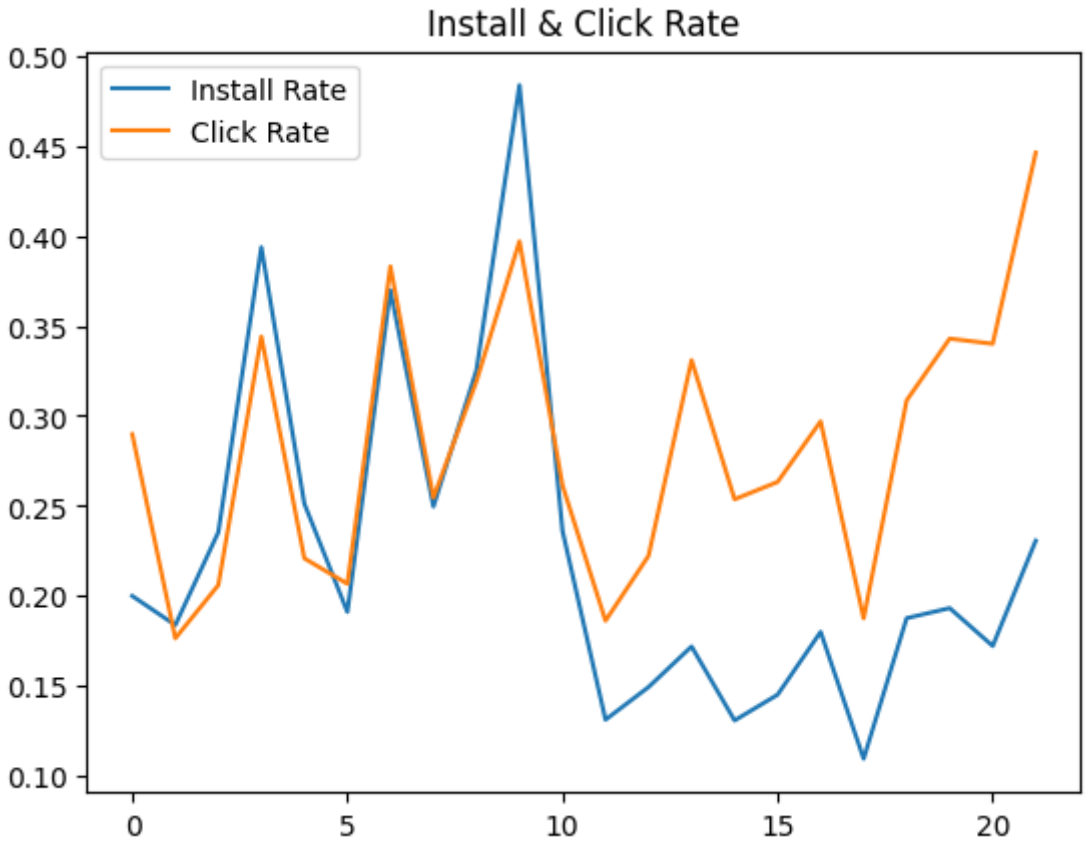
进行的尝试

数据初步分析

- 读取训练集和测试集数据并进行特征和标签的分离；
- 因为是一个时间周期下预测用户未来一天的安装概率，因此分析不同天数的样本总的点击率和安装率，如图所示



- 统计特征每列和y_install之间的相关性，方便后续对特征进行筛选

DeepFM

代码的完成

文件说明如下：

- args.py：设置公共的参数；dataset.py：处理数据集，设置获取单个样本数据的格式；
- DeepFM.py：网络结构的设置、模型训练、保存结果，核心文件；evaluation.py：官方给出的评估代码
- result_mean.py：将两个模型得到的结果进行平均取值
- main.py&main.ipynb：前者运行DeepFM模型的主函数，后者包括开始的数据分析和XGBoost模型的参数设置

主要根据GitHub前人的代码，同时结合对问题自身特点的理解，加入了自己的修改具体原理可参照前文的讲解，下面列举自己所加入的一些代码设置：

- 将预先求得的类别特征信息、数值特征标志、预测点击或安装等必要参数放入args.py中，不用每次都从命令行输入，运行时进行修改即可

- 记录每次训练的loss和accuracy（因为该任务样本量较大，不是将全部样本训练完后得到其loss，而是设置一个超参print_every，每经过这么多次batch之后记录训练loss和accuracy），根据预测标签的不同写入txt文件中
- 需要对离散特征进行重新编码和归一化，此时训练集和测试集应该放一起同步处理
- 将参数的设置全部放在main.py中方便调整，同时lab_id记录每次实验的结果和图像

不同的调整

保持基本的DeepFM模型不变，主要思路是调整网络结构的参数、初始数据的处理等

1.一开始进行模型的调参，关注的模型参数分别如下所述：

- embedding_size：将类别特征进行映射的大小
- hidden_dims：网络结构，基本的前馈神经网络每层神经元数
- dropout：经过每层网络后dropout的概率，默认全部相同
- 还有比如验证集的大小、早停策略paience的设置
- batch_size：每个batch的样本大小

```
# batch_size和activation可以有不同选择
# training
# 当筛选特征之后需要相应地修改nume_fea_size
model = DeepFM(feature_sizes, embedding_size=4,nume_fea_size=16,
               hidden_dims=[64,64,64],num_classes=1,dropout=[0.3,0.3,0.3],
               use_cuda=True,cuda_name='cuda:0')
# weight_decay 权重衰减系数 L2正则化项，防止过拟合，也可能导致模型训练效果下降
optimizer = optim.Adam(model.parameters(), lr=1e-2, weight_decay=0.0)
schedule = ReduceLROnPlateau(optimizer,'min',factor=0.2,patience=4,min_lr=1e-6,verbose=True)
print("Start Training...")
# wait用于早停策略
lab_id = 25
model.fit(loader_train, loader_val, optimizer,schedule, epochs=100,
          verbose=True,print_every=500,wait=12,lrd=True,figure_num=lab_id)
model.Get_result(loader_test,model)
```

经过调参最明显的体验就是变化规律并不明显，总结原因是因为超参数对结果的影响并没有那么显著，无法排除偶然因素的干扰，也因为测试集的情况是未来某一天的数据，其分布完全无法预知，所以并没有找到一定最优的超参数，但是基本确定了较为合适的超参。举例来说，比如embedding_size=2/4较为合适，因为类别特征唯一值个数也是2/4居多，因此符合实验结果和主观判断。

2.当调参陷入僵局之后，需要从别的方面来寻求进一步指标的提升，这时候我们开始思考将特征不加筛选一股脑的丢进去是否合理，结论是显然对特征进行初步的筛选是有必要的。

我们将数值特征和类别特征不作区分，统一计算它们和标签的相关性，同时设置不同的相关系数阈值，比如只选择相关性绝对值>0.02的特征放入网络训练，在经过几次不同阈值的尝试之后，我们发现选择特征不同对结果会产生明显的影响，因此在选择表现最好的阈值之后指标也提升了不少；

同时因为该任务并未对每个特征的语义阐明，我们无法根据个人的主观经验或是背景知识手动选取特征或是进行特征组合，只能分析数据的分布特点做一定的推测来选取特征尝试。

```
# 将f_cor中的数据按绝对值大小排序
f_cor_abs = abs(f_cor)
# 保留绝对值大于0.02的特征
f_cor_abs = f_cor_abs[f_cor_abs > 0.02]
# 得到保留的特征名，按照原来f_cor的顺序排列
f_cor_abs_name = f_cor_abs.index
# # 去掉is_installed和f_1
f_cor_abs_name = f_cor_abs_name.drop("is_installed")
f_cor_abs_name = f_cor_abs_name.drop("f_1")
print(len(f_cor_abs_name))
f_cor_abs_name
```

3.在官方公布测试集的评估函数之后，猜想不同的损失函数的设置是否也会对结果产生影响，比如因为数据集的正负样本比例是有差异的，如果尝试对正样本的误差给予更多关注，即如果用户标签是会下载而模型预测为不下载，那么分类错误，将该误差放大以提高对样本的关注，通过设置正负样本的权重实现。从结果来看确实发生了较大变化，但是是往坏的趋势变化，因此该尝试失败。

```
self.model_path = "./models/" + "lab" + str(figure_num) + "best_weights.pth"
# 尝试对正样本的误差给予更多关注
# criterion = nn.BCEWithLogitsLoss(pos_weight=torch.tensor([1.2]).to(self.device))
criterion = nn.BCELoss()
if lrd:
```

4.为了加快模型的训练，并不是在每个epoch全部样本训练之后才进行train_loss和val_loss的计算，而是在训练过程中经过给定次数个batch即计算一次，然后判断要不要早停，这样做的好处是训练epoch数会减少，坏处是可能与样本的分布有关导致过早早停，在后期更改了该设置加大训练时长后，指标有所提升。

XGBoost

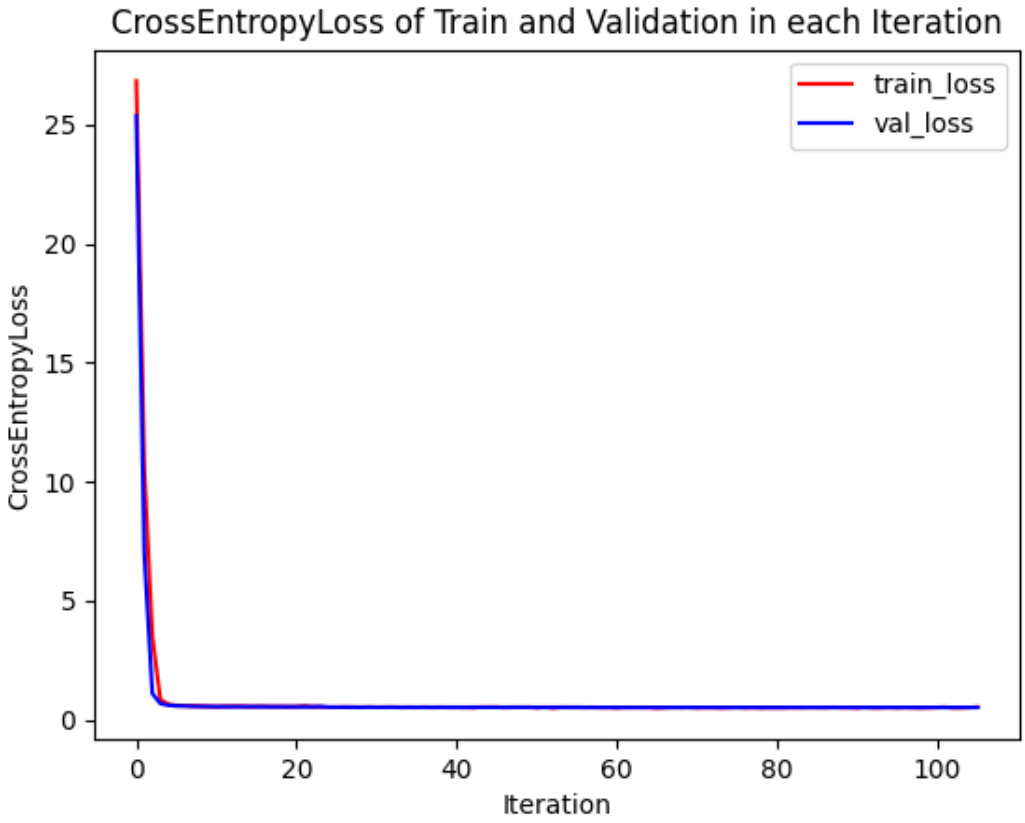
在DeepFM模型经过调参和不同尝试取得了还算不错的效果且提升不明显之后，想要尝试加入一个传统的机器学习模型进行两个模型的集成来获得结果，但是该模型并未花很多时间去调整。基本思路是直接调用xgboost库的XGBClassifier来处理该任务，初步试探了不同的特征选取和参数设置之后得到了实验结果。

```
X_train, X_val, y_train, y_val = train_test_split(X_train_select, y_install, test_size=0.15, stratify=y_
install_model = XGBClassifier(n_estimators=200,
                              max_depth=5,
                              reg_lambda=0.3,
                              reg_alpha=2,
                              learning_rate=0.15,
                              eval_metric="logloss",
                              early_stopping_rounds=10)
evalset = [(X_train, y_train), (X_val, y_val)]
install_model.fit(X=X_train, y=y_train, eval_set=evalset, verbose=0)
```

模型集成

简单的想法是将两个模型得到的预测概率做一个平均作为最终结果，同时也尝试了不同的比重提交查看排行榜分数，发现简单平均的效果最好，因此模型的集成只是将结果求平均，排名又上升了一些之后即结束了本次比赛。

训练过程中保存的截图（示例）


CrossEntropyLoss of Train and Validation in each Iteration
train_loss
val_loss
CrossEntropyLoss
Iteration
0
5
10
15
20
25
0
20
40
60
80
100