需要给输入数据指定相应的格式。

在TVM中，特征通过C++相关代码获取并打包，在Python进行解包并用于xgboost模型的更新。

```
features, normalized_throughputs, task_ids =
    get_per_store_features_from_measure_pairs(
    self.inputs, self.results, skip_first_n_feature_extraction=n_cac
```

即输入数据包含“特征”，“标准化吞吐量”和“任务id”。
然后将这几种数据打包为xgb.DMatrix格式，并按照task id按顺序：

```
dtrain = pack_sum_xgbmatrix(
    features, normalized_throughputs, task_ids, normalized_throughput
)
```

在pack_sum_xgbmatrix（）中，通过一系列子任务排序等操作，最终将(feature, label)定义xgb.DMatrix格式的数据。

```
ret = xgb.DMatrix(np.array(x_flatten), y_flatten)
```

至此完成xgboost模型训练数据的格式转换。

参考TVM的做法，我们没有子任务的划分，即对于整个程序，我们提取出与运行时间有关的特征以及程序相应的运行时间即可。

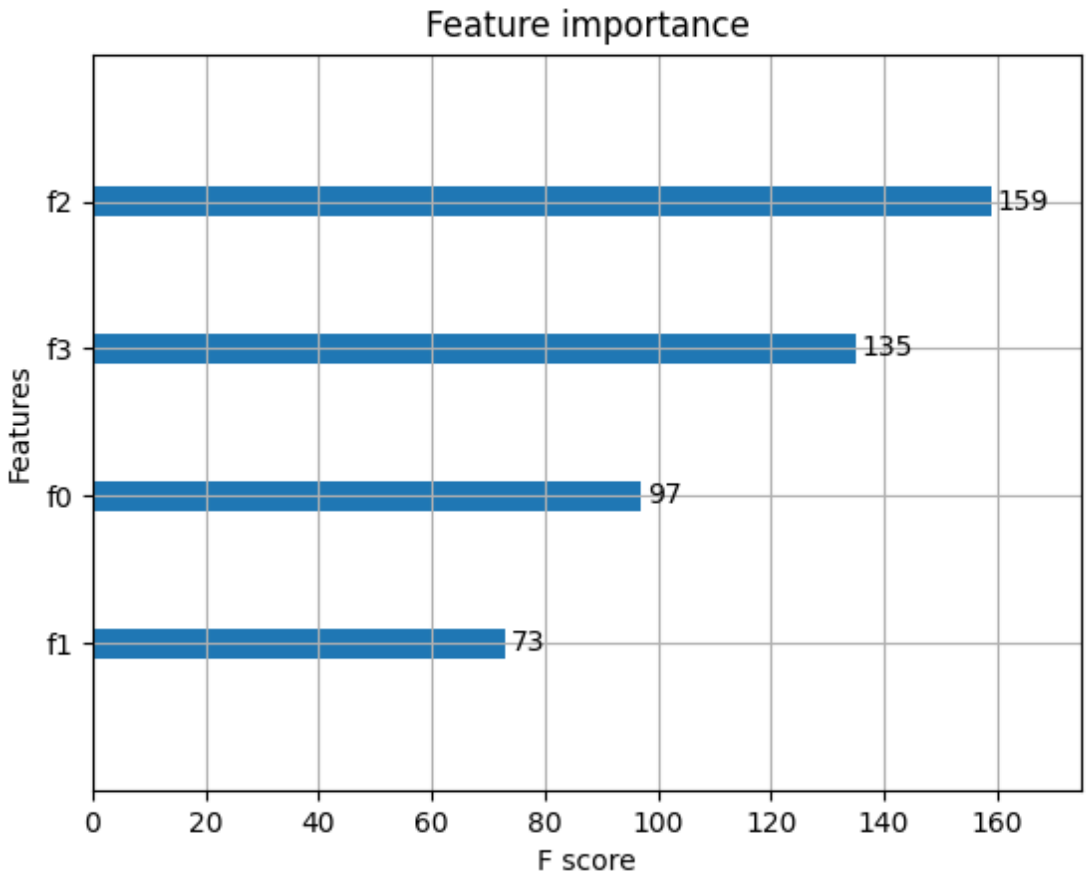
如何创造训练数据

由于没有TVM那样的代码生成和进化搜索过程，我们肯定会面临样本不足的问题。
但是如何生成最佳代码不是cost model的考虑，它所要做的就是对代码的评估，理论上只要我们微调代码，跑出runtime，得到足够多的程序特征-运行时间对，我们就可以拟合出一个可以根据特征预测运行时间的回归树模型。

如果获取到的不是groundtruth，而是根据规则计算出来的“runtime”，那么模型最终拟合出来的也只能是规则本身。

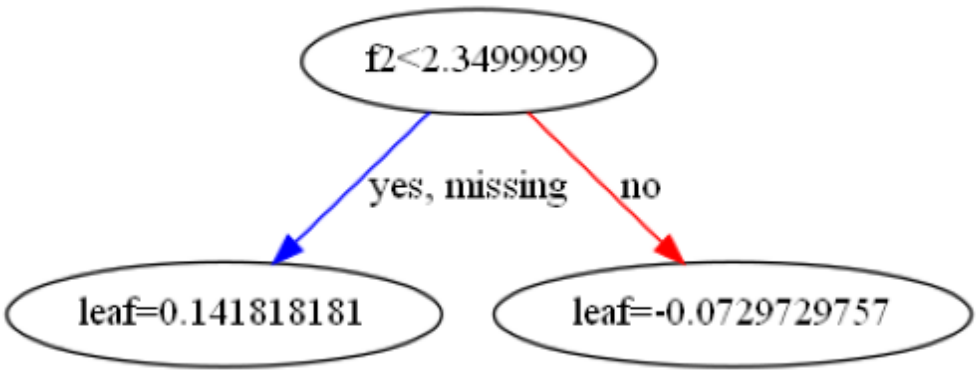
如何调用XGBoost模型

目前还没有特征提取的接口，那么特征数据就是裸露的文件，用C++或Python都是可以读取的。
下面是XGBoost完成训练后，对不同的特征的重要性排序（值越大说明特征越重要，所在树的节点也越上层）：

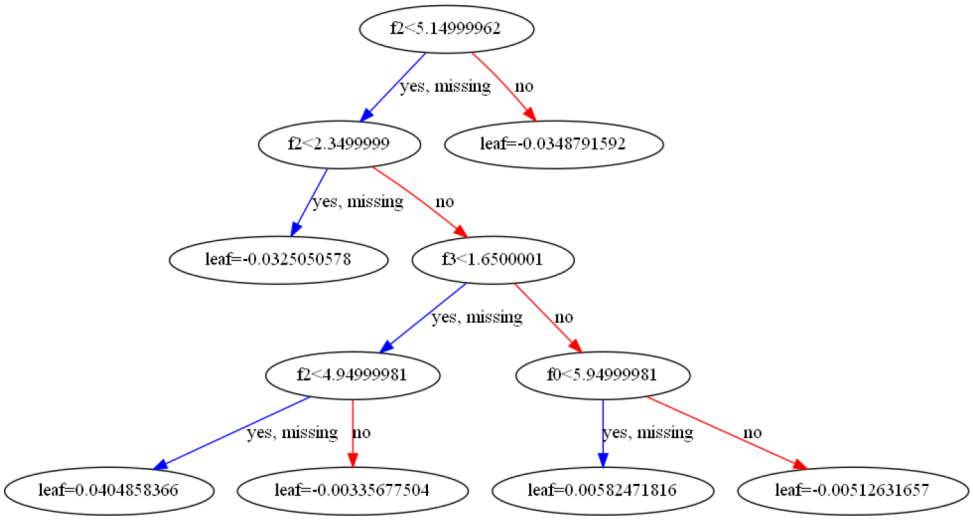


训练结果中的决策树：

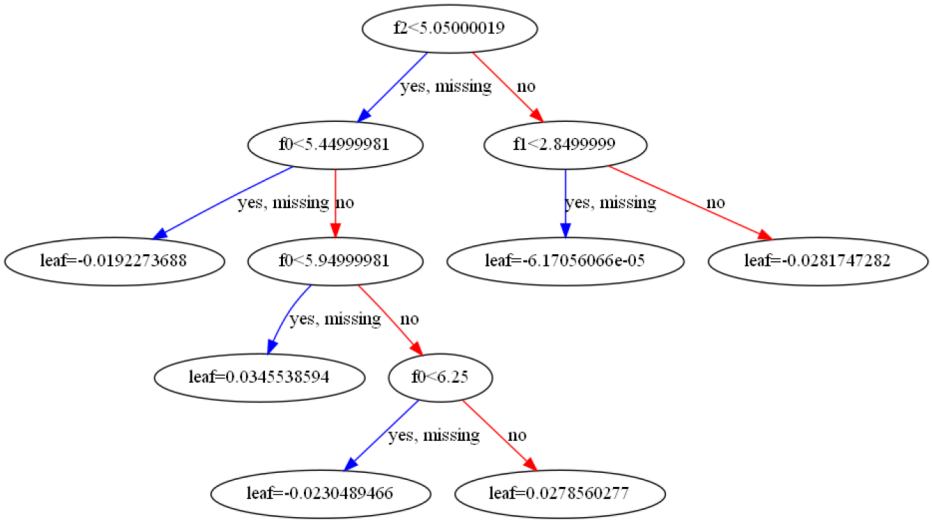
num_trees=0第一棵树



num_trees=100



num_trees=130，有stop_earlying的参数



下图是xgb实战相关

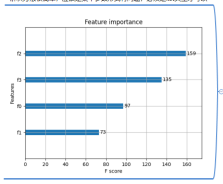
Xgboost实战

安装xgboost库 安装Scikit-learn库

```
pip3 install xgboost -i https://pypi.tuna.tsinghua.edu.cn/simple
pip3 install -U scikit-learn -i https://pypi.tuna.tsinghua.edu.cn/simple
```

记录报错

'dict_items' object has no attribute 'copy'
解决方法很简单，应该是某个参数形式有问题，必须是list类型才可以



数据处理

数据格式

1. Libsvm格式的文本数据 `Dtrain1 = Xgb.Dmatrix('Train.Svm.Txt')`
2. Numpy的二位数组

```
Data = Np.Random.Randi(5,10) # 5 个对象, 每个包含 10 个特征
Label = Np.Random.Randint(2, Size=5) # 二进制目标
Dtrain = Xgb.Dmatrix( Data, Label=Label)
```

3. Xgboost的二进制的缓存文件 `Dtrain2 = Xgb.Dmatrix('Train.Svm.Buffer')`

将Scipy.Sparse格式的数据转化为 Dmatrix 格式

```
Csr = Scipy.Sparse.Csr_Matrix( (Dat, (Row,Col)) )
Dtrain = Xgb.Dmatrix( Csr )
```

将 Dmatrix 格式的数据保存成Xgboost的二进制格式，在下次加载时可以提高加载速度

```
Dtrain = Xgb.Dmatrix('Train.Svm.Txt')
Dtrain.Save_Binary("Train.Buffer")
```

处理 Dmatrix中的缺失值 `Dtrain = Xgb.Dmatrix( Data, Label=Label, Missing = -999.0)`

给样本设置权重

```
W = Np.Random.Randi(5,1)
Dtrain = Xgb.Dmatrix( Data, Label=Label, Missing = -999.0, Weight=W)
```

参数设置

```
Params = {
    'Booster': 'Gbtree',    #Gbtree使用基于树的模型进行提升计算，Gblinear使用线性模型进行提升计算
    'Objective': 'MultiSoftmax',  # 定义学习任务及相应的学习目标
    'Num_Class': 10,              # 类别数，与 Multisoftmax 并用
    'Gamma': 0.1,                 # 用于控制是否后剪枝的参数,越大越保守，一般0.1、0.2这样子。
    'Max_Depth': 12,              # 构建树的深度，越大越容易过拟合
    'Lambda': 2,                # 控制模型复杂度的权重值的L2正则化项参数，参数越大，模型越不容易过拟合。
    'Subsample': 0.7,             # 随机采样训练样本的比例
    'Colsample_Bytree': 0.7,      # 在建立树时对特征采样的比例。缺省值为1
    'Min_Child_Weight': 3,      #子节点中最小的样本权重和。如果一个叶子节点的样本权重和小于Min_Child_Weight则拆分过程结束，越大越保守
    'Silent': 1,                  # 取0时表示打印出运行时信息，取1时表示以缄默方式运行，不打印运行时信息
    'Eta': 0.007,                 # 如同学习率。在每次提升计算之后，算法会直接获得新特征的权重。 Eta通过缩减特征的权重
    'Seed': 1000,                 # 随机数的种子
    'Nthread': 4,                 # Cpu 线程数
}
```

Objective [Default=RegLinear]

- "RegLinear" --线性回归。
- "RegLogistic" --逻辑回归。
- "BinaryLogistic" --二分类的逻辑回归问题，输出为概率。
- "BinaryLogitraw" --二分类的逻辑回归问题，输出的结果为Wtx。
- "CountPoisson" --计数问题的Poisson回归，输出结果为Poisson分布。在Poisson回归中，Max_Delta_Step的缺省值为0.7。(Used To Safeguard Optimization)
- "MultiSoftmax" --让Xgboost采用Softmax目标函数处理多分类问题，同时需要设置参数Num_Class（类别个数）
- "MultiSoftprob" --和Softmax一样，但是输出的是Ndata * Nclass的向量，可以将该向量Reshape成Ndata行Nclass列的矩阵。没行数据表示样本所属于每个类别的概率。
- "RankPairwise" --让Xgboostset 通过最小化成对损失来对任务进行排序

训练模型

```
num_round = 10
bst = xgb.train( plst, dtrain, num_round, evallist )
```

params (dict) – 参数
dtrain (DMatrix) –训练数据
num_boost_round (int) – 迭代次数
evals (list of pairs (DMatrix, string)) – 训练时用于评估的验证集
obj (function) – 自定义目标函数
feval (function) – 自定义评估函数
maximize (bool) – 是否最大化评估函数
early_stopping_rounds (int) – 提前停止
evals_result (dict) – 储存所有评估的评估结果的字典
verbose_eval (bool or int) – 打印评估信息等级
xgb_model (file name of stored xgb model or 'Booster' instance) – 加载的XGBoost模型，支持继续训练
callbacks (list of callback functions) – 每次迭代结束后的回调函数

模型预测

```
# X_test类型可以是二维List，也可以是numpy的数组
dtest = DMatrix(X_test)
ans = model.predict(dtest)
```

保存模型

```
bst.save_model('test.model')
# dump model导出
bst.dump_model('dump.raw.txt')
# 导出带有特征映射map
bst.dump_model('dump.raw.txt','featmap.txt')
```

加载模型

```
bst = xgb.Booster({'nthread':4}) # init model
bst.load_model("model.bin")     # load data
```