Table of Contents

XGBoost

$$\begin{aligned} XGBoost &= eXtreme + GBDT \\ &= eXtreme + (Gradient + BDT) \\ &= eXtreme + Gradient + (Boosting + DecisionTree) \end{aligned}$$

$$Boosting \rightarrow BDT \rightarrow GBDT \rightarrow XGBoost$$

XGBoost原理

提升方法（Boosting）

提升方法使用加法模型和前向分步算法。

加法模型

$$f(x) = \sum_{m=1}^{M} \beta_m b(x; \gamma_m) \tag{1.1}$$

其中，$b(x; \gamma_m)$为基函数，γ_m为基函数的参数，β_m为基函数的系数。

在给定训练数据$\{(x_i, y_i)\}_{i=1}^{N}$及损失函数$L(y, f(x))$的条件下，学习加法模型$f(x)$成为经验风险极小化问题：

$$\min_{\beta_m \gamma_m} \sum_{i=1}^{N} L\left(y_i, \sum_{m=1}^{M} \beta_m b(x_i; \gamma_m)\right) \tag{1.2}$$

前向分步算法求解这一优化问题的思路：因为学习的是加法模型，可以从前向后，每一步只学习一个基函数及其系数，逐步逼近优化目标函数式（1.2），则可以简化优化复杂度。具体地，每步只需优化如下损失函数：

$$\min_{\beta,\gamma}\sum_{i=1}^{N}L\left(y_i,\beta b\left(x_i;\gamma\right)\right) \tag{1.3}$$

算法1.1 前向分步算法
输入：训练数据集$T=\left\{\left(x_1,y_1\right),\left(x_2,y_2\right),\ldots,\left(x_N,y_N\right)\right\}$；损失函数$L\left(y,f\left(x\right)\right)$；基函数集合$\left\{b\left(x;\gamma\right)\right\}$；
输出：加法模型$f\left(x\right)$
（1）初始化$f_0\left(x\right)=0$
（2）对$m=1,2,\ldots,M$
（a）极小化损失函数

$$\left(\beta_m,\gamma_m\right)=\arg\min_{\beta,\gamma}\sum_{i=1}^{N}L\left(y_i,f_{m-1}\left(x_i\right)+\beta b\left(x_i;\gamma\right)\right) \tag{1.4}$$

得到参数β_m，γ_m
（b）更新

$$f_m\left(x\right)=f_{m-1}\left(x\right)+\beta_m b\left(x;\gamma_m\right) \tag{1.5}$$

（3）得到加法模型

$$f\left(x\right)=f_M\left(x\right)=\sum_{m=1}^{M}\beta_m b\left(x;\gamma_m\right) \tag{1.6}$$

前向分步算法将同时求解从$m=1$到M所有参数β_m,γ_m的优化问题简化为逐次求解各个β_m,γ_m的优化问题。

提升决策树 （BDT，Boosting Decision Tree）

以决策树为基函数的提升方法为提升决策树。

提升决策树模型可以表示为决策树的加法模型：

$$f_M=\sum_{m=1}^{M}T\left(x;\Theta_m\right) \tag{2.1}$$

其中，$T\left(x;\Theta_m\right)$表示决策树；Θ_m为决策树的参数；M为树的个数。

提升决策树采用前向分步算法。首先确定初始提升决策树$f_0\left(x\right)=0$，第m步的模型是

$$f_m\left(x\right)=f_{m-1}\left(x\right)+T\left(x;\Theta_m\right) \tag{2.2}$$

其中，$f_{m-1}\left(x\right)$为当前模型，通过经验风险极小化确定下一棵决策树的参数Θ_m，

$$\hat{\Theta}_m=\arg\min_{\Theta_m}\sum_{i=1}^{N}L\left(y_i,f_{m-1}\left(x_i\right)+T\left(x_i;\Theta_m\right)\right) \tag{2.3}$$

已知训练数据集$T=\left\{\left(x_1,y_1\right),\left(x_2,y_2\right),\dots\left(x_N,y_N\right)\right\}$，$x_i\in\mathcal{X}\subseteq\mathbb{R}^n$，$\mathcal{X}$为输入空间，$y_i\in\mathcal{Y}\subseteq\mathbb{R}$，$\mathcal{Y}$为输出空间。如果将输入空间$\mathcal{X}$划分为$J$个互不相交的区域$R_1,R_2,\dots,R_J$，并且在每个区域上确定输出的常量$c_j$，那么决策树可表示为

$$T\left(x;\Theta\right)=\sum_{j=1}^{J}c_jI\left(x\in R_j\right)\tag{2.4}$$

其中，参数$\Theta=\left\{\left(R_1,c_1\right),\left(R_2,c_2\right),\dots,\left(R_J,c_J\right)\right\}$表示决策树的区域划分和各区域上的常量值。$J$是决策树的复杂度即叶子结点个数。

提升决策树使用以下前向分步算法：

$$\begin{aligned}&f_0\left(x\right)=0\\&f_m\left(x\right)=f_{m-1}\left(x\right)+T\left(x;\Theta_m\right),\quad m=1,2,\dots,M\\&f_M\left(x\right)=\sum_{m=1}^{M}T\left(x;\Theta_m\right)\end{aligned}$$

在前向分步算法的第m步，给定当前模型$f_{m-1}\left(x\right)$，需要求解

$$\hat{\Theta}_m=\arg\min_{\Theta_m}\sum_{i=1}^{N}L\left(y_i,f_{m-1}\left(x_i\right)+T\left(x_i;\Theta_m\right)\right)$$

得到$\hat{\Theta}_m$，即第m棵树的参数。

当采用平方误差损失函数时，

$$L\left(y,f\left(x\right)\right)=\left(y-f\left(x\right)\right)^2$$

其损失变为

$$\begin{aligned}L\left(y,f_{m-1}\left(x\right)+T\left(x;\Theta_m\right)\right)&=\left[y-f_{m-1}\left(x\right)-T\left(x;\Theta_m\right)\right]^2\\&=\left[r-T\left(x;\Theta_m\right)\right]^2\end{aligned}$$

其中，

$$r=y-f_{m-1}\left(x\right)\tag{2.5}$$

是当前模型拟合数据的残差（residual）。对回归问题的提升决策树，只需要简单地拟合当前模型的残差。

算法2.1 回归问题的提升决策树算法
输入：训练数据集$T=\left\{\left(x_1,y_1\right),\left(x_2,y_2\right),\dots,\left(x_N,y_N\right)\right\}$
输出：提升决策树$f_M\left(x\right)$
（1）初始化$f_0\left(x\right)=0$
（2）对$m=1,2,\dots,M$
（a）按照式（2.5）计算残差

$$r_{mi}=y_i-f_{m-1}\left(x_i\right),\quad i=1,2,\dots,N$$

(b)拟合残差r_{mi}学习一个回归树，得到$T\left(x;\Theta_m\right)$
（c）更新$f_m\left(x\right)=f_{m-1}\left(x\right)+T\left(x;\Theta_m\right)$
（3）得到回归提升决策树

$$f_M\left(x\right)=\sum_{m=1}^{M}T\left(x;\Theta_m\right)$$

梯度提升决策树 （GBDT，Gradient Boosting Decision Tree）

梯度提升算法使用损失函数的负梯度在当前模型的值

$$-\left[\frac{\partial L\left(y, f\left(x_{i}\right)\right)}{\partial f\left(x_{i}\right)}\right]_{f(x)=f_{m-1}(x)} \tag{3.1}$$

作为回归问题提升决策树算法中残差的近似值，拟合一个回归树。

算法3.1 梯度提升算法

输入：训练数据集$T=\left\{\left(x_{1}, y_{1}\right),\left(x_{2}, y_{2}\right), \ldots,\left(x_{N}, y_{N}\right)\right\}$，损失函数$L(y, f(x))$

输出：梯度提升决策树$\hat{f}(x)$

（1）初始化

$$f_{0}(x)=\arg \min _{c} \sum_{i=1}^{N} L\left(y_{i}, c\right)$$

（2）对$m=1,2, \ldots, M$

(a）对$i=1,2, \ldots, N$，计算

$$r_{m i}=-\left[\frac{\partial L\left(y, f\left(x_{i}\right)\right)}{\partial f\left(x_{i}\right)}\right]_{f(x)=f_{m-1}(x)}$$

(b)对$r_{m i}$拟合一个回归树，得到第m棵树的叶结点区域$R_{m j}, j=1,2, \ldots, J$

（c）对$j=1,2, \ldots, J$，计算

$$c_{m j}=\arg \min _{c} \sum_{x_{i} \in R_{m j}} L\left(y_{i}, f_{m-1}\left(x_{i}\right)+c\right)$$

（d）更新$f_{m}(x)=f_{m-1}(x)+\sum_{j=1}^{J} c_{m j} I\left(x \in R_{m j}\right)$

（3）得到回归梯度提升决策树

$$\hat{f}(x)=f_{M}(x)=\sum_{m=1}^{M} \sum_{j=1}^{J} c_{m j} I\left(x \in R_{m j}\right)$$

极限梯度提升（XGBoost，eXtreme Gradient Boosting）

训练数据集$\mathcal{D}=\left\{\left(\mathbf{x}_{i}, y_{i}\right)\right\}$，其中$\mathbf{x}_{i} \in \mathbb{R}^{m}, y_{i} \in \mathbb{R},|\mathcal{D}|=n$。

决策树模型

$$f(\mathbf{x})=w_{q(\mathbf{x})} \tag{4.1}$$

其中，$q: \mathbb{R}^{m} \rightarrow\{1, \ldots, T\}$是有输入$\mathbf{x}$向叶子结点编号的映射，$w \in \mathbb{R}^{T}$是叶子结点向量，$T$为决策树叶子节点数。

提升决策树模型预测输出

$$\hat{y}_{i}=\phi\left(\mathbf{x}_{i}\right)=\sum_{k=1}^{K} f_{k}\left(\mathbf{x}_{i}\right) \tag{4.2}$$

其中，$f_{k}(\mathbf{x})$为第k棵决策树。

正则化目标函数

$$\mathcal{L}(\phi)=\sum_{i} l\left(\hat{y}_{i}, y_{i}\right)+\sum_{k} \Omega\left(f_{k}\right) \tag{4.3}$$

其中，$\Omega(f)=\gamma T+\frac{1}{2} \lambda\|w\|^{2}=\gamma T+\frac{1}{2} \lambda \sum_{j=1}^{T} w_{j}^{2}$。

第t轮目标函数

$$\mathcal{L}^{(t)}=\sum_{i=1}^{n} l\left(y_i, \hat{y}_i^{(t-1)}+f_t\left(\mathbf{x}_i\right)\right)+\Omega\left(f_t\right) \tag{4.4}$$

第t轮目标函数$\mathcal{L}^{(t)}$在$\hat{y}^{(t-1)}$处的二阶泰勒展开

$$\begin{aligned}\mathcal{L}^{(t)} & \simeq \sum_{i=1}^{n}\left[l\left(y_i, \hat{y}^{(t-1)}\right)+\partial_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right) f_t\left(\mathbf{x}_i\right)+\frac{1}{2} \partial_{\hat{y}^{(t-1)}}^2 l\left(y_i, \hat{y}^{(t-1)}\right) f_t^2\left(\mathbf{x}_i\right)\right]+\Omega \\ & =\sum_{i=1}^{n}\left[l\left(y_i, \hat{y}^{(t-1)}\right)+g_i f_t\left(\mathbf{x}_i\right)+\frac{1}{2} h_i f_t^2\left(\mathbf{x}_i\right)\right]+\Omega\left(f_t\right)\end{aligned}$$

其中，$g_i=\partial_{\hat{y}^{(t-1)}} l\left(y_i, \hat{y}^{(t-1)}\right), h_i=\partial_{\hat{y}^{(t-1)}}^2 l\left(y_i, \hat{y}^{(t-1)}\right)$。

第t轮目标函数$\mathcal{L}^{(t)}$的二阶泰勒展开移除关于 $f_t\left(\mathbf{x}_i\right)$常数项

$$\begin{aligned}\tilde{\mathcal{L}}^{(t)} & =\sum_{i=1}^{n}\left[g_i f_t\left(\mathbf{x}_i\right)+\frac{1}{2} h_i f_t^2\left(\mathbf{x}_i\right)\right]+\Omega\left(f_t\right) \\ & =\sum_{i=1}^{n}\left[g_i f_t\left(\mathbf{x}_i\right)+\frac{1}{2} h_i f_t^2\left(\mathbf{x}_i\right)\right]+\gamma T+\frac{1}{2} \lambda \sum_{j=1}^{T} w_j^2\end{aligned} \tag{4.6}$$

定义叶结点j上的样本的下标集合$I_j=\{i \mid q\left(\mathbf{x}_i\right)=j\}$，则目标函数可表示为按叶结点累加的形式

$$\tilde{\mathcal{L}}^{(t)}=\sum_{j=1}^{T}\left[\left(\sum_{i \in I_j} g_i\right) w_j+\frac{1}{2}\left(\sum_{i \in I_j} h_i+\lambda\right) w_j^2\right]+\gamma T \tag{4.7}$$

由于

$$w_j^*=\underset{w_j}{\arg \min } \tilde{\mathcal{L}}^{(t)}$$

可令

$$\frac{\partial \tilde{\mathcal{L}}^{(t)}}{\partial w_j}=0$$

得到每个叶结点j的最优分数为

$$w_j^*=-\frac{\sum_{i \in I_j} g_i}{\sum_{i \in I_j} h_i+\lambda} \tag{4.8}$$

代入每个叶结点j的最优分数，得到最优化目标函数值

$$\tilde{\mathcal{L}}^{(t)}(q)=-\frac{1}{2} \sum_{j=1}^{T} \frac{\left(\sum_{i \in I_j} g_i\right)^2}{\sum_{i \in I_j} h_i+\lambda}+\gamma T \tag{4.9}$$

假设I_L和I_R分别为分裂后左右结点的实例集，令$I=I_L \cup I_R$，则分裂后损失减少量由下式得出

$$\mathcal{L}_{split}=\frac{1}{2}\left[\frac{\left(\sum_{i \in I_L} g_i\right)^2}{\sum_{i \in I_L} h_i+\lambda}+\frac{\left(\sum_{i \in I_R} g_i\right)^2}{\sum_{i \in I_R} h_i+\lambda}-\frac{\left(\sum_{i \in I} g_i\right)^2}{\sum_{i \in I} h_i+\lambda}\right]-\gamma \tag{4.10}$$

用以评估待分裂结点。

算法4.1 分裂查找的精确贪婪算法
输入：当前结点实例集I;特征维度d
输出：根据最大分值分裂
（1）$gain \leftarrow 0$
（2）$G \leftarrow \sum_{i \in I} g_i$，$H \leftarrow \sum_{i \in I} h_i$
（3）for $k = 1$ to d do
（3.1）$G_L \leftarrow 0$，$H_L \leftarrow 0$
（3.2）for j in sorted(I, by $\mathbf{x}_{jk}$) do
（3.2.1）$G_L \leftarrow G_L + g_j$，$H_L \leftarrow H_L + h_j$
（3.2.2）$G_R \leftarrow G - G_L$，$H_R = H - H_L$
（3.2.3）$score \leftarrow \max\left(score, \frac{G_L^2}{H_L+\lambda} + \frac{G_R^2}{H_R+\lambda} - \frac{G^2}{H+\lambda}\right)$
（3.3）end
（4）end

XGBoost应用

XGBoost参数

XGBoost主要参数类型：

1. 通用参数：控制整体功能；
2. 提升器参数：在每一步控制单个提升器（tree、regression）；
3. 学习任务参数：控制最优化执行。

通用参数

booster [default=gbtree]
选择每次迭代的模型，有两个选择：

- gbtree：基于树的模型；
- gbliner：线性模型。

silent [default=0]

- 设置为1，静默模式被开启，不会显示运行信息；
- 通常设置为0，运行信息会更好的帮助理解模型。

nthread [default=最大可能的线程数]

- 该参数用以并行处理，应设置为系统内核数；
- 如果你希望使用所有内核，则不应设置该参数，算法会自动检测。

提升器参数

eta [default=0.3]

- 学习率；
- 典型值：0.01-0.2。

min_child_weight [default=1]

- 定义最小叶子节点样本权重和；
- 用于控制过拟合。较大的值可以避免模型学习到局部的特殊样本；
- 太大的值会导致欠拟合。

max_depth [default=6]

- 树的最大深度；
- 用于控制过拟合。较大的值模型会学到更具体更局部的样本；
- 典型值为3-10。

max_leaf_nodes

- 树中终端节点或叶子的最大数目；
- 可以代替max_depth参数。由于创建的是二叉树，一个深度为n的树最多生成2^n个叶子；
- 如果该参数被定义，max_depth参数将被忽略。

gamma [default=0]

- 只有在节点分裂后损失函数值下降，才会分裂该节点。gamma参数指定了节点分裂所需的最小损失函数下降值；
- 该参数的值越大，算法越保守。该参数的值和损失函数相关，所以是需要调整的。

max_delta_step [default=0]

- 该参数限制每棵树权重改变的最大步长。如果该参数为0，则表示没有约束。如果将其设置为正值，则使更新步骤更加保守；
- 通常该参数不需要设置。但是当各类别的样本十分不平衡时，它对逻辑回归是很有帮助的。

subsample [default=1]

- 该参数控制对于每棵树随机采样的比例；
- 减小该参数的值，算法会更加保守，避免过拟合。但是，如果该设置得过小，它可能会导致欠拟合；
- 典型值：0.5-1。

colsample_bytree [default=1]

- 该参数用来控制每棵随机采样的列数的占比(每一列是一个特征)；
- 典型值：0.5-1。

colsample_bylevel [default=1]

- 该参数用来控制树的每一级的每一次分裂，对列数的采样的占比；
- 该参数和subsample参数可以起到相同的作用。

lambda [default=1]

- 权重的L2正则化项。(类似于岭回归)。

alpha [default=0]

- 权重的L1正则化项。(类似于套索回归);
- 可以应用在很高维度的情况下，使得算法的速度更快。

scale_pos_weight [default=1]

- 在各类别样本十分不平衡时，把这个参数设定为一个正值，可以使算法更快收敛。

学习任务参数

objective [default=reg:linear]
该参数定义需要被最小化的损失函数。常用值有：

- binary:logistic 二分类的逻辑回归，返回预测的概率(不是类别);
- multi:softmax 使用softmax的多分类器，返回预测的类别(不是概率)。在这种情况下，你还需要多设一个参数：num_class(类别数目);
- multi:softprob 和multi:softmax参数一样，但是返回的是每个数据属于各个类别的概率。

eval_metric [default according to objective]

- 对于有效数据的度量方法;
- 对于回归问题，默认值是rmse，对于分类问题，默认值是error;
- 典型值：
  - rmse 均方根误差
  - mae 平均绝对误差
  - logloss 负对数似然函数值
  - error 二分类错误率(阈值为0.5)
  - merror 多分类错误率
  - mlogloss 多分类logloss损失函数
  - auc 曲线下面积

seed [default=0]

- 随机数的种子
- 设置它可以复现随机数据的结果，也可以用于调整参数

XGBoost的基本使用应用

导入XGBoost等相关包：

In [3]:

```
from numpy import loadtxt
from xgboost import XGBClassifier
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score

import os
os.environ['KMP_DUPLICATE_LIB_OK']='True'
```

加载数据，提取特征集和标签：

In [4]:

```
dataset = loadtxt('./data/pima-indians-diabetes.csv', delimiter=',')

X = dataset[:, 0:8]
y = dataset[:, 8]
```

将数据划分为训练集和测试集:

In [5]:

```
seed = 7
test_size = 0.33
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=test_size, random_state=seed
)
```

In [6]:

```
X_train.shape, X_test.shape, y_train.shape, y_test.shape
```

Out[6]:

```
((514, 8), (254, 8), (514,), (254,))
```

创建及训练模型:

In [7]:

```
model = XGBClassifier(n_jobs=-1)
model.fit(X_train, y_train)
```

Out[7]:

```
XGBClassifier(base_score=0.5, booster='gbtree', colsample_bylevel=1,
       colsample_bytree=1, gamma=0, learning_rate=0.1, max_delta_step=0,
       max_depth=3, min_child_weight=1, missing=None, n_estimators=100,
       n_jobs=-1, nthread=None, objective='binary:logistic',
       random_state=0, reg_alpha=0, reg_lambda=1, scale_pos_weight=1,
       seed=None, silent=True, subsample=1)
```

使用训练后的模型对测试集进行预测，并计算预测值与实际之间的acc值:

In [8]:

```
y_pred = model.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 77.95%
```

使用训练后的模型对测试集进行预测，得到每个类别的预测概率:

In [9]:

```
y_pred = model.predict(X_test)
y_pred
```

Out[9]:

```
array([0., 1., 1., 0., 1., 1., 0., 0., 1., 0., 1., 0., 1., 1., 0., 0., 0.,
       1., 0., 0., 0., 0., 1., 1., 0., 0., 0., 0., 0., 1., 1., 0., 0., 0.,
       0., 1., 1., 0., 1., 0., 1., 1., 1., 0., 0., 0., 1., 0., 0., 1., 0.,
       0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 1., 1., 1., 0., 1.,
       1., 1., 1., 1., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0., 0.,
       0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 1., 1., 0., 1., 0., 1., 0.,
       0., 0., 0., 1., 0., 0., 0., 1., 0., 1., 0., 0., 1., 1., 0., 0., 0.,
       1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0., 0., 1., 0., 0., 0., 0.,
       0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 0., 1., 0., 0., 1., 0.,
       0., 0., 0., 1., 0., 1., 0., 0., 0., 0., 0., 1., 0., 1., 0., 0., 1.,
       0., 1., 0., 0., 1., 0., 1., 0., 1., 0., 1., 0., 0., 0., 0., 0., 0.,
       0., 0., 1., 1., 0., 1., 1., 0., 0., 1., 0., 0., 1., 0., 1., 1., 1.,
       0., 0., 1., 0., 0., 0., 1., 1., 0., 1., 0., 0., 0., 1., 0., 0., 0.,
       0., 1., 0., 1., 1., 1., 1., 1., 0., 0., 1., 0., 0., 0., 0., 0., 1.,
       0., 0., 1., 1., 0., 0., 1., 0., 1., 0., 0., 0., 0., 1., 1., 1.])
```

In [10]:

```
y_pred_proba = model.predict_proba(X_test)
y_pred_proba
```

Out[10]:

```
array([[0.9545844 , 0.04541559],
       [0.05245447, 0.9475455 ],
       [0.41897488, 0.5810251 ],
       [0.9831998 , 0.0168002 ],
       [0.4119159 , 0.5880841 ],
       [0.31113452, 0.6888655 ],
       [0.9705527 , 0.02944732],
       [0.93274003, 0.06725994],
       [0.11494881, 0.8850512 ],
       [0.6501156 , 0.34988442],
       [0.03848034, 0.96151966],
       [0.99019825, 0.00980172],
       [0.07478714, 0.92521286],
       [0.0899508 , 0.9100492 ],
       [0.8759558 , 0.12404419],
       [0.8833156 , 0.11668438],
       [0.7805242 , 0.21947582],
       [0.35131902, 0.648681  ],
       [0.98205894, 0.01794105],
       [0.7698676 , 0.23013239],
       [0.6992918 , 0.30070814],
       [0.79318744, 0.20681255],
       [0.3731498 , 0.6268502 ],
       [0.23303777, 0.76696223],
       [0.9244116 , 0.07558843],
       [0.93374586, 0.06625411],
       [0.9396339 , 0.06036608],
       [0.5944243 , 0.4055757 ],
       [0.5389885 , 0.46101153],
       [0.4979669 , 0.5020331 ],
       [0.3711158 , 0.6288842 ],
       [0.9689762 , 0.03102378],
       [0.6525316 , 0.34746838],
       [0.6155367 , 0.3844633 ],
       [0.9903705 , 0.00962946],
       [0.49918646, 0.50081354],
       [0.44489336, 0.55510664],
       [0.7627247 , 0.23727532],
       [0.31710714, 0.68289286],
       [0.77836037, 0.22163963],
       [0.26101363, 0.7389864 ],
       [0.13494265, 0.86505735],
       [0.35929406, 0.64070594],
       [0.9963932 , 0.00360681],
       [0.6170161 , 0.38298395],
       [0.92774606, 0.07225396],
       [0.15301538, 0.8469846 ],
       [0.60130596, 0.39869407],
       [0.79963815, 0.20036186],
       [0.33829194, 0.66170806],
       [0.9839194 , 0.0160806 ],
       [0.9502892 , 0.04971079],
       [0.5472499 , 0.4527501 ],
       [0.9277443 , 0.07225566],
       [0.5314773 , 0.46852273],
       [0.613425  , 0.386575  ],
       [0.7924037 , 0.2075963 ],
       [0.9950148 , 0.00498524],
       [0.7400504 , 0.25994962],
```

```
[0.749115  , 0.25088504],
[0.98180836, 0.01819165],
[0.7836989 , 0.2163011 ],
[0.9104567 , 0.08954328],
[0.19470114, 0.80529886],
[0.41110873, 0.58889127],
[0.13466156, 0.86533844],
[0.9739846 , 0.02601542],
[0.40842408, 0.5915759 ],
[0.03246957, 0.9675304 ],
[0.40166223, 0.59833777],
[0.04176617, 0.95823383],
[0.09635341, 0.9036466 ],
[0.7429416 , 0.2570584 ],
[0.66398585, 0.33601415],
[0.69872403, 0.30127597],
[0.7699315 , 0.23006849],
[0.8938688 , 0.10613122],
[0.74650097, 0.25349906],
[0.99032426, 0.00967571],
[0.810189  , 0.18981098],
[0.89962673, 0.10037328],
[0.9574254 , 0.0425746 ],
[0.772874  , 0.227126  ],
[0.98344105, 0.01655894],
[0.5029772 , 0.4970228 ],
[0.5691345 , 0.43086553],
[0.01497293, 0.9850271 ],
[0.85218066, 0.14781934],
[0.12034631, 0.8796537 ],
[0.9485912 , 0.05140885],
[0.17748964, 0.82251036],
[0.98293597, 0.01706405],
[0.36969757, 0.6303024 ],
[0.7486875 , 0.2513125 ],
[0.9288969 , 0.07110308],
[0.4744296 , 0.5255704 ],
[0.45320487, 0.5467951 ],
[0.9389031 , 0.06109691],
[0.4675269 , 0.5324731 ],
[0.5375776 , 0.46242237],
[0.3889646 , 0.6110354 ],
[0.91216415, 0.08783586],
[0.88084364, 0.11915639],
[0.83369595, 0.16630404],
[0.93547785, 0.06452216],
[0.4981265 , 0.5018735 ],
[0.7939997 , 0.20600031],
[0.7923043 , 0.20769574],
[0.9111986 , 0.08880138],
[0.22100538, 0.7789946 ],
[0.98653555, 0.01346444],
[0.45952702, 0.540473  ],
[0.9085744 , 0.09142561],
[0.9781135 , 0.02188653],
[0.22032976, 0.77967024],
[0.44598758, 0.5540124 ],
[0.9262883 , 0.07371172],
[0.92085296, 0.07914706],
[0.66331017, 0.33668986],
[0.13952333, 0.8604767 ],
```

```
[0.81041753, 0.18958248],
[0.9896193 , 0.01038068],
[0.96018237, 0.03981761],
[0.9959437 , 0.0040563 ],
[0.97554094, 0.02445907],
[0.03378391, 0.9662161 ],
[0.99607235, 0.00392762],
[0.7080204 , 0.2919796 ],
[0.9855419 , 0.01445814],
[0.96412593, 0.03587409],
[0.9733594 , 0.02664059],
[0.08533424, 0.91466576],
[0.97558296, 0.02441705],
[0.8622012 , 0.13779879],
[0.8407246 , 0.1592754 ],
[0.9961686 , 0.00383137],
[0.9785708 , 0.0214292 ],
[0.5760479 , 0.4239521 ],
[0.99518293, 0.00481707],
[0.0550918 , 0.9449082 ],
[0.85514486, 0.14485516],
[0.3612969 , 0.6387031 ],
[0.5012821 , 0.49871793],
[0.5099146 , 0.49008542],
[0.8806706 , 0.11932941],
[0.7325761 , 0.26742393],
[0.52149546, 0.47850457],
[0.973495  , 0.02650498],
[0.04164171, 0.9583583 ],
[0.61220694, 0.38779306],
[0.9879225 , 0.01207752],
[0.25952768, 0.7404723 ],
[0.8303682 , 0.16963178],
[0.92665327, 0.07334672],
[0.7389723 , 0.26102766],
[0.92206323, 0.07793678],
[0.14965522, 0.8503448 ],
[0.8748666 , 0.12513341],
[0.33422202, 0.665778  ],
[0.8762611 , 0.12373889],
[0.7751093 , 0.22489071],
[0.93006825, 0.06993173],
[0.9858091 , 0.01419094],
[0.9868426 , 0.01315743],
[0.19853216, 0.80146784],
[0.72698987, 0.2730101 ],
[0.35169834, 0.64830166],
[0.9212139 , 0.07878608],
[0.76613265, 0.23386733],
[0.25219387, 0.74780613],
[0.98060507, 0.01939493],
[0.23698407, 0.7630159 ],
[0.55401087, 0.44598913],
[0.58738816, 0.4126118 ],
[0.33069748, 0.6693025 ],
[0.6730691 , 0.32693088],
[0.10515052, 0.8948495 ],
[0.82272303, 0.17727698],
[0.48799843, 0.5120016 ],
[0.5444821 , 0.4555179 ],
[0.27095395, 0.72904605],
```

```
[0.7812414 , 0.21875861],
[0.9742987 , 0.02570128],
[0.5995555 , 0.40044448],
[0.6039266 , 0.3960734 ],
[0.7065661 , 0.2934339 ],
[0.994187  , 0.00581301],
[0.9957897 , 0.00421028],
[0.994171  , 0.00582896],
[0.07712632, 0.9228737 ],
[0.25683784, 0.74316216],
[0.7415335 , 0.25846648],
[0.11841422, 0.8815858 ],
[0.09231913, 0.90768087],
[0.55010295, 0.44989708],
[0.52260685, 0.47739318],
[0.18215257, 0.81784743],
[0.997291  , 0.002709  ],
[0.9826735 , 0.01732648],
[0.37172633, 0.62827367],
[0.565622  , 0.43437806],
[0.46577197, 0.534228  ],
[0.01722312, 0.9827769 ],
[0.07406062, 0.9259394 ],
[0.6765144 , 0.3234856 ],
[0.96793205, 0.03206796],
[0.24381787, 0.75618213],
[0.99477583, 0.00522418],
[0.5577092 , 0.44229078],
[0.9967807 , 0.00321933],
[0.08804113, 0.9119589 ],
[0.04947805, 0.95052195],
[0.8901977 , 0.10980231],
[0.48584348, 0.5141565 ],
[0.6006367 , 0.39936325],
[0.98051137, 0.01948861],
[0.90228915, 0.09771082],
[0.10026193, 0.8997381 ],
[0.9786447 , 0.02135527],
[0.7318948 , 0.2681052 ],
[0.9847726 , 0.01522739],
[0.541728  , 0.45827198],
[0.2931553 , 0.7068447 ],
[0.64007187, 0.35992816],
[0.2672006 , 0.7327994 ],
[0.2482081 , 0.7517919 ],
[0.40669525, 0.59330475],
[0.34259337, 0.6574066 ],
[0.20737344, 0.79262656],
[0.95063394, 0.04936607],
[0.7813139 , 0.21868612],
[0.21015525, 0.78984475],
[0.7676461 , 0.23235396],
[0.8140151 , 0.18598492],
[0.9709272 , 0.02907283],
[0.95962995, 0.04037007],
[0.99101746, 0.00898254],
[0.21600759, 0.7839924 ],
[0.54862547, 0.45137453],
[0.8545133 , 0.14548668],
[0.36117506, 0.63882494],
[0.43292588, 0.5670741 ],
```

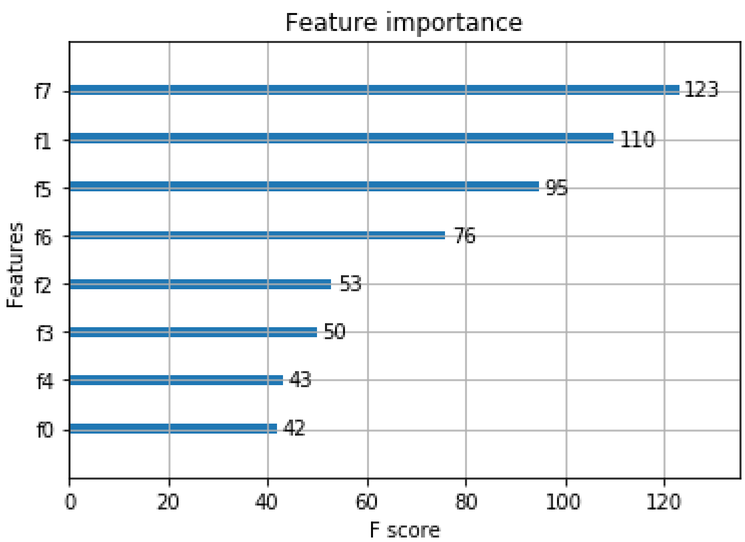
```
       [0.6847279 , 0.31527206],
       [0.89733434, 0.10266564],
       [0.10231268, 0.8976873 ],
       [0.9843091 , 0.01569091],
       [0.45564342, 0.5443566 ],
       [0.9380262 , 0.06197384],
       [0.82817245, 0.17182757],
       [0.7848037 , 0.2151963 ],
       [0.97062194, 0.02937807],
       [0.42821795, 0.57178205],
       [0.2364142 , 0.7635858 ],
       [0.05780089, 0.9421991 ]], dtype=float32)
```

输出各特征重要程度:

In [11]:

```
from xgboost import plot_importance
from matplotlib import pyplot
%matplotlib inline

plot_importance(model)
pyplot.show()
```



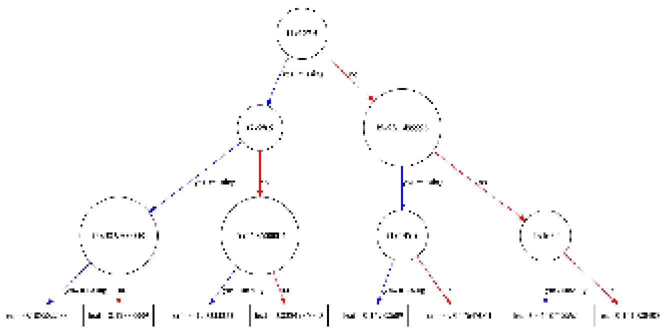
In [12]:

```
from xgboost import plot_tree

plot_tree(model)
```

Out[12]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a23d5db70>
```



导入调参相关包：

In [13]:

```
from sklearn.model_selection import GridSearchCV
from sklearn.model_selection import StratifiedKFold
```

创建模型及参数搜索空间：

In [14]:

```
model_GS = XGBClassifier()
learning_rate = [0.0001, 0.001, 0.01, 0.1, 0.2, 0.3]
max_depth = [1, 2, 3, 4, 5]
param_grid = dict(learning_rate=learning_rate, max_depth=max_depth)
```

设置分层抽样验证及创建搜索对象：

In [15]:

```
kfold = StratifiedKFold(n_splits=10, shuffle=True, random_state=seed)
grid_search = GridSearchCV(model_GS, param_grid=param_grid, scoring='neg_log_loss', n_jobs=-1, c
v=kfold)
grid_result = grid_search.fit(X, y)
```

In [16]:

```
y_pred = grid_result.predict(X_test)
accuracy = accuracy_score(y_test, y_pred)
print("Accuracy: %.2f%%" % (accuracy * 100.0))
```

```
Accuracy: 81.10%
```

In [17]:

```
grid_result.best_score_, grid_result.best_params_
```

Out[17]:

```
(-0.47171179660714796, {'learning_rate': 0.2, 'max_depth': 1})
```

XGBoost与LightGBM的对比分析

In [18]:

```
import time

import numpy as np
import pandas as pd
import matplotlib as mpl
import matplotlib.pyplot as plt
from matplotlib.pylab import rcParams
import seaborn as sns

from sklearn.metrics import accuracy_score
from sklearn.model_selection import GridSearchCV

import xgboost as xgb
from xgboost import XGBClassifier

import lightgbm as lgb
from lightgbm import LGBMClassifier
```

In [19]:

```
fetch_from = './data/fashionmnist/fashion-mnist_train.csv'
train = pd.read_csv(fetch_from)

fetch_from = './data/fashionmnist/fashion-mnist_test.csv'
test = pd.read_csv(fetch_from)
```

In [20]:

```
X_train, y_train, X_test, y_test = train.iloc[:, 1:], train['label'], test.iloc[:, 1:], test['la
bel']
X_train.head()
```

Out[20]:

	pixel1	pixel2	pixel3	pixel4	pixel5	pixel6	pixel7	pixel8	pixel9	pixel10	...	pixel7
0	0	0	0	0	0	0	0	0	0	0	...	0
1	0	0	0	0	0	0	0	0	0	0	...	0
2	0	0	0	0	0	0	0	5	0	0	...	0
3	0	0	0	1	2	0	0	0	0	0	...	3
4	0	0	0	0	0	0	0	0	0	0	...	0

5 rows × 784 columns

In [21]:

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[21]:

```
((60000, 784), (60000,), (10000, 784), (10000,))
```

In [22]:

```
def plot_digits(instances, images_per_row=10, **options):
    size = 28
    images_per_row = min(len(instances), images_per_row)
    images = [instance.reshape(size, size) for instance in instances]
    n_rows = (len(instances) -1) // images_per_row + 1
    row_images = []
    n_empty = images_per_row * n_rows - len(instances)
    images.append(np.zeros((size, size * n_empty)))
    for row in range(n_rows):
        rimages = images[row * images_per_row : (row + 1) * images_per_row]
        row_images.append(np.concatenate(rimages, axis=1))
    image = np.concatenate(row_images, axis=0)
    plt.imshow(image, cmap=mpl.cm.binary, **options)
    plt.axis("off")
```

In [23]:

```
plt.figure(figsize=(10,10))
example_images = X_train[:100]
plot_digits(example_images.values)
plt.show()
```

In [24]:

```
def show_time(diff):
    m, s = divmod(diff, 60)
    h, m = divmod(m, 60)
    s, m, h = int(round(s, 0)), int(round(m, 0)), int(round(h, 0))

    print("Execution Time: " + "{0:02d}:{1:02d}:{2:02d}".format(h, m, s))
```

In [25]:

```
training_times = []
testing_times = []
scores = []
```

In [26]:

```
def training_and_testing(clf, X, y, X_test, y_test):
    print("Training...")
    start = time.time()
    model = clf.fit(X, y)
    end = time.time()
    training_times.append(end - start)
    show_time(end - start)


    print("\nTesting...")
    start = time.time()
    scores.append(accuracy_score(y_test, model.predict(X_test)))
    end = time.time()
    testing_times.append(end - start)
    show_time(end - start)

    return model
```

In [34]:

```
xgb_model = training_and_testing(XGBClassifier(n_estimators=50, max_depth=5), X_train, y_train,
X_test, y_test)
```

```
Training...
Execution Time: 00:21:04

Testing...
Execution Time: 00:00:00
```

In [35]:

```
lgb_model = training_and_testing(LGBMClassifier(n_estimators=50, max_depth=5), X_train, y_train,
X_test, y_test)
```

```
Training...
Execution Time: 00:01:46

Testing...
Execution Time: 00:00:01
```

In [27]:

```
def training_and_testing_with_grid_search(clf, params, X, y, X_test, y_test):
    print("Training with Grid Search...")
    start = time.time()
    model = GridSearchCV(clf, params, scoring='accuracy', n_jobs=-1, cv=5).fit(X,y).best_estimat
or_
    end = time.time()
    training_times.append(end - start)
    show_time(end - start)


    print("Testing with Grid Search...")
    start = time.time()
    scores.append(accuracy_score(y_test, model.predict(X_test)))
    end = time.time()
    testing_times.append(end - start)
    show_time(end - start)

    return model
```

In [38]:

```
param_grid = [{'max_depth': [5,10],
               'n_estimators': [100],
               'learning_rate': [0.05, 0.1],
               'colsample_bytree': [0.8, 0.95]}]

xgb_model_gs = training_and_testing_with_grid_search(XGBClassifier(random_state=42), param_grid,
                                                     X_train[:4000], y_train[:4000], X_test, y_test)
```

```
Training with Grid Search...

/anaconda3/envs/dev/lib/python3.6/site-packages/sklearn/externals/joblib/external
s/loky/process_executor.py:706: UserWarning: A worker stopped while some jobs were
given to the executor. This can be caused by a too short worker timeout or by a me
mory leak.
  "timeout or by a memory leak.", UserWarning

Execution Time: 00:45:16
Testing with Grid Search...
Execution Time: 00:00:01
```

In [37]:

```
lgb_model_gs = training_and_testing_with_grid_search(LGBMClassifier(random_state=42), param_grid
,
                                                     X_train[:4000], y_train[:4000], X_test, y_test)
```

```
Training with Grid Search...
Execution Time: 00:17:14
Testing with Grid Search...
Execution Time: 00:00:01
```

In [39]:

```
scores, training_times, testing_times
```

Out[39]:

```
([0.8716, 0.8717, 0.8509, 0.8476],
 [1264.2565653324127,
  106.44469475746155,
  1033.7386996746063,
  2716.4520568847656],
 [0.42176175117492676,
  0.5533642768859863,
  1.329496145248413,
  0.7895469665527344])
```

In [41]:

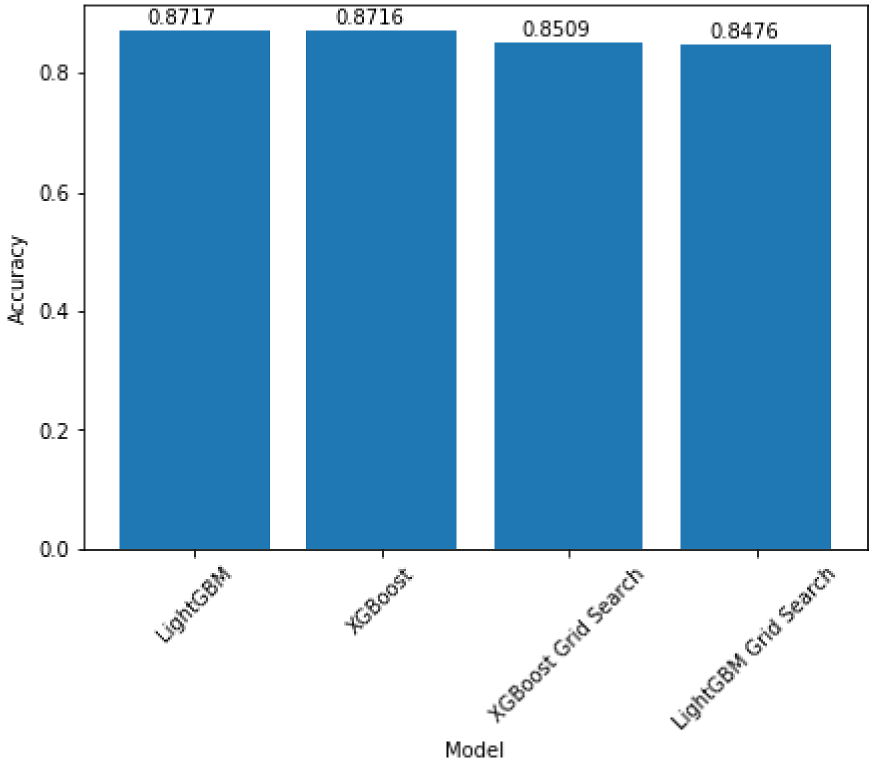
```
models = [('XGBoost', xgb_model),
          ('LightGBM', lgb_model),
          ('XGBoost Grid Search', xgb_model_gs),
          ('LightGBM Grid Search', lgbm_model_gs)]
```

In [28]:

```
def plot_metric(model_scores, score='Accuracy'):
    rcParams['figure.figsize'] = 7,5
    plt.bar(model_scores['Model'], height=model_scores[score])
    xlocs, xlabs = plt.xticks()
    xlocs=[i for i in range(0,6)]
    xlabs=[i for i in range(0,6)]
    for i, v in enumerate(model_scores[score]):
        plt.text(xlocs[i] - 0.25, v + 0.01, str(v))
    plt.xlabel('Model')
    plt.ylabel(score)
    plt.xticks(rotation=45)
    plt.show()
```

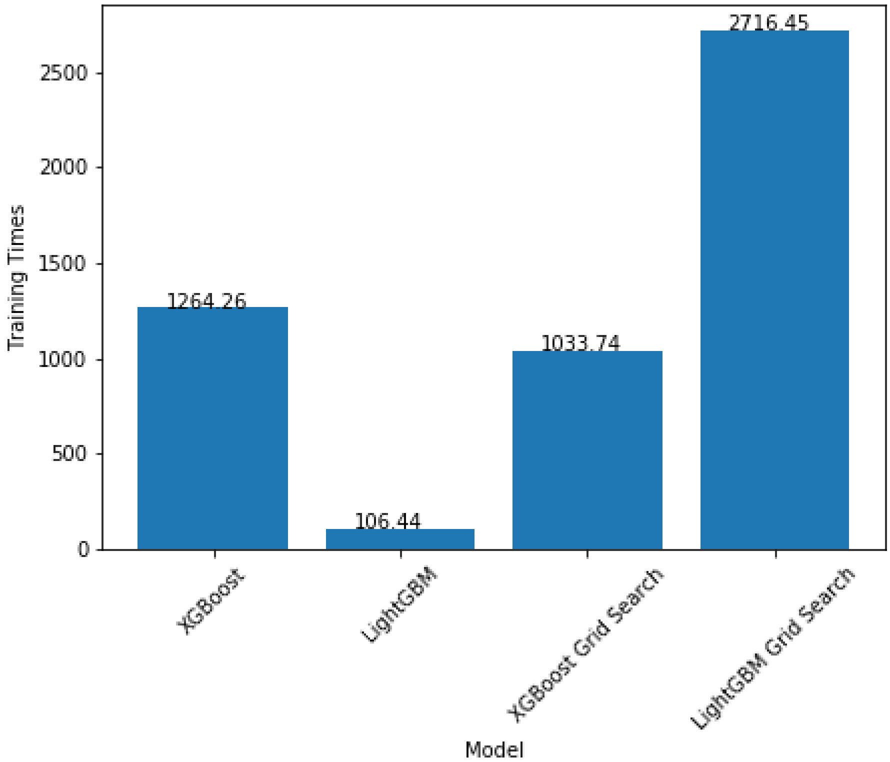
In [49]:

```
model_scores = pd.DataFrame({ 'Model': [name for name, _ in models], 'Accuracy': scores })
model_scores.sort_values(by='Accuracy',ascending=False,inplace=True)
plot_metric(model_scores)
```



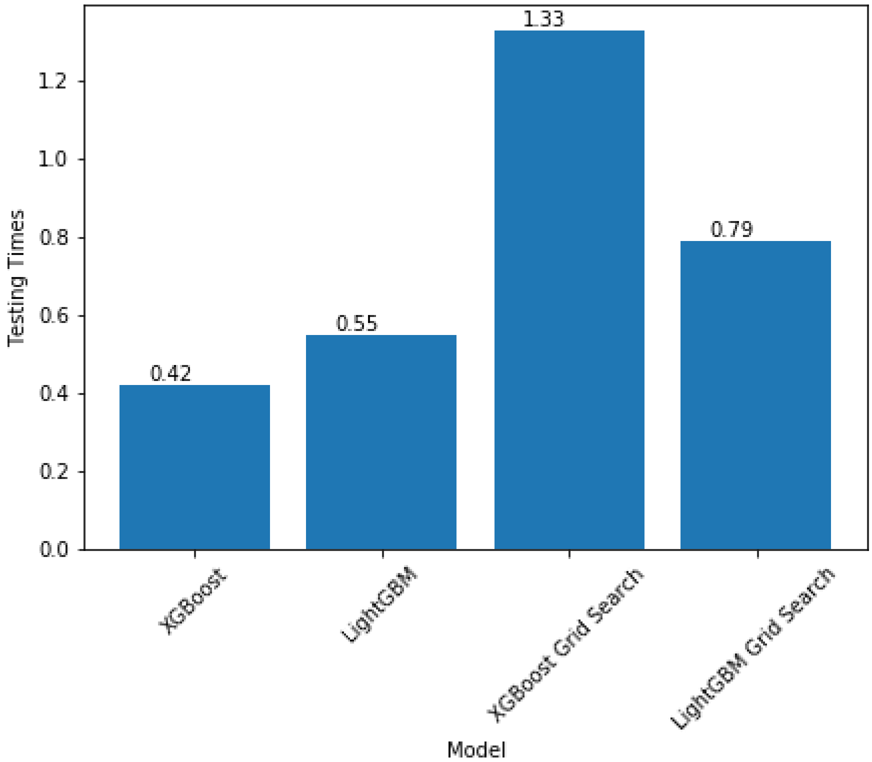
In [50]:

```
training_times = [round(time,2) for time in training_times]
model_train_times = pd.DataFrame({ 'Model': [name for name, _ in models], 'Training Times': trai
ning_times })
plot_metric(model_train_times, score='Training Times')
```



In [54]:

```
testing_times = [round(time,2) for time in testing_times]
model_train_times = pd.DataFrame({ 'Model': [name for name, _ in models], 'Testing Times': testi
ng_times })
plot_metric(model_train_times, score='Testing Times')
```

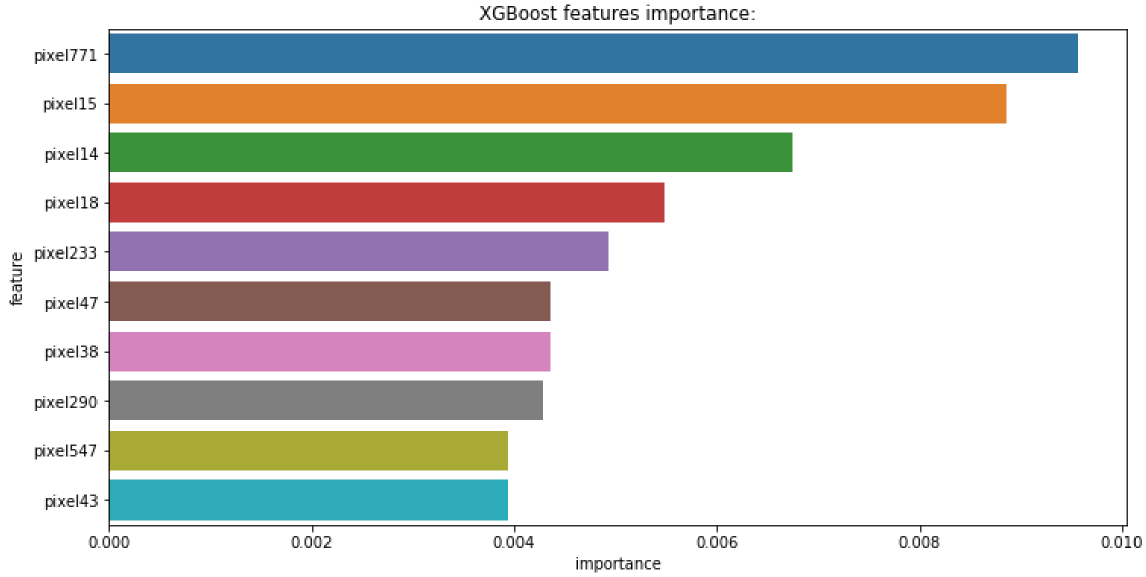


In [29]:

```
def feature_importances(df, model, model_name, max_num_features=10):
    feature_importances = pd.DataFrame(columns = ['feature', 'importance'])
    feature_importances['feature'] = df.columns
    feature_importances['importance'] = model.feature_importances_
    feature_importances.sort_values(by='importance', ascending=False, inplace=True)
    feature_importances = feature_importances[:max_num_features]
    # print(feature_importances)
    plt.figure(figsize=(12, 6));
    sns.barplot(x="importance", y="feature", data=feature_importances);
    plt.title(model_name+' features importance:');
```

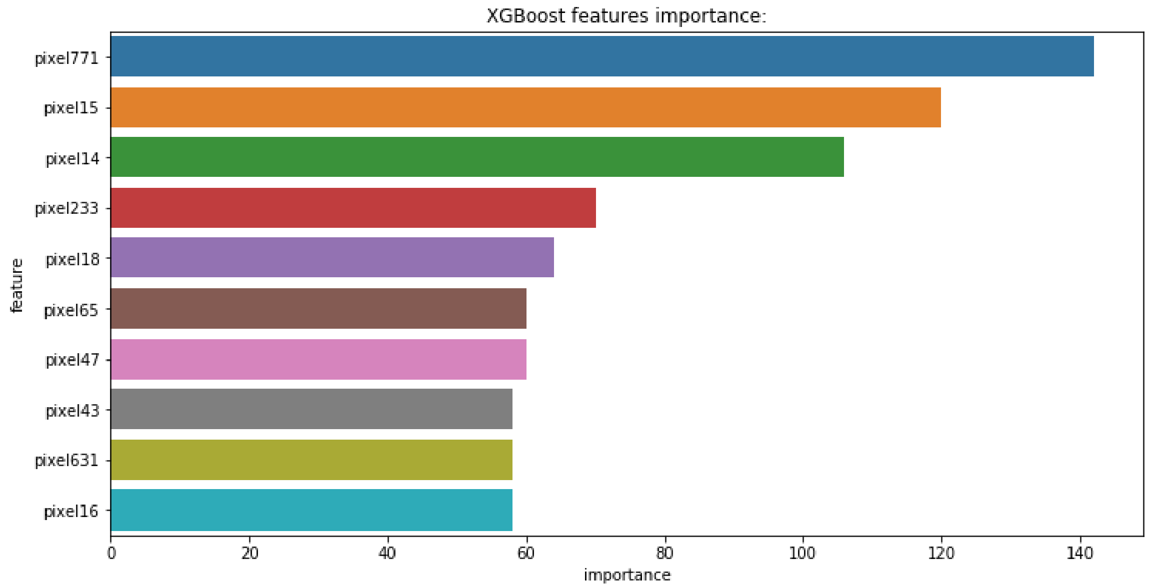
In [59]:

```
feature_importances(X_train, xgb_model, 'XGBoost')
```



In [60]:

```
feature_importances(X_train, lgb_model, 'XGBoost')
```

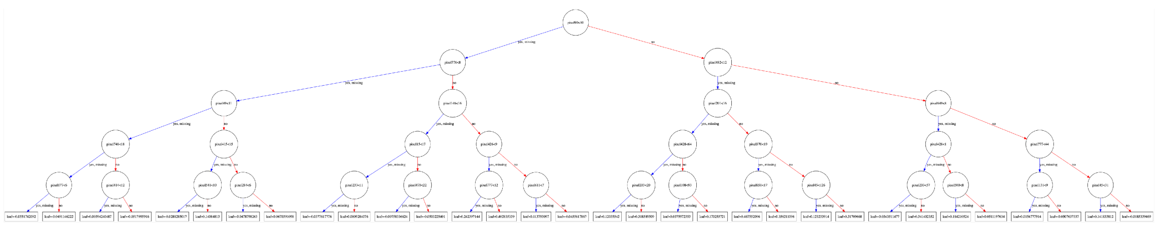


In [61]:

```
rcParams['figure.figsize'] = 80,50
```

In [64]:

```
xgboost.plot_tree(xgb_model);
```



In [66]:

```
lightgbm.plot_tree(lgb_model)
```

Out[66]:

<matplotlib.axes._subplots.AxesSubplot at 0x1a1c927198>

LightGBM的应用

In [80]:

```
# 导入相关包
import numpy as np
import pandas as pd

import lightgbm as lgb

from sklearn.metrics import mean_squared_error
```

In [70]:

```
# 加载训练数据和测试数据，以及对应的权重数据
df_train = pd.read_csv("./data/binary.train", header=None, sep='\t')
df_test = pd.read_csv("./data/binary.test", header=None, sep='\t')

W_train = pd.read_csv("./data/binary.train.weight", header=None)[0]
W_test = pd.read_csv("./data/binary.test.weight", header=None)[0]

y_train = df_train[0].values
X_train = df_train.drop(0, axis=1).values

y_test = df_test[0].values
X_test = df_test.drop(0, axis=1).values

num_train, num_feature = X_train.shape

lgb_train = lgb.Dataset(X_train, y_train, weight=W_train, free_raw_data=False)
lgb_eval = lgb.Dataset(X_test, y_test, weight=W_test, free_raw_data=False)
```

In [71]:

```
#设置模型参数
params = {
    'boosting_type': 'gbdt',
    'objective': 'binary',
    'metric': 'binary_logloss',
    'num_leaves': 31,
    'learning_rate': 0.05,
    'feature_fraction': 0.9,
    'bagging_fraction': 0.8,
    'bagging_freq': 5,
    'verbose': 0
}
```

In [73]:

```
#训练模型1-10轮迭代
lgb_model = lgb.train(params,
                      lgb_train,
                      num_boost_round=10,
                      valid_sets=lgb_train,
                      feature_name=feature_name)
```

```
[1]	training's binary_logloss: 0.680151
[2]	training's binary_logloss: 0.671664
[3]	training's binary_logloss: 0.664144
[4]	training's binary_logloss: 0.655383
[5]	training's binary_logloss: 0.647397
[6]	training's binary_logloss: 0.640486
[7]	training's binary_logloss: 0.634669
[8]	training's binary_logloss: 0.628028
[9]	training's binary_logloss: 0.621547
[10]	training's binary_logloss: 0.615672
```

In [74]:

```
#保存模型
lgb_model.save_model('./data/lgb_model.txt')
```

Out[74]:

```
<lightgbm.basic.Booster at 0x1a2815bb00>
```

In [72]:

```
#重新加载模型进行预测
lgb_model_reload = lgb.Booster(model_file='./data/lgb_model.txt')
y_pred = lgb_model_reload.predict(X_test)
print(mean_squared_error(y_test, y_pred) ** 0.5)
```

```
0.472411476758235
```

In [76]:

```python
#用已训练模型初始化模型训练11-20轮迭代
lgb_model_retrain = lgb.train(params,
                              lgb_train,
                              num_boost_round=10,
                              init_model='./data/lgb_model.txt',
                              valid_sets=lgb_eval
                              # categorical_feature=[21]
                             )
```

```
[11]	valid_0's binary_logloss: 0.617554
[12]	valid_0's binary_logloss: 0.614363
[13]	valid_0's binary_logloss: 0.609672
[14]	valid_0's binary_logloss: 0.606011
[15]	valid_0's binary_logloss: 0.602056
[16]	valid_0's binary_logloss: 0.599294
[17]	valid_0's binary_logloss: 0.595538
[18]	valid_0's binary_logloss: 0.591744
[19]	valid_0's binary_logloss: 0.58883
[20]	valid_0's binary_logloss: 0.585746
```

```
/anaconda3/envs/dev/lib/python3.6/site-packages/lightgbm/basic.py:814: UserWarnin
g: The prediction of init_model will be overridden by init_score.
  warnings.warn("The prediction of init_model will be overridden by init_score.")
```

In [79]:

```python
#调整学习率训练模型21-30轮迭代
lgb_model_retrain = lgb.train(params,
                              lgb_train,
                              num_boost_round=10,
                              init_model=lgb_model_retrain,
                              learning_rates=lambda iter: 0.05 * (0.99 ** iter),
                              valid_sets=lgb_eval)
```

```
[41]	valid_0's binary_logloss: 0.617554
[42]	valid_0's binary_logloss: 0.614394
[43]	valid_0's binary_logloss: 0.609792
[44]	valid_0's binary_logloss: 0.606231
[45]	valid_0's binary_logloss: 0.602417
[46]	valid_0's binary_logloss: 0.599771
[47]	valid_0's binary_logloss: 0.59621
[48]	valid_0's binary_logloss: 0.592633
[49]	valid_0's binary_logloss: 0.589609
[50]	valid_0's binary_logloss: 0.586783
```

In [78]:

```
#调整其他参数训练模型31-40轮迭代
lgb_model_retrain = lgb.train(params,
                              lgb_train,
                              num_boost_round=10,
                              init_model=lgb_model_retrain,
                              valid_sets=lgb_eval,
                              callbacks=[lgb.reset_parameter(bagging_fraction=[0.7] * 5 + [0.6]
* 5)])
```

```
[31]	valid_0's binary_logloss: 0.617579
[32]	valid_0's binary_logloss: 0.614267
[33]	valid_0's binary_logloss: 0.609643
[34]	valid_0's binary_logloss: 0.605865
[35]	valid_0's binary_logloss: 0.60161
[36]	valid_0's binary_logloss: 0.598602
[37]	valid_0's binary_logloss: 0.595474
[38]	valid_0's binary_logloss: 0.593449
[39]	valid_0's binary_logloss: 0.591171
[40]	valid_0's binary_logloss: 0.588738
```

In [81]:

```
#自定义损失函数
def loglikelood(preds, train_data):
    labels = train_data.get_label()
    preds = 1. / (1. + np.exp(-preds))
    grad = preds - labels
    hess = preds * (1. - preds)
    return grad, hess

#自定义评估函数
def binary_error(preds, train_data):
    labels = train_data.get_label()
    return 'error', np.mean(labels != (preds > 0.5)), False

#使用自定义损失及评估函数训练模型41-50轮迭代
lgb_model_retrain = lgb.train(params,
                              train_set=lgb_train,
                              num_boost_round=10,
                              init_model=lgb_model_retrain,
                              fobj=loglikelood,
                              feval=binary_error,
                              valid_sets=lgb_eval)
```

```
[51]	valid_0's binary_logloss: 5.16783	valid_0's error: 0.402
[52]	valid_0's binary_logloss: 5.46634	valid_0's error: 0.392
[53]	valid_0's binary_logloss: 5.07286	valid_0's error: 0.39
[54]	valid_0's binary_logloss: 5.30891	valid_0's error: 0.382
[55]	valid_0's binary_logloss: 5.54901	valid_0's error: 0.37
[56]	valid_0's binary_logloss: 5.65039	valid_0's error: 0.368
[57]	valid_0's binary_logloss: 5.56936	valid_0's error: 0.356
[58]	valid_0's binary_logloss: 5.73844	valid_0's error: 0.354
[59]	valid_0's binary_logloss: 5.66427	valid_0's error: 0.352
[60]	valid_0's binary_logloss: 5.61407	valid_0's error: 0.35
```