GitHub https://github.com/kongzz311/MachineLearningNotes

假如有帮助欢迎加星

水平有限，若有错误欢迎指出：kozenzei@outlook.com

# 1. 集成学习学习笔记

## 1.1. Ensemble

**Motivations**

- 提高准确度
- 降低选择差模型的概率(Model Selection)

**常见**

- Bagging
- Boosting

**前提分类器不同**

- 不同的学习算法
- 不同的训练过程
    - 不同参数
    - 不同训练集
    - 不同特征集

**弱分类器**

- 容易产生不同的dicision boundaries
- Stumps...(树桩)

**怎样combine不同分类器**

- 平均
- 投票

    - Majority Voting
        - Random Forest
    - Weighted Majority Voting(加权)
        - AdaBoost
- Learning Combiner
    - General Combiner
        - Stacking
    - Piecewise Combiner
        - RegionBoost
- No Free Lunch

# 1.2. Bagging (Random Forests)

**Bootstrap Aggregation(Bagging)**

- 有放回的采样(Resample with Replacement)

对D采样K次独立训练分类器然后voting

**Random Forests**

- 有放回的采样(Resample with Replacement)
- 使用大约 2/3 的原始数据（why⬇️)

$$1-\lim_{n\to\infty}(1-\frac{1}{n})^n$$

- Majority Voting
- Number of Varibles

$$\sqrt{k}$$

- k = available variables
- Number of Trees
    - 500 or more
- Self-Testing
    - Around one third of the original data are left out.
    - Out if Bag(OOB)
    - Similar to Cross-Validation

优点

- All data can be used in the training process
    - Data in **OOB** are used to evaluate the current tree
- High levels of predictive accuracy
    - 只要调少量的参数，如树的数量
    - 分类和回归都可以
- Resistant to overfitting

- No need for prior feature selection

## 1.3. Stacking

先生成不同的分类器再在输出上进行学习**权重**（模型）

Base Classifiers —> Meta Classification

## 1.4. Boosting

**Boosting**

串行，先生成第一个C1，再在第一个的基础上生成后续模型

> 加入之前**分错**的数据进来

得到C2

> 加入分类器分类不一致的数据进来

得到C3

> 用来学习C1 和 C2 分类不同的

Boosting 和 bagging 的区别

Bagging aims at reducing **variance**, not **bias**.

In Boosting, classifiers are generated **sequentially**. Focuses on most informative data points. Training samples are **weighted**. Outputs are combined via **weighted** voting. Can create arbitrarily **strong** classifiers. The base learners can be arbitraily **weak**(比瞎猜好就可以). As long as they are better than random forests

## 1.5. AdaBoost

**Input:** Data set $D = \{(\boldsymbol{x}_1, y_1), (\boldsymbol{x}_2, y_2), \cdots, (\boldsymbol{x}_m, y_m)\}$;
Base learning algorithm $\mathcal{L}$;
Number of learning rounds $T$.

**Process:**

1. $\mathcal{D}_1(i) = 1/m$. % Initialize the weight distribution
2. **for** $t = 1, \cdots, T$:
3. $h_t = \mathcal{L}(D, \mathcal{D}_t)$; % Train a learner $h_t$ from $D$ using distribution $\mathcal{D}_t$
4. $\epsilon_t = \Pr_{\boldsymbol{x} \sim \mathcal{D}_t, y} \boldsymbol{I}[h_t(\boldsymbol{x}) \neq y]$; % Measure the error of $h_t$
5. **if** $\epsilon_t > 0.5$ **then break**
6. $\alpha_t = \frac{1}{2} \ln \left( \frac{1-\epsilon_t}{\epsilon_t} \right)$; % Determine the weight of $h_t$
7. $\mathcal{D}_{t+1}(i) = \frac{\mathcal{D}_t(i)}{Z_t} \times \begin{cases} \exp(-\alpha_t) & \text{if } h_t(\boldsymbol{x}_i) = y_i \\ \exp(\alpha_t) & \text{if } h_t(\boldsymbol{x}_i) \neq y_i \end{cases}$
$= \frac{\mathcal{D}_t(i) \exp(-\alpha_t y_i h_t(\boldsymbol{x}_i))}{Z_t}$ % Update the distribution, where
% $Z_t$ is a normalization factor which
% enables $\mathcal{D}_{t+1}$ to be a distribution
8. **end**

**Output:** $H(\boldsymbol{x}) = \text{sign} \left( \sum_{t=1}^{T} \alpha_t h_t(\boldsymbol{x}) \right)$

**它的自适应在于：前一个基本分类器分错的样本会得到加强，加权后的全体样本再次被用来训练下一个基本分类器。同时，在每一轮中加入一个新的弱分类器，直到达到某个预定的足够小的错误率或达到预先指定的最大迭代次数。可以认为是一种模型为加法模型、损失函数为指数函数、学习算法为前向分步算法的而分类学习方法**

可证明的结论：

1. 训练误差可以通过减小$Z_t$而减小，减小$Z_t$就是通过$\alpha = ln(\frac{1-\varepsilon}{\varepsilon})$ (有解析解)
2. 训练误差一定趋近0

优点

- 没什么参数要调整。一般设定50.
- 不要做特征删选
- 高精度

缺点

- $\alpha$局部最优，类似贪心($\sum z$)
- 易受噪声干扰
- 依赖于弱分类器的选择

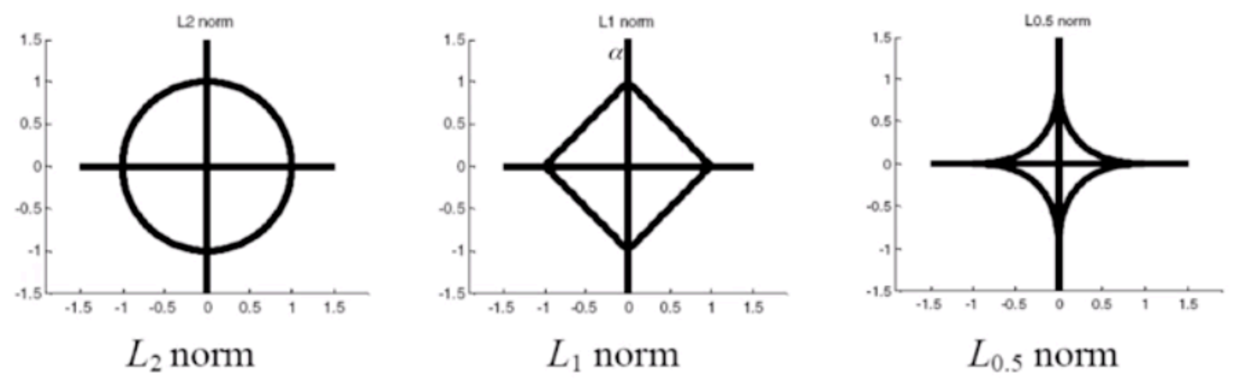
```
# AdaBoost Algorithm
from sklearn.ensemble import AdaBoostClassifier
clf = AdaBoostClassifier()
# n_estimators = 50 (default value)
# base_estimator = DecisionTreeClassifier (default value)
clf.fit(x_train,y_train)
clf.predict(x_test)
```

## 1.6. RegionBoost

动态调整权重（1998）

计算$\alpha_j(x_i)$

- 通过KNN
- 计算正确率




## 1.7. GBDT(Gradient Boost Decision Tree)

与AdaBoost不同，**GBDT每一次的计算是都为了减少上一次的残差，进而在残差减少（负梯度）的方向上建立一个新的模型。**GBDT与Adboost最主要的区别在于两者如何识别模型的问题。Adaboost用错分数据点来识别问题，通过调整错分数据点的权重来改进模型。GBDT通过负梯度来识别问题，通过计算负梯度来改进模型。

```
# Gradient Boosting
from sklearn.ensemble import GradientBoostingClassifier
clf = GradientBoostingClassifier()
# n_estimators = 100 (default)
# loss function = deviance(default) used in Logistic Regression
clf.fit(x_train,y_train)
clf.predict(x_test)
```

## 1.8. XGBoost

GBDT算法只利用了一阶的导数信息，xgboost对损失函数做了二阶的泰勒展开，并在目标函数之外加入了正则项对整体求最优解，用以权衡目标函数的下降和模型的复杂程度，避免过拟合。所以不考虑细节方面，两者最大的不同就是目标函数的定义。

```python
import xgboost as xgb
# read in data
dtrain = xgb.DMatrix('demo/data/agaricus.txt.train')
dtest = xgb.DMatrix('demo/data/agaricus.txt.test')
# specify parameters via map
param = {'max_depth':2, 'eta':1, 'silent':1, 'objective':'binary:logistic' }
num_round = 2
bst = xgb.train(param, dtrain, num_round)
# make prediction
preds = bst.predict(dtest)
```

›