

RF的主要缺点有：

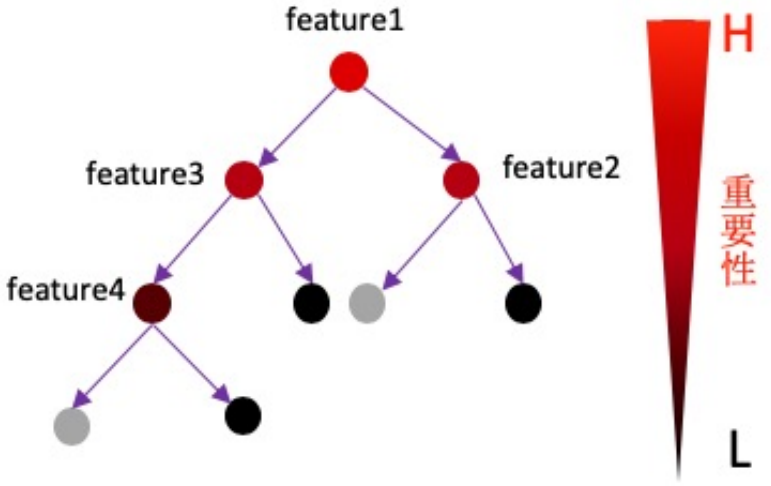
- 1）在某些噪音比较大的样本集上，RF模型容易陷入过拟合。
- 2）取值比较多的特征容易对RF的决策产生更大的影响，从而影响拟合的模型的效果。

虽然增加集成弱学习器的个数可以提高精确度，但不是可以无限制的增加，弱学习器的数量越多训练花费的时间越多，而且精度在弱学习器达到一定数量之后就不在增加，因此需要合理的设置弱学习器的个数。

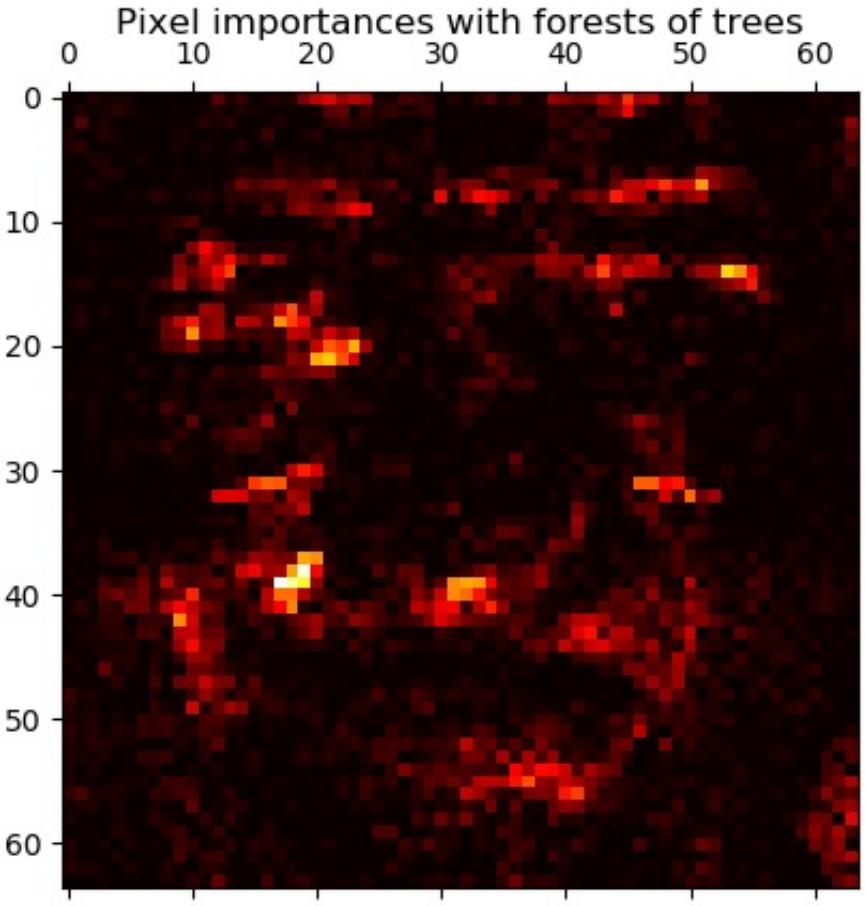
应用 - 像素及特征重要性

- sklearn - Pixel importances with a parallel forest of trees
- sklearn - Feature importances with forests of trees

特征在决策决定的相对深度，可以用于评估特征对于预测目标变量的重要性。特征在上树的顶层，对于输入样本组中的预测精确度影响越大。他们对样本的预期分数的影响，可以用作评估特征的相对重要性：

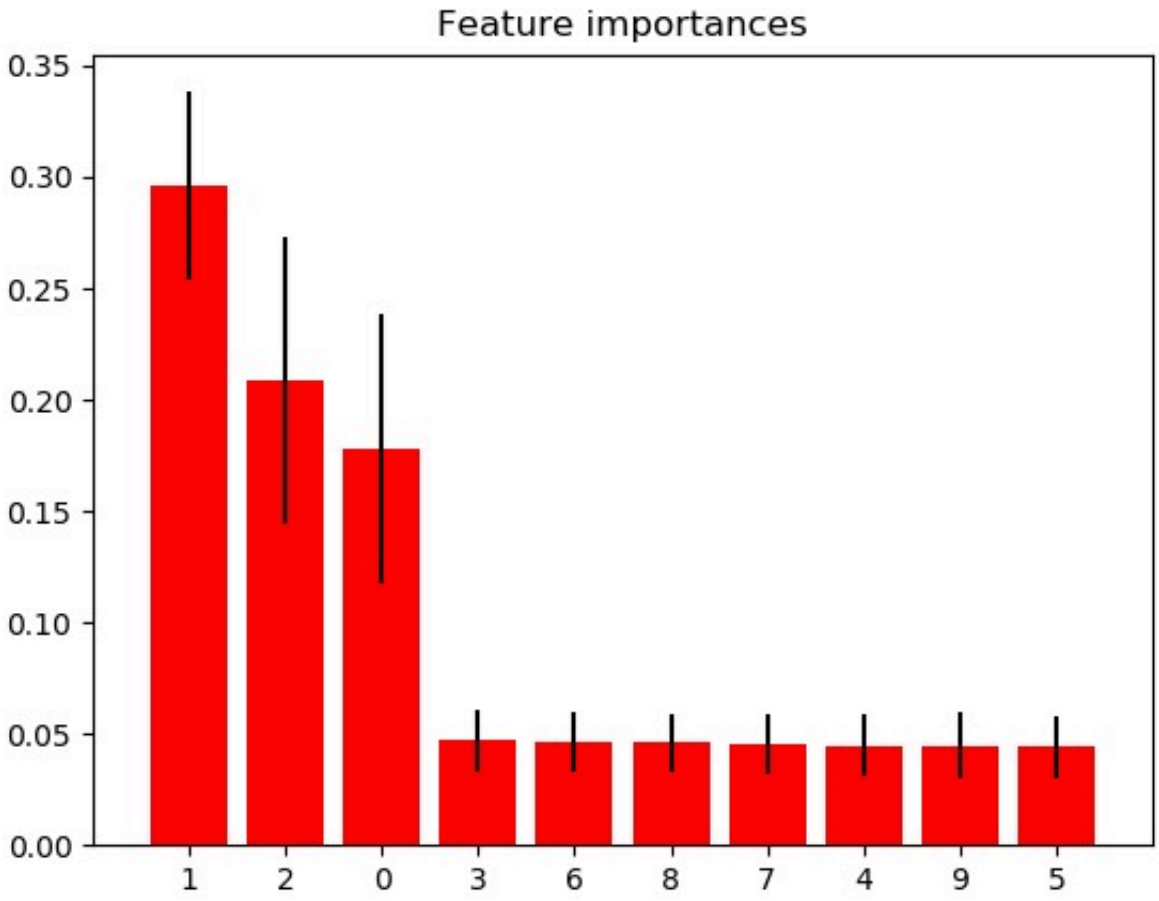


如下图，使用随机森林对图像像素点在分类中重要性的评估：



可以了解到，颜色越亮的部分，对于最终分类结果影响越大。对于图像来说这也是可以理解的，大部分图像中的物体都是居中分布，边缘上的像素对于最终的分类基本上没有贡献。

同样此方法可以用于特征的选取，选择几个对分类最重要的特征进行训练，可以大大减少训练的时间和避免过拟合：



数据转换

- sklearn - Totally Random Trees Embedding
- sklearn - Hashing feature transformation using Totally Random Trees

简介

【参考】

- 个站 - Why One-Hot Encode Data in Machine Learning?
- quora - What is one-hot encoding and when is it used in data science?
- hackernoon - What is One Hot Encoding? Why And When do you have to use it?

随机深林不仅可以用于特征或者像素重要性的判断，也可以实现对数据的转换通过非监督的方法，使用`完全随机森林(completely random tree)`通过样本最终落入的叶子节点的索引来编码数据。索引通过 one-of-K（one hot encoding，即 独热编码） 的形式进行编码，形成高维的，二分类稀疏编码。编码的大小和稀疏度可以通过选择集成树的数量和树深度来控制，对于集成中的每一个树，编码包含了树种的一个，即每一个编码对应了集成树中的一个样本。编码的最大值可以通过集成中树的个数`n_estimators`和个树的深度`max_depth`确定：`n_estimators * 2 ** max_depth`。

因为邻近的数据点倾向于落在相同的树节点上，因此转换是隐式的、非参数化的密度平度。

实现

使用 sklearn 的 `RandomTreesEmbedding` 进行数据转换可以通过如下方式进行：

```
import numpy as np
import graphviz
from sklearn import tree
from sklearn.datasets import make_circles
from sklearn.ensemble import RandomTreesEmbedding

# 生成人造数据
X, y = make_circles(n_samples=10, factor=0.5, random_state=0, noise=0.05)
print(X, X.shape)
print(y, y.shape)

# 使用 RandomTreesEmbedding 转换数据
# 使用 3 个评估器，也就是会生成三棵树
hasher = RandomTreesEmbedding(n_estimators=3, random_state=0, max_depth=3)
X_transformed = hasher.fit_transform(X)
# 获取评估器
estimators = hasher.estimators_
for i, dt in enumerate(estimators):
    # 导数生成树节点
    dot_data = tree.export_graphviz(dt, out_file=None,
                                    class_names=np.array([0, 1]),
feature_names=['xAx', 'yAx'],
                                    filled=True, rounded=True,
special_characters=True,
                                    leaves_parallel=True)
    graph = graphviz.Source(dot_data)
    graph.render("featureTran" + str(i))

print(X_transformed.shape)
print(X_transformed)
```

解析

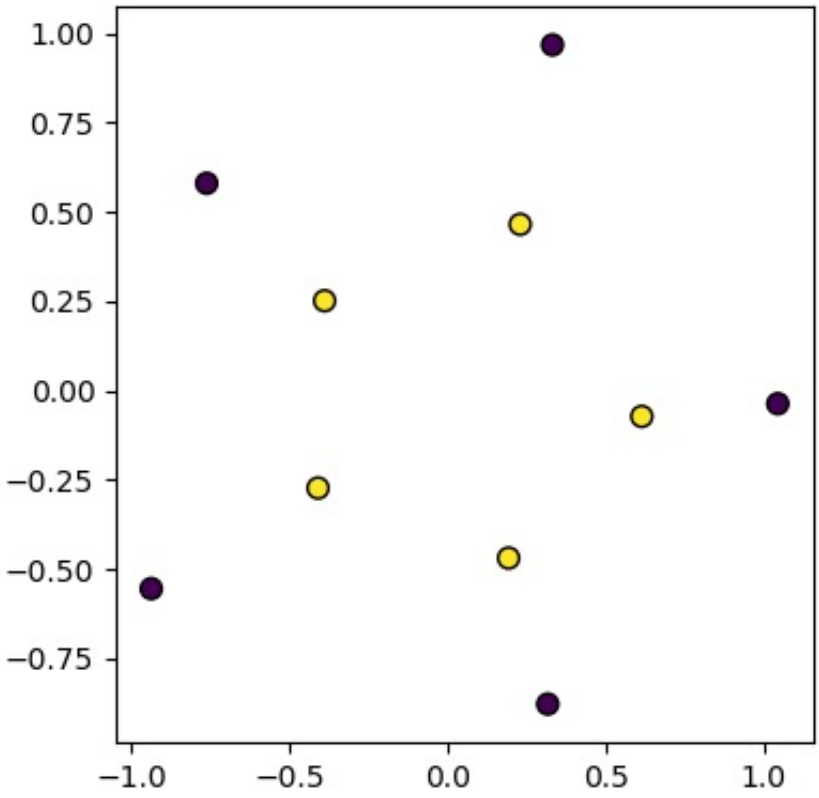
生成的人造数据大小为 (10×2) 如下：

```
[[-0.76151257  0.58021739]
 [-0.40966944 -0.2733627 ]
 [ 0.31621917 -0.87834284]
 [ 0.19256038 -0.46944451]
 [ 0.33121016  0.96774023]
 [ 0.22921245  0.46527034]
 [-0.38885511  0.25118784]
 [-0.93666649 -0.55510432]
 [ 1.04322181 -0.03710825]
 [ 0.61348773 -0.07271828]]
```

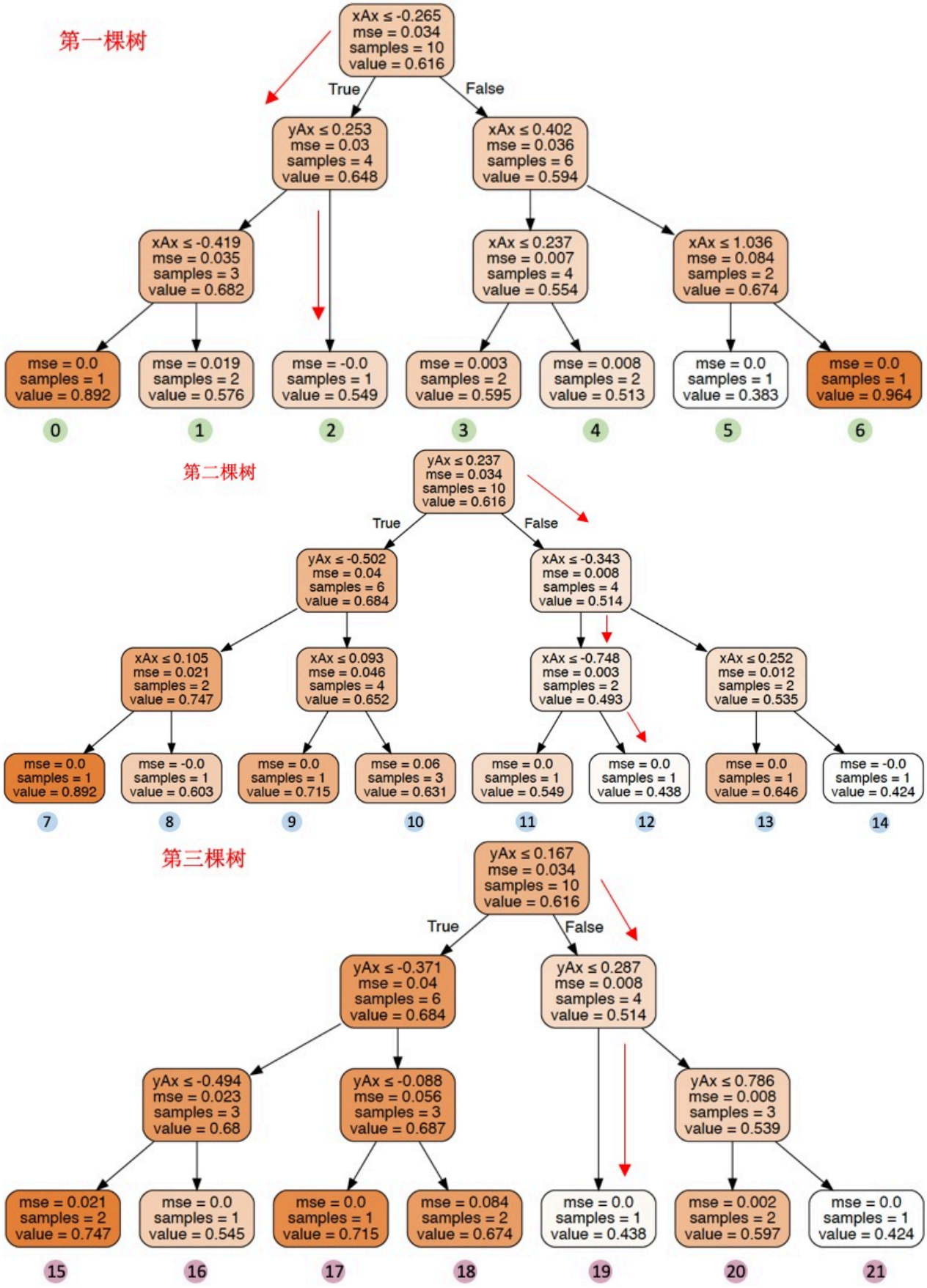
标签为：

```
[0 1 0 1 0 1 1 0 0 1]
```

生成的点如下图：



生成的三棵树的及其叶子的编码如下：



每个样本转换之后的独热编码大小为(10×22)，10 表示 10 个样本，22 表示三棵树一共有22个叶子节点，编号从第一棵树开始直到最后一棵树，结果如下：

$(0,2)$	1.0	$(3,10)$	1.0	$(6,19)$	1.0
$(0,11)$	1.0	$(3,16)$	1.0	$(7,0)$	1.0
$(0,20)$	1.0	$(4,4)$	1.0	$(7,7)$	1.0
$(1,1)$	1.0	$(4,14)$	1.0	$(7,15)$	1.0
$(1,9)$	1.0	$(4,21)$	1.0	$(8,6)$	1.0
$(1,17)$	1.0	$(5,3)$	1.0	$(8,10)$	1.0
$(2,4)$	1.0	$(5,13)$	1.0	$(8,18)$	1.0
$(2,8)$	1.0	$(5,20)$	1.0	$(9,5)$	1.0
$(2,15)$	1.0	$(6,1)$	1.0	$(9,10)$	1.0
$(3,3)$	1.0	$(6,12)$	1.0	$(9,18)$	1.0

以第七个样本带你为例，即`[-0.38885511 0.25118784]`，从上面可以看到他在三个树中的编码为：第一棵树`(6,1)`，即落在了第一个树的第二个叶子节点中、第二棵树`(6,12)`即落在了第二棵树的第 13 个节点中、第三棵树`(6,19)`即落在了第三棵树的第 20 个节点中。对应下图红色背景部分：

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20	21
0	0	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1	0
1	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0
2	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	1	0	0	0	0	0	0
3	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	1	0	0	0	0	0
4	0	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1
5	0	0	0	1	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1
6	0	1	0	0	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	1	0	0
7	1	0	0	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0	0	0	0
8	0	0	0	0	0	0	1	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0
9	0	0	0	0	0	1	0	0	0	0	1	0	0	0	0	0	0	0	1	0	0	0

上图的每一行代表了一个样本，每一列代表了一个树节点，值为 1 表示样本点落入了此节点中。统计每一列值为 1 的个数，就可以知道有多少样本落入了此节点，以编号为 10 的节点为例，可以看到有三个值为 1 ，表示有三个样本点落入了此节点。从生成的第二课节点树上我们可以看到，编号为 10 的节点树他的`samples = 3`。

通过上面的操作，就完成了样本映射到高维系数空间，很多算法对这些稀疏样本处理更高效。

总结

结合策略

常见的方法有三种方法：

1) 平均法（averaging），平均法又可以分为简单平均（simple averaging）和加权平均（weighted averaging）。

2) 投票法（voting），又可以分为绝大多数投票法（majority voting）、相对多数投票法（plurality voting）、加权投票法（weighted voting）

3) 学习法，最有名的是 stacking算法。

对比

- [sklearn - Plot the decision surfaces of ensembles of trees on the iris dataset]