XGBoost 实验

PB18071477 敖旭扬

原理

给定 n 个样本、m 个特征的二分类问题数据集 $D=\{(X_i,y_i)\}(|D|=n,X_i\in\mathbb{R}^m,y_i\in\{0,1\})$，`XGBoost` 模型采用 K 次迭代的结果作为输出结果。X_i 的输出为 $\hat{y_i}$，最终预测结果为 p_i，有

$$\hat{y_i}=\phi(X_i)=\sum_{k=1}^{K}f_k(X_i),\quad f_k\in F \tag{1}$$

$$p_i=round\Big(\frac{1}{1+e^{-\hat{y_i}}}\Big) \tag{2}$$

其中 $F=\{f(X)=w_{q(X)}\}(q:\mathbb{R}^m\to\{1,2,\dots,T\},w\in\mathbb{R}^T)$ 表示 `XGBoost` 树结构空间集，各个变量的含义如下：

- q 表示树的结构，可以将样本映射到相应的叶子节点；
- T 表示 `XGBoost` 树叶子节点的数量；
- 每个 f_k 都对应独立的树结构 q 和权重 w。

最终损失函数化简为

$$Obj^{(t)}=\sum_{j=1}^{T}[G_jw_j+\frac{1}{2}(H_j+\lambda)w_j^2]+\gamma T \tag{3}$$

其中

$$g_i=\frac{\partial l(y_i,\hat{y_i}^{(t-1)})}{\partial\hat{y_i}^{(t-1)}},\quad h_i=\frac{\partial^2 l(y_i,\hat{y_i}^{(t-1)})}{\partial^2\hat{y_i}^{(t-1)}} \tag{4}$$

$$I_j=\{i|q(X_i)=j\}$$

$$G_j=\sum_{i\in I_j}g_i,\quad H_j=\sum_{i\in I_j}h_i \tag{5}$$

$Obj^{(t)}$ 对 w_j 求导等于 0 可得

$$w_j^*=-\frac{G_j}{H_j+\lambda} \tag{6}$$

所以接下来需要不断寻找划分方式来构造出树的结构，可以对于数据集，遍历所有划分属性和分界点，求出

$$Gain=\frac{1}{2}\Big[\frac{G_L^2}{H_L+\lambda}+\frac{G_R^2}{H_R+\lambda}-\frac{(G_L+G_R)^2}{H_L+H_R+\lambda}\Big] \tag{7}$$

按照取得 $Gain$ 最大值的方式进行划分，再对划分后的两个子集继续按照这样的规则划分，同时树的层次增加 1。当树的层次达到最大值或当前集合小于最小划分块大小时停止划分得到一个叶子结点，按照 (6) 式计算出该叶子的权值。训练完当前树，更新 $\hat{y_i}$ 后再训练下一棵树，最终得到包含 K 课树的 `XGBoost` 模型，据此可用该模型对数据集进行预测，输出训练集和测试集的精度。

在本实现中，损失函数使用的是 $logistic$ 损失，即

$$l(y_i, \hat{y_i}) = y_i \ln(1 + e^{-\hat{y_i}}) + (1 - y_i) \ln(1 + e^{-\hat{y_i}}) \tag{8}$$

令

$$p_i = \frac{1}{1 + e^{-\hat{y_i}}} \tag{9}$$

则计算得

$$g_i = p_i - y_i, \quad h_i = p_i(1 - p_i) \tag{10}$$

编程实现

矩阵运算使用 `python` 的 `numpy` 库实现。

`XGBoost` 模型的训练和预测部分如下：

```python
class XGBoost:
    def fit(self, X, y):
        p = np.zeros(np.shape(y))
        for tree in self.trees:
            tree.fit(X, y, p)
            p += tree.predict(X)

    def predict(self, X):
        y_pred = np.zeros(X.shape[0])
        for tree in self.trees:
            y_pred += tree.predict(X)
        h = 1.0/(1.0+np.exp(-y_pred))
        return np.round(h)
```

单棵子树的训练和预测部分如下：

```python
class XGBoostTree:
    def fit(self, X, y, p):
        G = self.loss.grad(y, p)
        H = self.loss.hess(y, p)
        XypGH = np.c_[X, y, p, G, H]
        self.root = self.build_tree(XypGH)

    def build_tree(self, XypGH, cur_depth=1):
        max_gain = 0
        LXyp = None
        RXyp = None
        feature = None
        threshold = None
        split_point = None
        size = np.shape(XypGH)[0]
        d = np.shape(XypGH)[1] - 4  # 属性个数
        if size >= self.split_size and cur_depth <= self.max_depth:
            for f in range(d):
                XypGH = XypGH[np.lexsort(XypGH.T[f, None])]
                unique_values = np.unique(XypGH[:, f])
                for fv in unique_values:
                    split_i = np.searchsorted(XypGH[:, f], fv)
                    g_l = np.sum(XypGH[:split_i, -2])
                    g_r = np.sum(XypGH[split_i:, -2])
```

```
                    h_l = np.sum(XypGH[:split_i, -1])
                    h_r = np.sum(XypGH[split_i:, -1])
                    gain = 0.5*(np.square(g_l)/(h_l+self.la)
                                + np.square(g_r)/(h_r+self.la)
                                - np.square(g_l+g_r)/(h_l+h_r+self.la))
                    - self.gamma
                    if gain >= max_gain and gain > self.min_gain:
                        max_gain = gain
                        feature = f
                        threshold = fv
                        split_point = split_i
            if max_gain > self.min_gain:
                XypGH = XypGH[np.lexsort(XypGH.T[feature, None])]
                LXypGH = XypGH[:split_point]
                RXypGH = XypGH[split_point:]
                left = self.build_tree(LXypGH, cur_depth + 1)
                right = self.build_tree(RXypGH, cur_depth + 1)
                return self.Node(feature, threshold, left, right)
        g = np.sum(XypGH[:, -2])
        h = np.sum(XypGH[:, -1])
        leaf_value = self.eta*(-g / (h + self.la))
        return self.Node(value=leaf_value)

    def predict_value(self, x, tree):
        if tree.value is not None:
            return tree.value
        feature_value = x[tree.feature]
        if feature_value < tree.threshold:
            return self.predict_value(x, tree.left)
        else:
            return self.predict_value(x, tree.right)

    def predict(self, X):
        y_pred = []
        for x in X:
            y_pred.append(self.predict_value(x, self.root))
        p = np.array(y_pred)
        return p

```

完整实验源码见压缩包中的XGBoost.py。

运算结果

实例

在主函数中使用默认超参数(详见源码)调用下面的实例

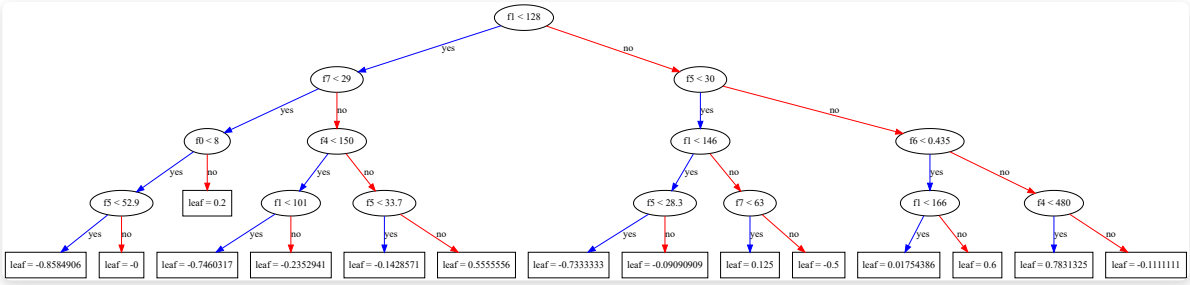
```python
# 训练模型
xgbt = XGBoost()
xgbt.fit(x_train, y_train)

# 训练集和测试集精度
p_train = xgbt.predict(x_train)
p_test = xgbt.predict(x_test)
print("训练集精度为：{:.2f} %".format(
    (p_train == y_train).mean() * 100))
print("测试集精度为：{:.2f} %".format(
    (p_test == y_test).mean() * 100))
```

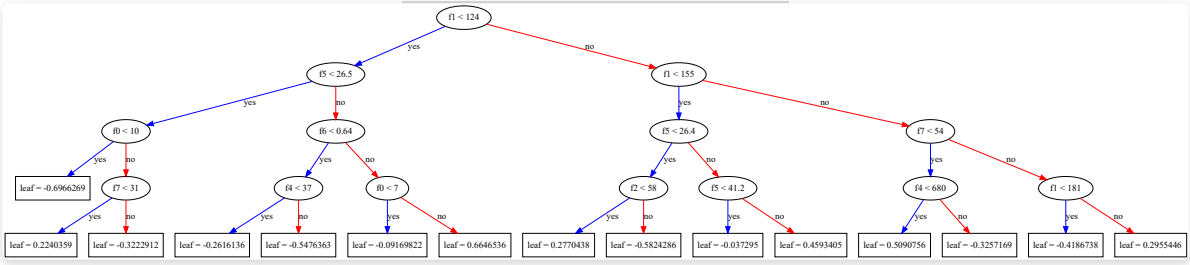
命令行输出结果为

```
训练集精度为：83.58 %
测试集精度为：76.47 %
```

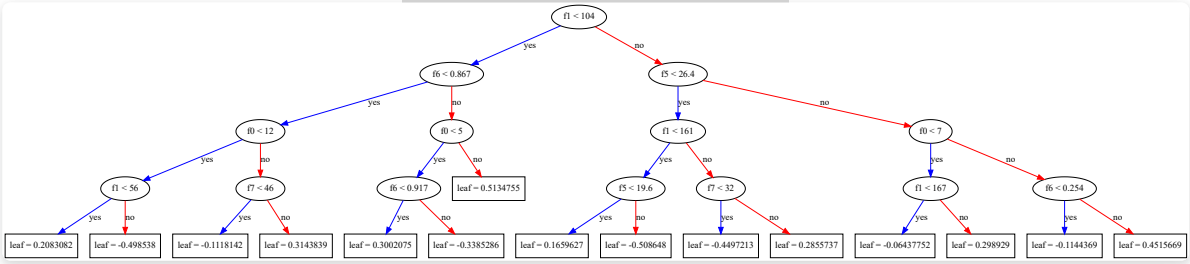
训练结果



XGBoost模型中的第1棵决策子树



XGBoost模型中的第2棵决策子树



XGBoost模型中的第3棵决策子树

预测效果

在测试集上的预测精度为 76.47% 。

总结

题目要求的 Baseline 为

```
测评指标：精度值，正确预测占整体的比例
测试集精度：0.7
```

我训练出的**XGBoost模型**测试集精度为 76.47%，性能达标。