机器学习笔记–随机森林和 *Bagging*

空修菜

1 装袋（bagging）

1.1 Bootstrap 和 Aggreation

1. “**bagging**”是“bootstrap aggreation”的简写.
2. Bootstrap 是对训练集进行抽样的方法. X 是训练集, 每次从样本中抽取 $|X|$ 个样本, 允许重复. 即有可能某个样本点被多次重复抽到, 而另外的样本点从未被抽到过.
3. 每当得到一个新样本, 就进行一次训练, 得到一个新模型.
4. Bootstrap 可以在一定程度上处理数据集不平衡的问题.
5. 通过 Bootstrap 得到的新数据集 $Z_i, i = 1, 2, \ldots, M$ 都有一个与之对应的模型（estimator）$G_i(\cdot)$. 此时，定义 **aggregate predictor** G 为:

$$G(X) = \sum_{i=1}^{M} \frac{G_i(x)}{M}.$$

   - 利用式子 (**??**), 有

$$\mathrm{Var}(\bar{G}) = \mathrm{Var}(G) = \rho\sigma^2 + \frac{1-\rho}{M}\sigma^2.$$

   - 从上式可知, 当增加模型数量的时候, 第二项分母增大, 所以模型的方差 $\mathrm{Var}(G)$ 就降低了, 因此过大的方差就被降低了.

6. bagging 加上决策树就得到了随机森林. 随机森林的“随机”体现在:
   - feature 的选择是随机的. 若有 p 个特征, 随机选取的特征数为 $\lfloor\sqrt{p}\rfloor$, 这就使得所训练的模型间的相关系数 ρ 变小;
   - 训练集的选择是随机的 (bootstrap).

7. 在 bagging 中,还有一个被称为“**out-of bag estimation**”的优点. bootstrap 得到的数据是随机抽取的，可以证明被抽到的样本点占到整个训练集 S 的 $\frac{2}{3}$, 所以，剩余的 $\frac{1}{3}$ 样本点就可以用来估计模型的误差，这个误差称为“袋外误差”(out of bag error).

1.2 随机森林（Random Forest）

1. 对于决策树来说, 存在偏差低, 方差高, 预测效果差的情况;
2. 对训练集 S 进行 bootstrap 后分别得到 m 个新的训练集 $Z_i, 1 \leq i \leq m$, 分别对每一个训练集 Z_i 训练一颗决策树 T_i, 总共就有 m 棵决策树. 这样就得到了一个森林.
3. 得到决策树集合 $\{T_i : i = 1, 2, \ldots, m\}$ 后, 最后的预测函数为 T:
$$T(x) = \frac{1}{m} \sum_{i=1}^{m} T_i(x).$$
4. 随机森林与决策树不同的体现在节点的分类标准的选择上. 对于每一棵树 T_i，在选择 feature 以及切分点点时候，并不全部考虑 p 个 features, 而是“随机地不重复”选取 p 个 feature 中的 t 个进行考虑 $(t < p)$, 一般地，取 $t = \lfloor \sqrt{p} \rfloor$(取小于等于 $\sqrt{p}$ 的最大整数).
5. 模型优点:
   - 降低了模型误差
   - 提高了准确性
   - 有额外的验证集 (set out of bag)
6. 模型缺点:
   - 偏差 bias 增加
   - 模型复杂更难解释
   - 耗时更长

1.3 Python Code for Decision Tree

随机森林等于 bagging 加上完全的（未剪枝）决策树. 将前一节决策树修改一下就可以用于生成森林. 需要修改的部分如下:

Listing 1: 随机特征选择函数

```
def BestFeatureSplit( self,X,y, feat_index_list ):
    bestSplitFea = None
    bestThre = None
    minGini = 1
    for i in feat_index_list:
       fea_i = X[ :,i ]
       for term_j in fea_i:
           threshold = term_j
           y_left = y[ fea_i <= threshold ]
           y_right = y[ fea_i > threshold ]
           if y_left.shape[ 0 ]==0 or
              y_right.shape[ 0 ]==0:
               continue

            GiniLeft = self.Gini( y_left )
            GiniRight = self.Gini( y_right )
            ConGini = (y_left.shape[ 0 ]*GiniLeft +
                       y_right.shape[0]*GiniRight)/
                       y.shape[ 0 ]

            if minGini > ConGini:
               minGini = ConGini
               bestSplitFea = i
               bestThre = threshold
   if bestSplitFea == None:
       bestSplitFea = np.random.choice(feat_index_list)
   X_col = X[ :,bestSplitFea ]
   if bestThre == None:
      bestThre = np.random.choice(X_col)
   X_left = X[ X_col <= bestThre,: ]
   X_right = X[ X_col > bestThre,: ]
   y_left = y[ X_col <= bestThre ]
   y_right = y[ X_col > bestThre]
   return (bestSplitFea,bestThre,
          X_left,y_left,X_right,y_right)
```

1. `BestFeatureSplit( X,y, feat_index_list )` 中的`feat_index_list` 是特征的索引, 用于确定需要考虑的特征;

2. `if bestSplitFea == None` 和 `if bestThre == None` 用于处理 Gini 为 0 时的情况. 即整个子集的类别都是相同的.

在决策树的函数`CreateTree( X,y,node )`中, 在使用特征分离函数前，需要确定随机的特征索引, 修改如下:

Listing 2: 修改后的 CreateTree 函数

```
feat_index_list = np.random.choice(X.shape[ 1 ],
                  int(np.sqrt(X.shape[ 1 ]+1)),
                  replace = False)
```

Listing 3: 随机森林

```
class RandomForestClassifier():
    def __init__(self,n_trees=15,max_depth = 200,
                 min_sample_leaf = 1,
                 min_sample_split = 2):
        self.n_trees = n_trees
        self.Trees = [ ]
        self.max_depth = max_depth
        self.min_sample_leaf = min_sample_leaf
        self.min_sample_split = min_sample_split

    def BootStrap(self,X,y):
        sample_num = X.shape[ 0 ]
        index = np.random.choice(sample_num,
                                 sample_num,
                                 replace=True)
        return X[index,:],y[index]

    def max_common_label(self,y):
        counter = Counter(y)
        return counter.most_common(1)[0][0]

    def fit(self,X,y):
        self.Trees = [ ]
        for i in range(self.n_trees):
            tree = DecisionTreeClassifier(
                   max_depth =self.max_depth,
                   min_sample_leaf = self.min_sample_leaf,
                   min_sample_split = self.min_sample_split)
```

```
            X_bootstrap,y_booststrap = self.BootStrap(X,y)
            tree.fit(X_bootstrap,y_booststrap)
            self.Trees.append(tree)

    def predict(self,X):
        tree_preds = np.array([tree.predict(X)
                    for tree in self.Trees])
        tree_preds = np.swapaxes(tree_preds,0,1)
        y_pred = [self.max_common_label(tree_pred)
                  for tree_pred in tree_preds]
        return np.asarray(y_pred)

if __name__ == "__main__":
    data=pd.read_csv("/Users/zzk/Desktop/M_Learning/
                    DATA/data.csv",
                    names = ['label','x1', 'x2'])
    X=data.values[:,1:]
    y=data.values[ :,0 ]
    y_val=np.where( y == 1,1,0 )
    (X_train, X_test, y_train, y_test) =
           train_test_split( X, y_val,
           test_size=0.2, random_state = 0)
    model = RandomForestClassifier(
            n_trees = 15,max_depth = 30,
            min_sample_leaf = 1,min_sample_split = 2)
    model.fit( X_train, y_train )
    y_pred = model.predict( X_test )
    print('模型的准确率是:',
          accuracy_score(y_test, y_pred))
```

1. 函数 BootStrap(self,X,y) 随机（重复）生成训练的样本.
2. 函数 max_common_label(self,y) 将占多数的类别确定为最终的类别.