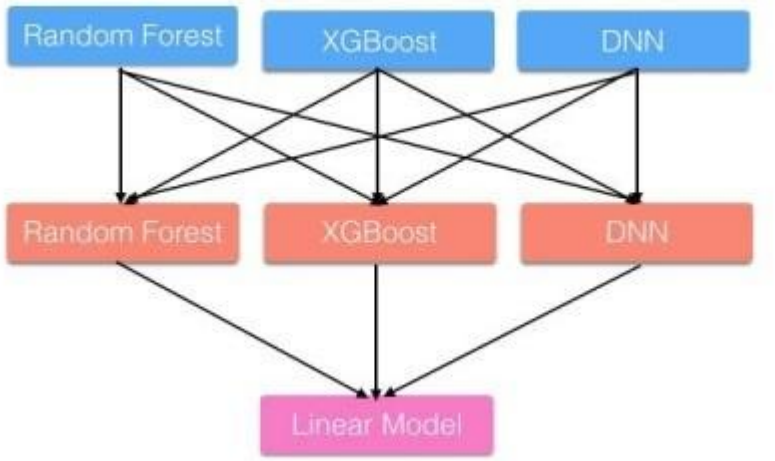
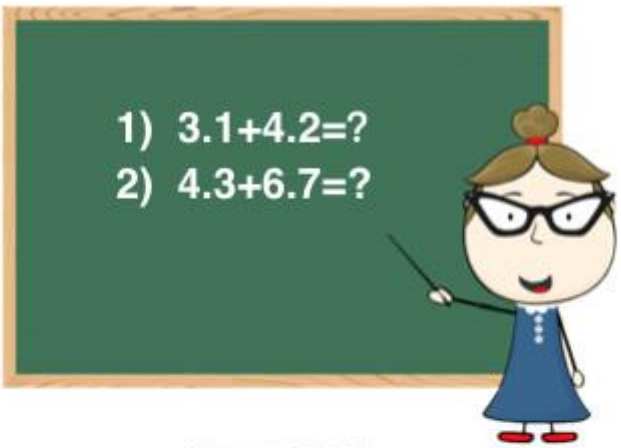
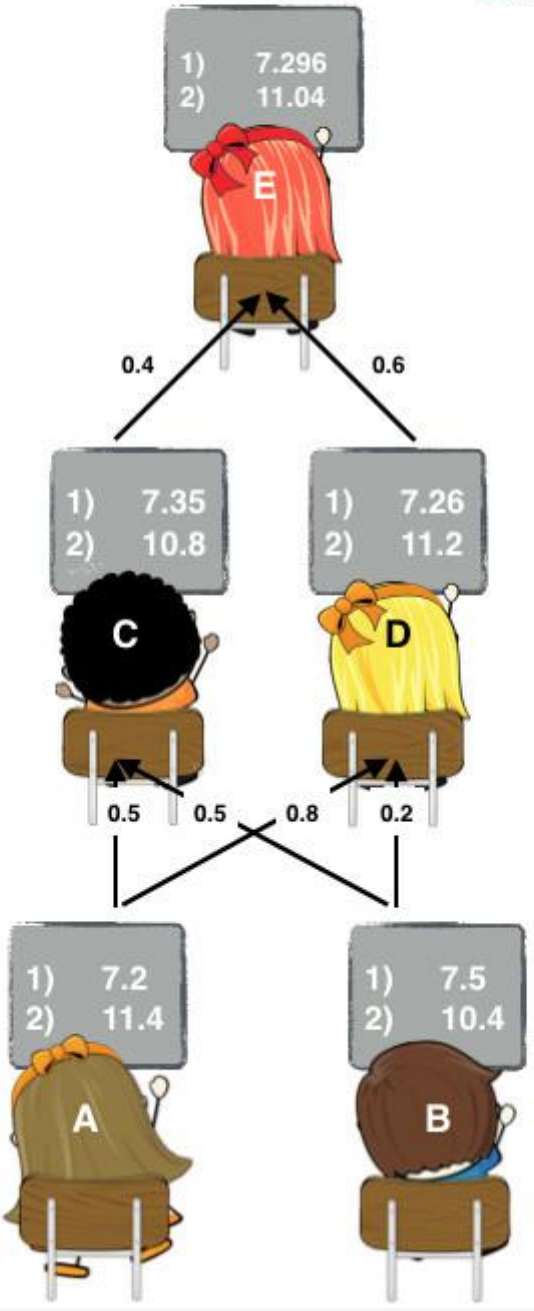
关于 Stacking 模型融合

Stacking 是用**新的模型（次学习器）去学习怎么组合那些基学习器**，它的思想源自于 Stacked Generalization（http://www.machine-learning.martinsewell.com/ensembles/stacking/Wolpert1992.pdf）这篇论文。如果把 Bagging 看作是多个基分类器的线性组合，那么 Stacking 就是多个基分类器的非线性组合。Stacking 可以很灵活，它可以将学习器一层一层地堆砌起来，形成一个网状的结构，如下图：

举个更直观的例子，还是那两道加法题：



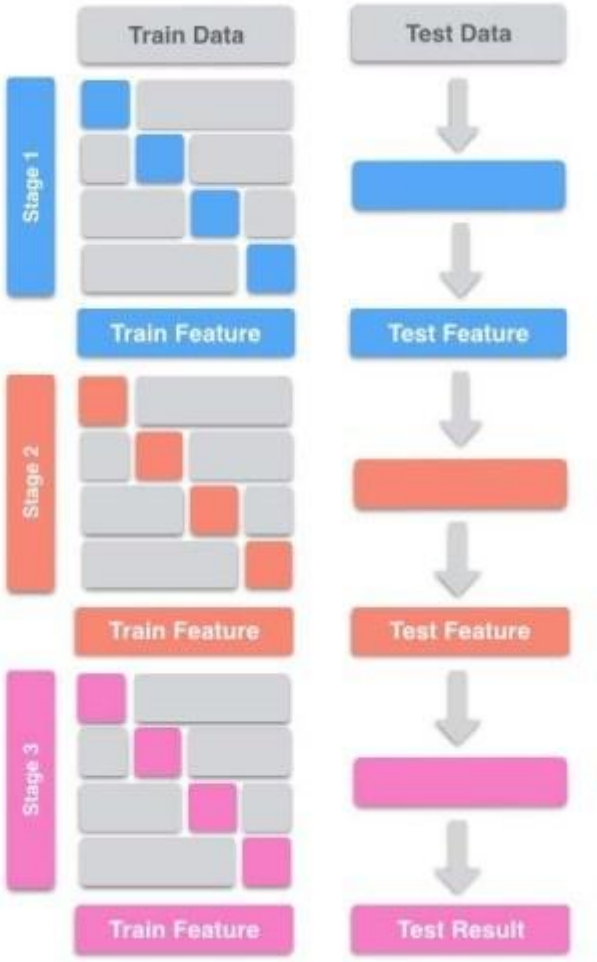

1) 3.1+4.2=?
2) 4.3+6.7=?


1) 7.296
2) 11.04
E
0.4
0.6
1) 7.35
2) 10.8
C
1) 7.26
2) 11.2
D
0.5
0.5
0.8
0.2
1) 7.2
2) 11.4
A
1) 7.5
2) 10.4
B

这里 A 和 B 可以看作是基学习器，C、D、E 都是次学习器。

- Stage1: A 和 B 各自写出了答案。
- Stage2: C 和 D 偷看了 A 和 B 的答案，C 认为 A 和 B 一样聪明，D 认为 A 比 B 聪明一点。他们各自结合了 A 和 B 的答案后，给出了自己的答案。
- Stage3: E 偷看了 C 和 D 的答案，E 认为 D 比 C 聪明，随后 E 也给出自己的答案作为最终答案。

在实现 Stacking 时，要注意的一点是，避免标签泄漏(Label Leak)。在训练次学习器时，需要上一层学习器对 Train Data 的测试结果作为特征。如果我们在 Train Data 上训练，然后在 Train Data 上预测，就会造成 Label Leak。为了避免 Label Leak，需要对每个学习器使用 K-fold，将 K 个模型对 Valid Set 的预测结果拼起来，作为下一层学习器的输入。如下图：

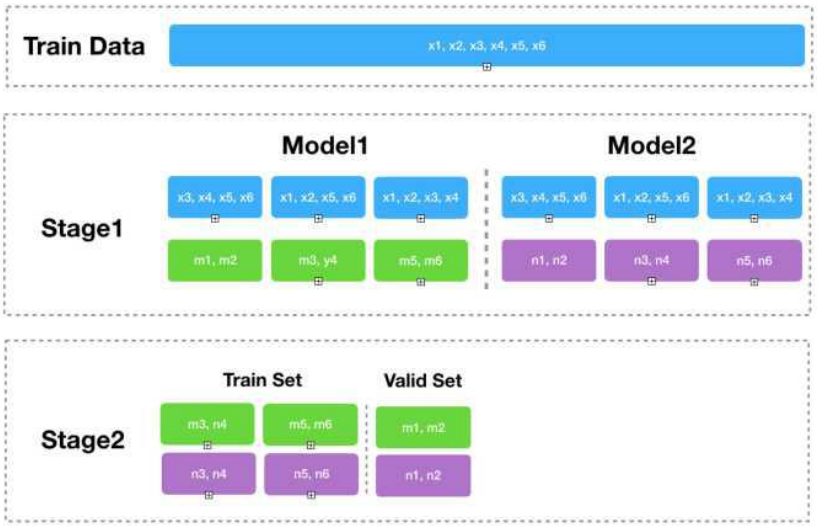


由图可知，我们还需要对 Test Data 做预测。这里有两种选择，**可以将 K 个模型对 Test Data 的预测结果求平均，也可以用所有的 Train Data 重新训练一个新模型来预测 Test Data**。所以在实现过程中，我们最好把每个学习器对 Train Data 和对 Test Data 的测试结果都保存下来，方便训练和预测。

对于 Stacking 还要注意一点，固定 K-fold 可以尽量避免 Valid Set 过拟合，也就是全局共用一份 K-fold，如果是团队合作，组员之间也是共用一份 K-fold。如果想具体了解为什么需要固定 K-fold，

举个例子，假设训练数据一共有 x1, x2, x3, x4, x5, x6 这 6 个，并且使用 3-fold，在 Stage1 的时候使用两个 Model。

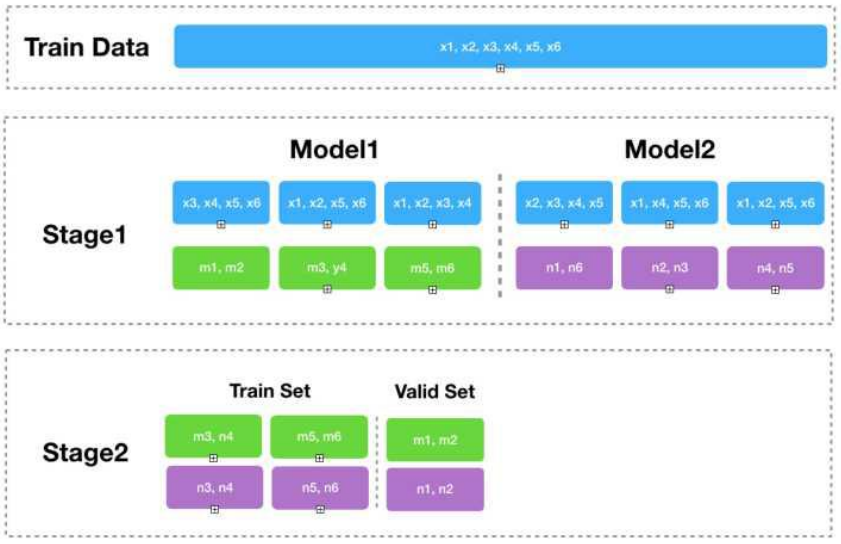
1. 使用固定的 k-fold 的情况如下图：



先看 Stage1：可以看到 m1-m6 和 n1-n2 是相同的 k-fold 预测出来的结果，m1,m2 由 x3,x4,x5,x6 预测所得，所以 m1,m2 包含了 x3,x4,x5,x6 的信息，以此类推。

Stage2：假设我要用如图中所示的 Train Set 来预测 Valid Set，那么 Train Set 包含了 x1-x6 的信息，而 Valid Set 包含了 x3-x6 的信息。

看到这里你会疑惑，这有什么用呢？别急，再来看看 Stage1 的两个 Model 各自用不同的 k-fold：



请仔细观察 Model2 的 k-fold，现在 n1,n6 包含了 x2-x5 的信息，以此类推。

关键在于 Stage2，现在 Train Set 也是包含了 x1-x6 的信息，而 Valid Set 也包含了 x1-x6 的信息，这就是不固定 kfold 与固定 kfold 的区别。尽管从固定 kfold 的图看来，它也有可能出现一定程度的过拟合，但不固定 kfold 它对 Valid Set 的过拟合

情况会更加严重，所以按照 Nutastray 说的，通过 kfold 可以避免人为造成的过拟合。

可能你会问，那如果我们同一层的 Stage 固定 k-fold，而不同层之间不固定，会发生什么？答案是，情况也会比固定 k-fold 要糟糕，具体的话可以按照上图画一下。所以最终给出的建议是，做 Stacking 最好还是固定 k-fold，如果是团队合作完成项目，那就组员之间共享一份 k-fold。

下面给出我的代码：

```
bb=data_train.iloc[:, 4:6749]
#dd=data_train.iloc[:, 3:4]
cc = bb.apply(lambda x: x.fillna(x.mean()), axis=0)
x_data = preprocessing.minmax_scale(cc.iloc[:, :].values,
feature_range=(-1,1))
cc['tag'] = data_train.iloc[:, 3:4]
test = dfp.iloc[:, 2:6747].apply(lambda x: x.fillna(x.mean()),
axis=0)
#test_data = preprocessing.minmax_scale(test.iloc[:, :].values,
feature_range=(-1,1))
Xtest = list(test.columns.values)[4:6745]
x_data_output = dfp.iloc[:, 0:1].values
#print(data_train.iloc[:, 3:4])


predictors = list(cc.columns.values)[4:6745]
alg1 = lgb.LGBMClassifier(boosting_type='gbdt', num_leaves=42,
max_depth=-1, learning_rate=0.054, n_estimators=490,
                          subsample_for_bin=200,
objective='binary', class_weight=None, min_split_gain=0.0,
                          min_child_weight=1,
min_child_samples=21, subsample=0.72, subsample_freq=1,
                          colsample_bytree=0.63,
reg_alpha=6.18, reg_lambda=2.718, random_state=142857, n_jobs=-1,
                          silent=True,
importance_type='split')
alg2 =
XGBClassifier(n_estimators=60,max_depth=9,min_child_weight=2,g
amma=0.9,subsample=0.8,learning_rate=0.02,

colsample_bytree=0.8,objective='binary:logistic',nthread=-1,sc
ale_pos_weight=1)
alg3 = GradientBoostingClassifier(learning_rate=0.01,
n_estimators=600, max_depth=7, min_samples_leaf=60,
                        min_samples_split=1200, max_features=9,
subsample=0.7, random_state=10)
```

```
lr = LogisticRegression()

pipe1 = make_pipeline(ColumnSelector(cols=predictors[4:2000]),
lr)
pipe2 = make_pipeline(ColumnSelector(cols=predictors[2000:4000]),
alg2)
pipe3 = make_pipeline(ColumnSelector(cols=predictors[4000:5000]),
alg3)

sclf = StackingClassifier(classifiers=[lr, pipe2, pipe3],
meta_classifier=alg1)

# Compute the accuracy score for all the cross validation folds.
(much simpler than what we did before!)
#
kf=cross_validation.KFold(data_train.shape[0],n_folds=10,rando
m_state=1)
kf = model_selection.KFold(n_splits=120, shuffle=False,
random_state=1)
scores = model_selection.cross_val_score(sclf, cc[predictors],
cc['tag'], cv=kf)
print("scores.mean=", scores.mean())

File = open("data/prob_stackingLXG.txt", "w", encoding=u'utf-8',
errors='ignore')
File.write("id"+"," + "prob" + "\n")
classifier = sclf.fit(cc[predictors], cc['tag'])
predictiontest = classifier.predict_proba(test[Xtest])[:, 1]
for step in range(len(test)):
    File.write(str(x_data_output[step][0])+"," +
str(predictiontest[step]) + "\n")

end = time.time()
stamp = end - start
print("耗时", stamp / 3600)
#print(predictiontest)
```

大家有感兴趣的可以自己拿代码测试一下。我的 github 完整测试代码地址如下：
https://github.com/crystal-tensor/-360

另外重要的一点补充，以上的实现方法是串行话的方式，我后来实现了并行化的方式，具体代码见：mul_algorithm.py