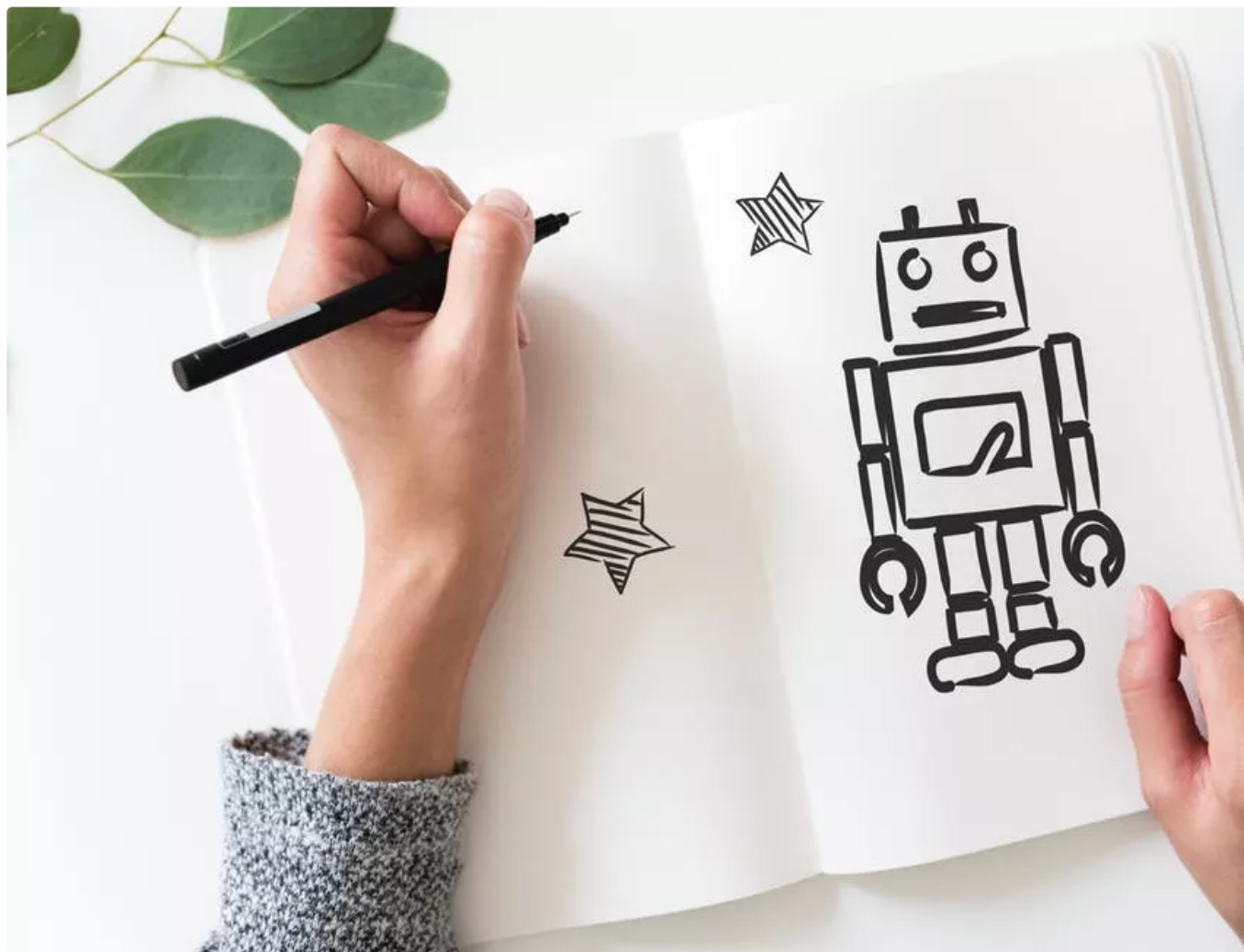


CTR预估专栏 | 详解Wide&Deep理论与实践

李宁宁 AI前线 2018-07-13



作者 | 李宁宁

编辑 | Debra

- 计算广告CTR预估系列(四)--Wide&Deep理论与实践
 - 1. 名词解释
 - 1.1 Memorization 和 Generalization
 - 1.2 Wide 和 Deep
 - 1.3 Cross-product transformation
 - 2. Wide & Deep Model
 - 2.1 推荐系统
 - 2.1.1 介绍
 - 2.1.2 线性模型
 - 2.1.3 Embedding-Based
 - 2.1.4 工作流程
 - 2.2 Wide Part
 - 2.3 Deep Part
 - 2.4 模型训练
 - 3. 系统实现
 - 3.1 训练数据生成
 - 3.2 模型训练
 - 3.3 线上使用
 - 4. 适用范围
 - 5. 优缺点
 - 6.代码实践
 - 6.1 Wide Linear Model
 - 6.2 Wide & Deep Model
 - Reference

今天的主角是 Wide & Deep Model，在推荐系统和 CTR 预估中都有应用。万字长文，墙裂推荐！

1. 名词解释

1.1 Memorization 和 Generalization

Google Wide&Deep 论文中，通篇都是这两个词，必须搞懂是怎么回事！

这个是从人类的认知学习过程中演化来的。人类的大脑很复杂，它可以记忆 (memorize) 下每天发生的事情（麻雀可以飞，鸽子可以飞）然后泛化 (generalize) 这些知识到之前没有看到过的东西（有翅膀的动物都能飞）。但是泛化的规则有时候不是特别的准，有时候会出错（有翅膀的动物都能飞吗）。那怎么办那，没关系，记忆 (memorization) 可以修正泛化的规则 (generalized rules)，叫做特例（企鹅有翅膀，但是不能飞）。

这就是 Memorization 和 Generalization 的来由或者说含义。

Wide&Deep Mode 就是希望计算机可以像人脑一样，可以同时发挥 memorization 和 generalization 的作用。--Heng-Tze Cheng(Wide&Deep 作者)

1.2 Wide 和 Deep

同样，这两个词也是通篇出现，究竟什么意思你明白了没？

其实，Wide 也是一种特殊的神经网络，他的输入直接和输出相连。属于广义线性模型的范畴。Deep 就是指 Deep Neural Network，这个很好理解。Wide Linear Model 用于 memorization；Deep Neural Network 用于 generalization。左侧是 Wide-only，右

侧是 Deep-only, 中间是 Wide & Deep:

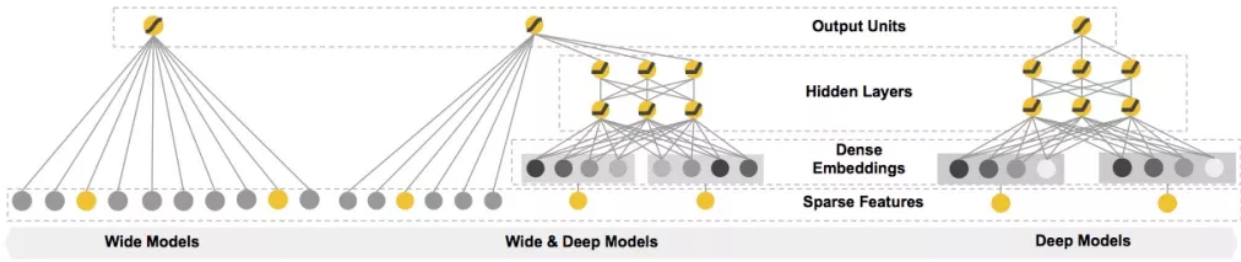


Figure 1: The spectrum of Wide & Deep models.

1.3 Cross-product transformation

Wide 中不断提到这样一种变换用来生成组合特征，也必须搞懂才行哦。它的定义如下：

$$\phi_k(\mathbf{x}) = \prod_{i=1}^d x_i^{c_{ki}} \quad c_{ki} \in \{0, 1\}$$

k 表示第 k 个组合特征。 i 表示输入 X 的第 i 维特征。 c_{ki} 表示这个第 i 维度特征是否要参与第 k 个组合特征的构造。 d 表示输入 X 的维度。那么到底有哪些维度特征要参与构造组合特征那？这个是你之前自己定好的，在公式中没有体现。

饶了一大圈，整这么一个复杂的公式，其实就是我们之前一直在说的 one-hot 之后的组合特征：仅仅在输入样本 X 中的特征 $\text{gender}=\text{female}$ 和特征 $\text{language}=\text{en}$ 同时为 1，新的组合特征 $\text{AND}(\text{gender}=\text{female}, \text{language}=\text{en})$ 才为 1。所以只要把两个特征的值相乘就可以了。

Cross-product transformation 可以在二值特征中学习到组合特征，并且为模型增加非线性

2. Wide & Deep Model

Memorization: 之前大规模稀疏输入的处理是：通过线性模型 + 特征交叉。所带来的 Memorization 以及记忆能力非常有效和可解释。但是 Generalization（泛化能力）需要更多的人工特征工程。

Generalization: 相比之下，DNN 几乎不需要特征工程。通过对低纬度的 dense embedding 进行组合可以学习到更深层次的隐藏特征。但是，缺点是有点 over-generalize（过度泛化）。推荐系统中表现为：会给用户推荐不是那么相关的物品，尤其是 user-item 矩阵比较稀疏并且是 high-rank（高秩矩阵）

两者区别: Memorization 趋向于更加保守，推荐用户之前有过行为的 items。相比之下，generalization 更加趋向于提高推荐系统的多样性（diversity）。

Wide & Deep: Wide & Deep 包括两部分：线性模型 + DNN 部分。结合上面两者的优点，平衡 memorization 和 generalization。原因：综合 memorization 和 generalization 的优点，服务于推荐系统。相比于 wide-only 和 deep-only 的模型，wide & deep 提升显著（这么比较是不是有点大。。。）

2.1 推荐系统

2.1.1 介绍

推荐系统分为两种：CF-Based（协同过滤）、Content-Based（基于内容的推荐）

- 协同过滤 (collaborative filtering) 就是指基于用户的推荐，用户 A 和 B 比较相似，那么 A 喜欢的 B 也可能喜欢
- 基于内容推荐是指物品 item1 和 item2 比较相似，那么喜欢 item1 的用户多半也喜欢 item2

2.1.2 线性模型

大规模的在线推荐系统中，logistic regression 应用非常广泛，因为其 **简单、易扩展、可解释性**。LR 的输入多半是二值化后的 one-hot 稀疏特征。Memorization 可以通过在稀疏特征上 cross-product transformations 来实现，比如：

AND(user_installed_app=QQ, impression_app=WeChat)，当特征 user_installed_app=QQ 和特征 impression_app=WeChat 取值都为 1 的时候，组合特征 AND(user_installed_app=QQ, impression_app=WeChat) 的取值才为 1，否则为 0。

推荐系统可以看成是一个 **search ranking 问题**，根据 query 得到 items 候选列表，然后对 items 通过 ranking 算法排序，得到最终的推荐列表。**Wide & Deep 模型是用来解决 ranking 问题的。**

如果仅仅使用线性模型：无法学习到训练集中没有的 query-item 特征组合。
Embedding-based Model 可以解决这个问题。

2.1.3 Embedding-Based

FM 和 DNN 都算是这样的模型，可以在很少的特征工程情况下，通过学习一个低纬度的 embedding vector 来学习训练集中从未见过的组合特征。

FM 和 DNN 的缺点在于：当 query-item 矩阵是稀疏并且是 high-rank 的时候（比如 user 有特殊的爱好，或 item 比较小众），很难非常效率的学习出低维度的表示。这种情况下，大部分的 query-item 都没有什么关系。但是 dense embedding 会导致几乎所有的 query-item 预测值都是非 0 的，这就导致了推荐过度泛化，会推荐一些不那么相关的物品。

相反，linear model 却可以通过 cross-product transformation 来记住这些 **exception rules**，而且仅仅使用了非常少的参数。

总结一下：

线性模型无法学习到训练集中未出现的组合特征；FM 或 DNN 通过学习 embedding vector 虽然可以学习到训练集中未出现的组合特征，但是会过度泛化。

Wide & Deep Model 通过组合这两部分，解决了这些问题。

2.1.4 工作流程

总的来说，推荐系统 = Retrieval + Ranking

推荐系统工作流程如下：

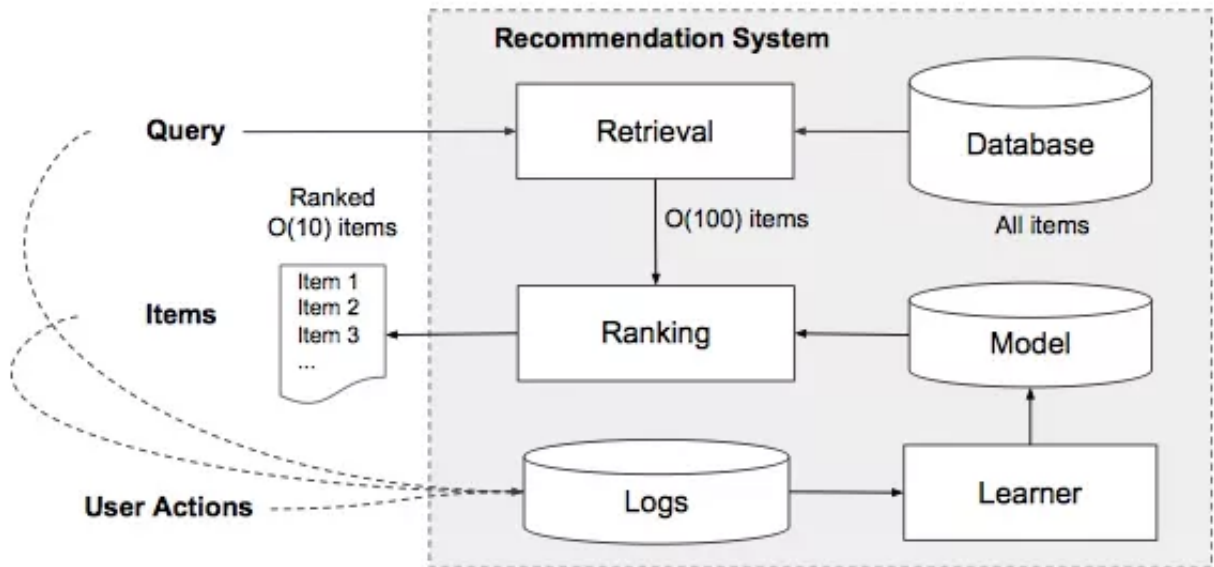


Figure 2: Overview of the recommender system.

想象这样一个实际情况：**我们打开 Google APP store，首页展示给我们一些 APP，我们点击或者下载或者购买了其中一个 APP。** 在这样一个流程中，推荐系统是如何工作的那？

我们对比上面的图一点点来说：

Query:当我们打开 APP Store 的时候，就产生了一次 Query，它包含两部分的特征：**User features, contextual features**。UserFeatures 包括性别、年龄等人口统计特征，ContextualFeatures 包括设备、时间等上下文特征。

Items:APP store 接着展示给我们一系列的 app，这些 app 就是推荐系统针对我们的 Query 给出的推荐。这个也被叫做 **impression**。

User Actions:针对推荐给你的任何一个 APP，我们都可以点击、下载、购买等操作。也就是说推荐给你的 APP，你产生了某种行为。这不正是我们的最终目的吗！

Logs:Logs = Query + Impression + UserAction 查询、展示列表、操作会被记录到 logs 中作为 **训练数据** 给 Learner 来学习。

Retrieval: 假如让你来想一个最简单的推荐系统，**针对这一次 Query**，来给出推荐列表。你能想到的最简单，最暴力的做法是什么那？

给数据库中所有的 APP 都打一个分数，然后按照分数从高到低返回前 N 个（比如说前 100 个）

但是有个问题，这样数据库中的 APP 实在是太多了，为了保证响应时间，这样做太慢了！**Retrieval** 就是用来解决这个问题的。它会利用机器学习模型和一些人为定义的规则，来返回最匹配当前 Query 的一个小的 items 集合，这个集合就是最终的推荐列表的候选集。

Ranking:

今天的主角 Wide&Deep Model 就是用来做这个事情的啦。

前面 Learner 学习到了一个 Model，利用这个 Model 对 Retrieval 给出的候选集 APP 打分！并按照打分从高到低来排序，并返回前 10 个 APP 作为最终的推荐结果展示给用户。

Retrieval system 完了之后，就是 Ranking system。Retrieval 减小了候选 items 池，Ranking system 要做的就是对比当前的 Query，对 Candidate pool 里面的所有 item 打分！

得分 score 表示成 $P(y|x)$ ，表示的是一个条件概率。y 是 label，表示 user 可以采取的 action，比如点击或者购买。x 表示输入，特征包括：

- User features (年龄、性别、语言、民族等)
- Contextual features(上下文特征：设备，时间等)
- Impression features (展示特征：app age、app 的历史统计信息等)

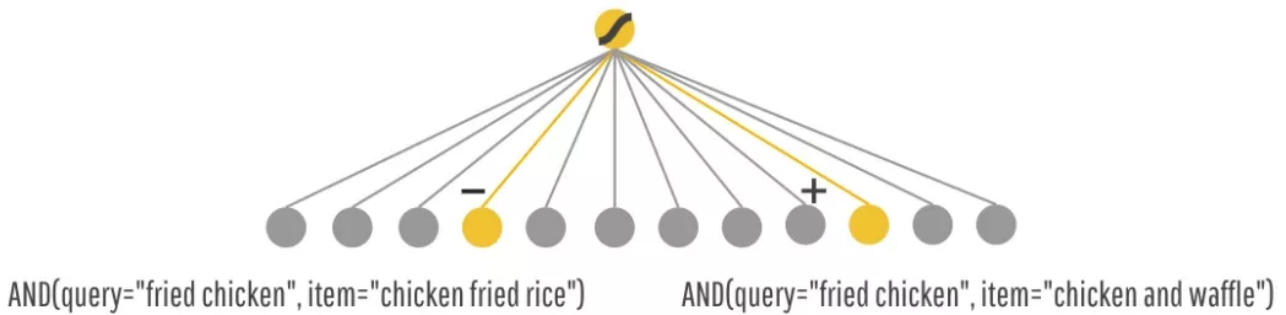
2.2 Wide Part

Wide Part 其实是一个广义的线性模型

使用特征包括：

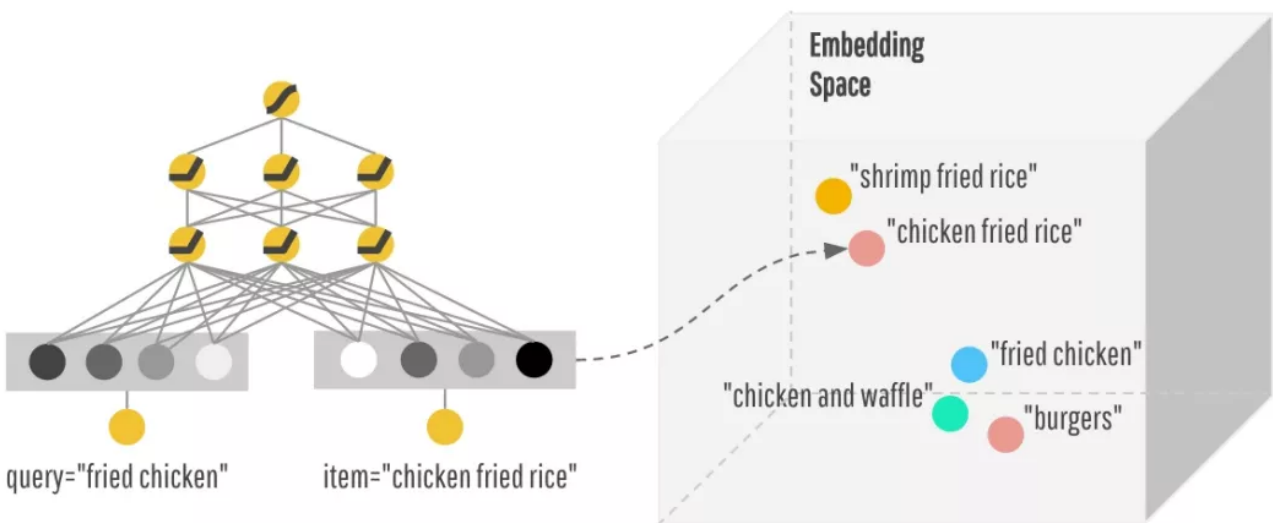
- raw input 原始特征
- cross-product transformation 组合特征

接下来我们用同一个例子来说明：你给 model 一个 query（你想吃的美食），model 返回给你一个美食，然后你购买 / 消费了这个推荐。也就是说，推荐系统其实要学习的是这样一个条件概率： **$P(\text{consumption} | \text{query}, \text{item})$**



Wide Part 可以对一些特例进行 memorization。比如 $\text{AND}(\text{query}=\text{"fried chicken"}, \text{item}=\text{"chicken fried rice"})$ 虽然从字符角度来看很接近，但是实际上完全不同的东西，那么 Wide 就可以记住这个组合是不好的，是一个特例，下次当你再点炸鸡的时候，就不会推荐给你鸡肉炒米饭了。

2.3 Deep Part

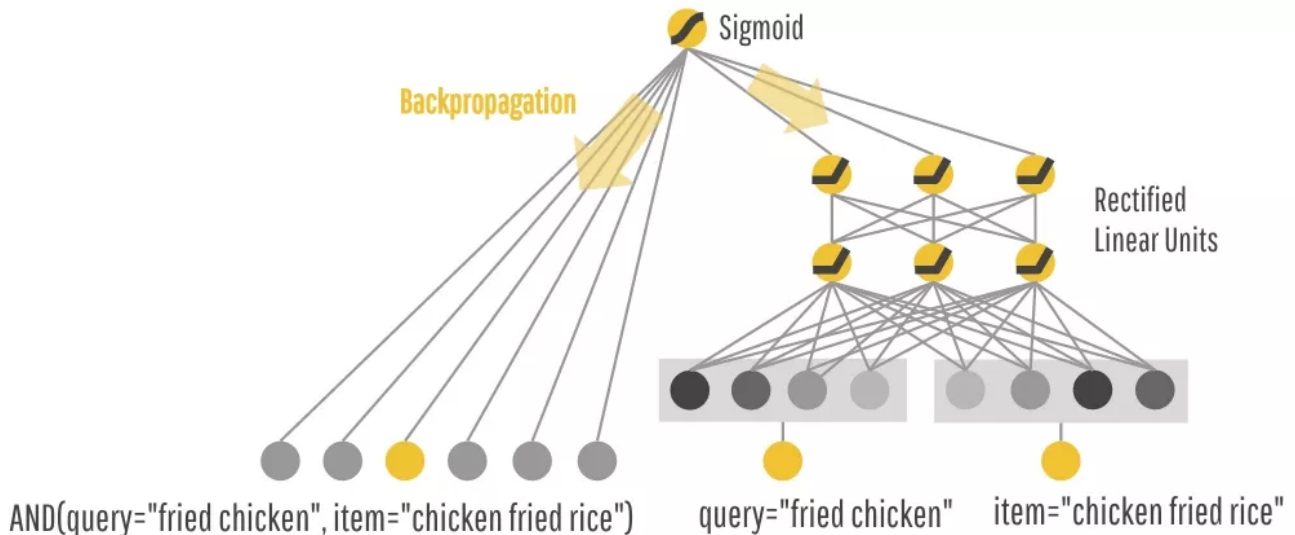


Deep Part 通过学习一个低纬度的 dense representation（也叫做 embedding vector）对于每一个 query 和 item，来 **泛化** 给你推荐一些字符上看起来不那么相关，但是你可能也是需要的。比如说：你想要炸鸡，Embedding Space 中，炸鸡和汉堡很接近，所以也会给你推荐汉堡。

Embedding vectors 被随机初始化，并根据最终的 loss 来反向训练更新。这些低维度的 dense embedding vectors 被作为第一个隐藏层的输入。隐藏层的激活函数通常使用 ReLU。

2.4 模型训练

原始的稀疏特征，在两个组件中都会用到，比如query="fried chicken" item="chicken fried rice"：



在训练的时候，根据最终的 loss 计算出 gradient，反向传播到 Wide 和 Deep 两部分中，分别训练自己的参数。也就是说，**两个模块是一起训练的**，注意这不是模型融合。

- Wide 部分中的组合特征可以 **记住** 那些稀疏的，特定的 rules
- Deep 部分通过 Embedding 来 **泛化** 推荐一些相似的 items

Wide 模块通过组合特征可以很效率的学习一些特定的组合，但是这也导致了他并不能学习到训练集中没有出现的组合特征。所幸，Deep 模块弥补了这个缺点。另外，因为是一起训练的，wide 和 deep 的 size 都减小了。wide 组件只需要填补 deep 组件的不足就行了，所以需要比较少的 cross-product feature transformations，而不是 full-size wide Model。

论文中的实现：

- 训练方法是用 mini-batch stochastic optimization。
- Wide 组件是用 FTRL (Follow-the-regularized-leader) + L1 正则化学习。
- Deep 组件是用 AdaGrad 来学习。

3. 系统实现

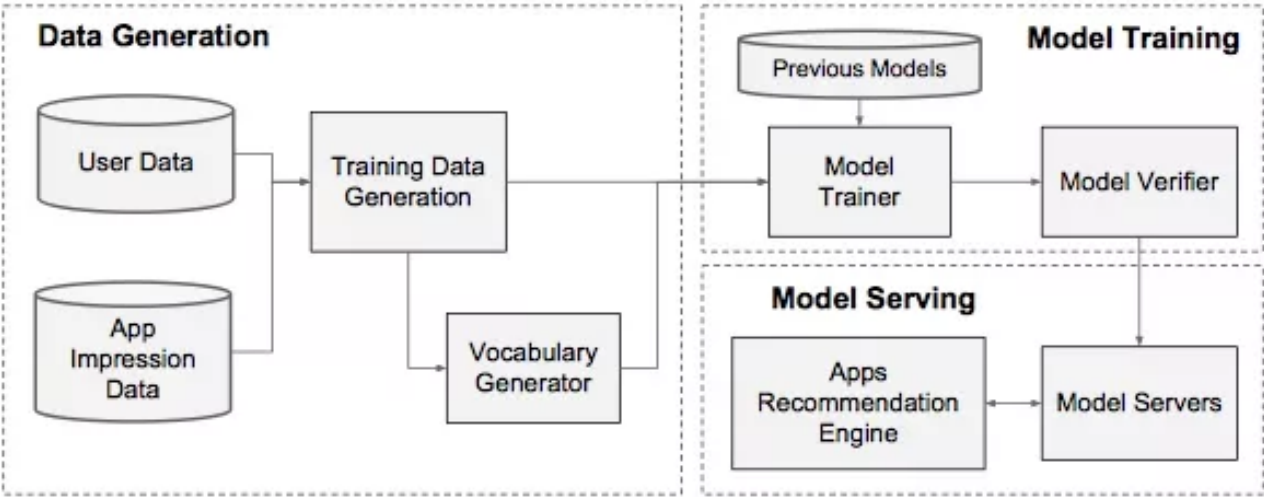


Figure 3: Apps recommendation pipeline overview.

3.1 训练数据生成

请大家一定格外的关注训练数据到底是什么？这对于理解推荐系统到底是怎么回事很重要。先给出结论：

一次展示中的一个 Item 就是一条样本。

样本的 label 要根据实际的业务需求来定，比如 APP Store 中想要提高 APP 的下载率，那么就以这次展示的这个 Item 中用户有没有下载，作为 label。下载了 label 为 1，否则为 0.说白了，模型需要预测，在当前 Query 的条件下，对于这个 Item，用户下载的条件概率。

离散特征 map 成 id

过滤掉出现次数少于设定阈值的离散特征取值，然后把这些全部 map 成一个 ID。离散特征取值少，就直接编号。多的话可能要 Hash

连续特征通过分位数规范化到 [0,1]

先把所有的值分成 n 份，那么属于第 i 部分的值规范化之后的值为 $(i - 1)/(n - 1)$ 。

3.2 模型训练

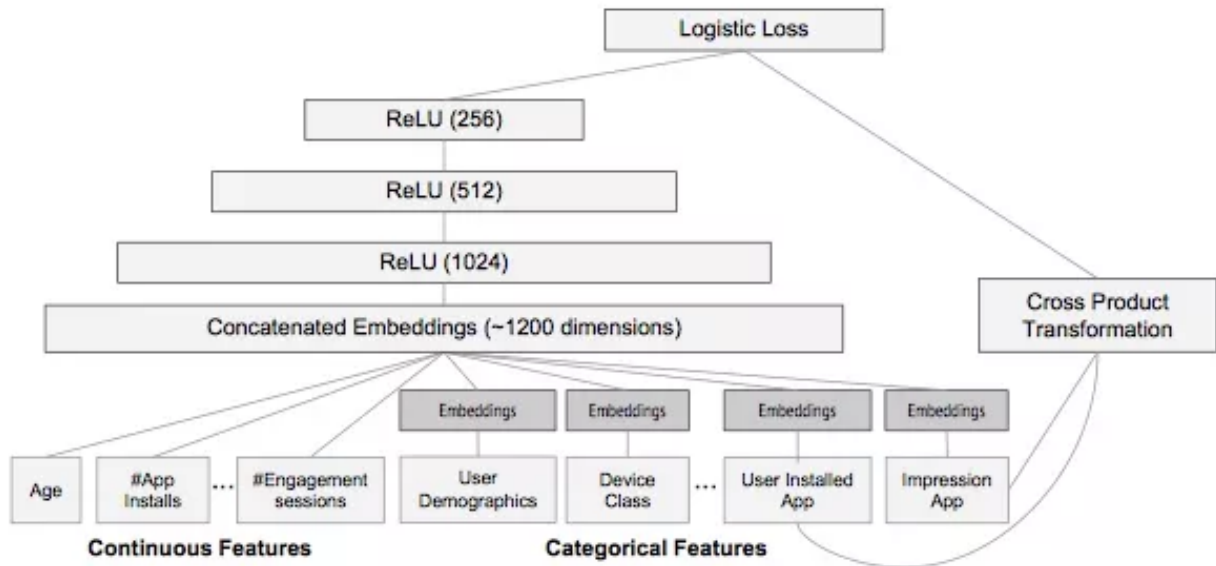


Figure 4: Wide & Deep model structure for apps recommendation.

Deep 部分使用的特征：

- 连续特征
- Embedding 后的离散特征，Item 特征

Wide 部分使用的特征：

- Cross Product Transformation 生成的组合特征

但是，官方给出的示例代码中，Wide 部分还使用了离散特征（没有 one-hot）。也有大佬说不用特征交叉效果也很好，这个大家在实际项目中就以实验为准吧。

每当有新的数据到达的时候，就要重新训练。如果每次都从头开始会非常耗时，Google 给出的解决办法是：实现了 warm-starting system, 它可以用之前模型的 embeddings 和线性模型的 weights 来初始化新的模型。

Embedding 维度大小的建议：Wide&Deep 的作者指出，从经验上来讲 Embedding 层的维度大小可以用如下公式来确定：

$$k\sqrt[4]{n_i}$$

n 是原始维度上特征不同取值的个数； k 是一个常数，通常小于 10。

3.3 线上使用

模型被部署之后。每一次请求，服务器会收到一系列的 app 候选集（从 app retrieval system 输出的）以及 user features（用于为每一个 app 打分）。然后，模型会把 APP 按照 score 排序，并展示给 user，按照这个顺序展示。score 就是对于 wide & deep 模型的一次 forward pass。为了控制每一次 request 响应时间在 10ms 内，引入了并行化技术。将 app 候选集分成多个小的 batches，并行化预测 score。

4. 适用范围

Wide & Deep Model 适用于输入非常稀疏的大规模分类或回归问题。比如推荐系统、search、ranking 问题。输入稀疏通常是由离散特征有非常非常多个可能的取值造成的，one-hot 之后维度非常大。

5. 优缺点

缺点：Wide 部分还是需要人为的特征工程。优点：实现了对 memorization 和 generalization 的统一建模。

6. 代码实践

代码放到 github 上了：

https://github.com/gutouyu/ML_CIA/tree/master/Wide%26Deep

数据集：<https://archive.ics.uci.edu/ml/machine-learning-databases/adult>

代码主要包括两部分：Wide Linear Model 和 Wide & Deep Model。

数据集长这样，最后一行是 label，预测收入是否超过 5 万美元，二分类问题。

Column Name	Type	Description
age	Continuous	The age of the individual
workclass	Categorical	The type of employer the individual has (government, military, private, etc.).
fnlwgt	Continuous	The number of people the census takers believe that observation represents (sample weight). Final weight will not be used.
education	Categorical	The highest level of education achieved for that individual.
education_num	Continuous	The highest level of education in numerical form.
marital_status	Categorical	Marital status of the individual.
occupation	Categorical	The occupation of the individual.
relationship	Categorical	Wife, Own-child, Husband, Not-in-family, Other-relative, Unmarried.
race	Categorical	Amer-Indian-Eskimo, Asian-Pac- Islander, Black, White, Other.
gender	Categorical	Female, Male.
capital_gain	Continuous	Capital gains recorded.
capital_loss	Continuous	Capital Losses recorded.
hours_per_week	Continuous	Hours worked per week.
native_country	Categorical	Country of origin of the individual.
income_bracket	Categorical	">50K" or "<=50K", meaning whether the person makes more than \$50,000 annually.

6.1 Wide Linear Model

离散特征 处理分为两种情况：

- 知道所有的不同取值，而且取值不多。

```
tf.feature_column.categorical_column_with_vocabulary_list
```

- 不知道所有不同取值，或者取值非常多。

```
tf.feature_column.categorical_column_with_hash_bucket
```



```
## 3.1 Base Categorical Feature Columns
# 如果我们知道所有的取值，并且取值不是很多
relationship = tf.feature_column.categorical_column_with_vocabulary_list(
    'relationship', [
        'Husband', 'Not-in-family', 'Wife', 'Own-child', 'Unmarried',
        'Other-relative'
    ]
)
# 如果不知道有多少取值
occupation = tf.feature_column.categorical_column_with_hash_bucket(
    'occupation', hash_bucket_size=1000
)
```

原始连续特征： `tf.feature_column.numeric_column`

```
# 3.2 Base Continuous Feature Columns
age = tf.feature_column.numeric_column('age')
education_num = tf.feature_column.numeric_column('education_num')
capital_gain = tf.feature_column.numeric_column('capital_gain')
capital_loss = tf.feature_column.numeric_column('capital_loss')
hours_per_week = tf.feature_column.numeric_column('hours_per_week')
```

规范化到 [0,1] 的连续特征： `tf.feature_column.bucketized_column`

```
# 3.2.1 连续特征离散化
# 之所以这么做是因为：有些时候连续特征和label之间不是线性的关系。
# bucketization 装桶
# 10个边界， 11个桶
age_buckets = tf.feature_column.bucketized_column(
    age, boundaries=[18, 25, 30, 35, 40, 45, 50, 55, 60, 65])
```

组合特征 / 交叉特征： `tf.feature_column.crossed_column`

```
# 3.3 组合特征/交叉特征
education_x_occupation = tf.feature_column.crossed_column(
    ['education', 'occupation'], hash_bucket_size=1000)

age_buckets_x_education_x_occupation = tf.feature_column.crossed_column(
    [age_buckets, 'education', 'occupation'], hash_bucket_size=1000
)
```

组装模型： 这里主要用了 *离散特征 + 组合特征*

4. 模型

"""

之前的特征:

1. CategoricalColumn
2. NumericalColumn
3. BucketizedColumn
4. CrossedColumn

这些特征都是FeatureColumn的子类, 可以放到一起

"""

```
base_columns = [
    education, marital_status, relationship, workclass, occupation,
    age_buckets,
]

crossed_column = [
    tf.feature_column.crossed_column(
        ['education', 'occupation'], hash_bucket_size=1000
    ),
    tf.feature_column.crossed_column(
        [age_buckets, 'education', 'occupation'], hash_bucket_size=1000
    )
]

model_dir = "./model/wide_component"
model = tf.estimator.LinearClassifier(
    model_dir=model_dir, feature_columns=base_columns + crossed_column
)
```

训练 & 评估:

```
# 5. Train & Evaluate & Predict
model.train(input_fn=lambda: input_fn(data_file=train_file, num_epochs=1, shuffle=True, batch_size=512))
results = model.evaluate(input_fn=lambda: input_fn(val_file, 1, False, 512))
for key in sorted(results):
    print("{0:20}: {1:.4f}".format(key, results[key]))
```

运行截图:

```
Parsing ./data/adult.data
accuracy          : 0.8434
accuracy_baseline  : 0.7592
auc               : 0.8940
auc_precision_recall: 0.7230
average_loss      : 0.3402
global_step       : 192.0000
label/mean        : 0.2408
loss              : 173.0865
prediction/mean    : 0.2405
```

6.2 Wide & Deep Model

Deep 部分用的特征：未处理的连续特征 + Embedding(离散特征)

在 Wide 的基础上，增加 Deep 部分：

离散特征 embedding 之后，和连续特征串联。

```
# 3. The Deep Model: Neural Network with Embeddings
"""
1. Sparse Features -> Embedding vector -> 串联(Embedding vector, 连续特征) -> 输入到Hidden Layer
2. Embedding Values随机初始化
3. 另外一种处理离散特征的方法是: one-hot or multi-hot representation. 但是仅仅适用于维度较低的, embedding是更加通用的做法
4. embedding_column(embedding);indicator_column(multi-hot);
"""
deep_columns = [
    age,
    education_num,
    capital_gain,
    capital_loss,
    hours_per_week,
    tf.feature_column.indicator_column(workclass),
    tf.feature_column.indicator_column(education),
    tf.feature_column.indicator_column(marital_status),
    tf.feature_column.indicator_column(relationship),

    # To show an example of embedding
    tf.feature_column.embedding_column(occupation, dimension=8)
]
```

组合 Wide & Deep: DNNLinearCombinedClassifier

```
# 4. Combine Wide & Deep
model = tf.estimator.DNNLinearCombinedClassifier(
    model_dir = model_dir,
    linear_feature_columns=base_columns + crossed_columns,
    dnn_feature_columns=deep_columns,
    dnn_hidden_units=[100,50]
)
```

训练 & 评估:

```
for n in range(train_epochs // epochs_per_eval):
    model.train(input_fn=lambda: input_fn(train_file, epochs_per_eval, True, batch_size))
    results = model.evaluate(input_fn=lambda: input_fn(
        test_file, 1, False, batch_size))

    # Display Eval results
    print("Results at epoch {}".format((n+1) * epochs_per_eval))
    print('-'*30)

    for key in sorted(results):
        print("{0:20}: {1:.4f}".format(key, results[key]))
```

运行结果:

```
Parsing ./data/adult.test
Results at epoch 2

-----

accuracy                : 0.8486
accuracy_baseline       : 0.7638
auc                     : 0.8931
auc_precision_recall    : 0.7550
average_loss            : 0.3385
global_step             : 8145.0000
label/mean              : 0.2362
loss                    : 13.5087
prediction/mean         : 0.2346
Parsing ./data/adult.data
```

```
Parsing ./data/adult.data
```

```
Parsing ./data/adult.test
```

```
Results at epoch 4
```

```
accuracy           : 0.8457
accuracy_baseline   : 0.7638
auc                : 0.8955
auc_precision_recall: 0.7421
average_loss        : 0.3433
global_step         : 9774.0000
label/mean          : 0.2362
loss                : 13.6987
prediction/mean      : 0.2429
```

```
Parsing ./data/adult.data
```

```
Parsing ./data/adult.test
```

```
Results at epoch 6
```

```
accuracy           : 0.8476
accuracy_baseline   : 0.7638
auc                : 0.8954
auc_precision_recall: 0.7495
average_loss        : 0.3367
global_step         : 11403.0000
label/mean          : 0.2362
loss                : 13.4349
prediction/mean      : 0.2404
```