论文调研报告

Nov 12 2017 黄威

「Field-aware Factorization Machines for CTR Prediction」

FM 和 FFM 模型都是最近几年提出的模型，凭借其在数据量比较大并且特征稀疏的情况下，仍然能够得到优秀的性能和效果的特性，屡次在各大公司举办的 CTR 预估比赛中获得不错的战绩。

FFM（Field-aware Factorization Machine）最初的概念来自 Yu-Chin Juan（阮毓钦，毕业于中国台湾大学，现在美国Criteo工作）与其比赛队员，是他们借鉴了来自 Michael Jahrer 的论文「Ensemble of Collaborative Filtering and Feature Engineered Models for Click Through Rate Prediction」中的 field 概念提出了 FM 的升级版模型，该篇于 2016 年发布。

Introduction

在 CTR 任务中，大部分样本数据特征是非常稀疏的，使用 One-hot 编码会造成特征空间剧增。而通过对大量样本数据的观察发现某些特征经过关联之后，与最后预测的 label 的相关性就会提高。

因此使用多项式模型，考虑特征之间的组合会更加直观。

Related Work

Poly 2

在多项式模型中，特征 x_i 和 x_j 的组合采用 $x_i x_j$ 表示，即 x_i 和 x_j 都非零时，组合特征 $x_i x_j$ 才有意义。典型的二阶多项式模型的表达式如下：

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} w_{ij} x_i x_j$$

其中，n 代表样本的特征数量，x_i 是第 i 个特征的值，w_0、w_i、w_{ij} 是模型参数。这也是 Poly2（Degree-2 Polynomial Model ）模型的表达式。

从公式（1）可以看出，组合特征的参数一共有 $\frac{n(n-1)}{2}$ 个，任意两个参数都是独立的。然而，在数据稀疏性普遍存在的实际应用场景中，二次项参数的训练是很困难的。其原因是，每个参数 w_{ij} 的训练需要大量 x_i 和 x_j 都非零的样本；由于样本数据本来就比较稀疏，满足"x_i 和 x_j 都非零"的样本将会非常少。训练样本的不足，很容易导致参数 w_{ij} 不准确，最终将严重影响模型的性能。

FM

因此，为了解决二次项参数的训练问题，FM 模型基于矩阵分解的思想，在原有表达式对其稍微进行了一点改动：

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle v_i, v_j \rangle x_i x_j$$

其中，n 代表样本的特征数量，x_i 是第 i 个特征的值，w_0、w_i、v_i、v_j 是模型参数。v_i、v_j 表示长度为 k 的隐向量，包含 k 个描述特征的因子，k 为超参数，$\langle v_i, v_j \rangle$ 表示向量点积。

根据公式，二次项的参数数量减少为 kn 个，远小于 Poly2 模型的参数数量。而且，最重要的是，参数的因子化使得 $x_i x_j$ 的参数和 $x_i x_t$ 的参数不再是相互独立的，因此我们可以在样本稀疏的情况下相对合理地估计 FM 的二次项参数。具体来说，$x_i x_j$ 和 $x_i x_t$ 的系数分别为 $\langle v_i, v_j \rangle$ 和 $\langle v_i, v_t \rangle$，它们之间有共同项 v_i。也就是说，所有包含“x_i 的非零组合特征”（存在某个 $j \neq i$，使得 $x_i x_j \neq 0$）的样本都可以用来学习隐向量 v_i，这很大程度上避免了数据稀疏性造成的影响。而在 Poly2 模型中，$w_i w_j$ 和 $w_i w_t$ 是相互独立的。

FM 模型比起 Poly 2 模型，其优点显而易见：

- 通常 CTR 任务的数据量都是十分庞大的，FM 的参数数量比起 Poly 2 要明显减少，减少了模型的训练时间。
- FM 的参数并不是相互独立，可以从其他的参数学习得到，在样本稀疏性非常明显的情况下，能够的得到更好更准确的参数，提高了模型的精度。

FFM

而我们今天的主角 FFM 模型，在 FM 模型表达式的基础上，更进一步：

$$y(\mathbf{x}) = w_0 + \sum_{i=1}^{n} w_i x_i + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \left\langle v_{i,f_2}, v_{j,f_1} \right\rangle x_i x_j$$

其中，n 代表样本的特征数量，x_i 是第 i 个特征的值，w_0、w_i、v_{i,f_2}、v_{j,f_1} 是模型参数。v_{i,f_2}、v_{j,f_1} 表示长度为 k 的隐向量，包含 k 个描述特征的因子，k 为超参数，$\langle v_i, v_j \rangle$ 表示向量点积。$\langle v_i, v_j \rangle$ 变成了 $\left\langle v_{i,f_2}, v_{j,f_1} \right\rangle$，其中 f_1、f_2 表示 x_i 与 x_j 所属的 field。

举个例子：

Clicked	Publisher(P)	Advertiser(A)	Gender(G)
Yes	ESPN	Nike	Male

对于 FM 模型而言，$\phi_{FM}(w, x)$ 为：

$$\phi_{FM}(w, x) = w_{ESPN} \cdot w_{Nike} + w_{ESPN} \cdot w_{Male} + w_{Nike} \cdot w_{Male}$$

而对于 FFM 模型而言，$\phi_{FFM}(w, x)$ 为：

$$\phi_{FM}(w, x) = w_{ESPN,A} \cdot w_{Nike,P} + w_{ESPN,G} \cdot w_{Male,P} + w_{Nike,G} \cdot w_{Male,A}$$

在 FM 模型中，每个特征只有一个隐向量需要学习，而 FFM 则需要学习多个隐向量，取决于与其组合的特征的 field。例如对于特征 ESPN 的参数 w_{ESPN}，它可以通过与特征 Nike 组合（$w_{ESPN} \cdot w_{Nike}$）或者 特征 Male 的组合（$w_{ESPN} \cdot w_{Male}$）来学习。然而，由于 Nike 与 Male 分别属于不同的 field，因此（ESPN, NIKE）与（ESPN, Male）所造成的影响是不一样。

$w_{ESPN,A}$ 是因为 Nike 的 field 为 Advertiser（A），$w_{ESPN,G}$ 是因为 Male 的 field 为 Gender（G）。

FFM 模型数据格式

$$label \quad field_1 : feat_1 : val_1 \quad field_2 : feat_2 : val_2 \quad \dots$$

对于大多数特征都可以用这样的方法表示，但是对于一些特征：

1. 类别型特征

   例如上表的数据，处理成 FFM 数据格式的话：

   $$1 \quad P : ESPN : 1 \quad A : Nike : 1 \quad G : Male : 1$$

2. 数值型特征

   对于数值型特征，例如下表数据：

Accepted	AR	Hidx	Cite
Yes	45.73	2	3
No	1.04	100	50000

   有两种处理成 FFM 数据格式的方式：

   - Treat each feature as a dummy field:

     $$1 \quad AR : AR : 45.73 \quad Hidx : Hidx : 2 \quad Cite : Cite : 3$$

   - Discretize each numerical feature to a categorical one:

     $$1 \quad AR : 45 : 1 \quad Hidx : 2 : 1 \quad Cite : 3 : 1$$

注意到第二种处理方法，将 AR 这 field 中的特征值进行了处理，将 45.73 处理成 45.7、45、40 甚至是 $int(log(45.73))$ 都是可行的。

3. 单 field 特征

   经常在 NLP 任务上出现，例如如下表数据：

good mood	sentence
Yes	Hooray! Our paper is accepted!
No	Well, our paper is rejected..

   所有的特征都属于同一个 field — `sentence`，那么我们的做法就是将 `sentence` 这一 field 放置在每个分词特征之前，相当于是从 FFM 降低到 FM。

Experiments

Experiment Setting

1. 数据集

数据集为 Kaggle 两个比赛的数据集：

- Criteo
- Avazu

Data Set	instances	features	fields
Criteo	45,840,617	10^6	39
Avazu	40,428,968	10^6	33

2. 模型训练及参数

模型的优化方法为普通的 SG （Stochastic Gradient），再加上 FFM 中需要我们设定的超参数 k，因此模型的参数为：

- k 隐向量的长度；
- λ 学习率；
- η 步长；

3. 实验结果

Model and implementation	parameters	training time (seconds)	public set logloss	public set rank	private set logloss	private set rank
LM-SG	$\eta = 0.2, \lambda = 0, t = 13$	688	0.46262	93	0.46224	91
LM-LIBLINEAR-CD	$s = 7, c = 2$	2,280	0.46239	91	0.46201	89
LM-LIBLINEAR-Newton	$s = 0, c = 2$	10,380	0.46320	105	0.46263	96
Poly2-SG	$\eta = 0.2, \lambda = 0, B = 10^7, t = 10$	18,581	0.45012	14	0.44996	15
Poly2-LIBLINEAR-Hash-CD	$s = 7, c = 2$	62,144	0.44917	14	0.44897	14
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 40, t = 8$	2,362	0.44935	14	0.44923	14
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100, t = 9$	4,017	0.44826	10	0.44818	10
LIBFM	$\lambda = 40, k = 40, t = 20$	23,700	0.45012	14	0.45000	15
LIBFM	$\lambda = 40, k = 40, t = 50$	131,000	0.44904	14	0.44887	14
LIBFM	$\lambda = 40, k = 100, t = 20$	54,320	0.44853	11	0.44834	11
LIBFM	$\lambda = 40, k = 100, t = 50$	398,800	0.44794	9	0.44778	8
FFM	$\eta = 0.2, \lambda = 2 \times 10^{-5}, k = 4, t = 9$	10,454	0.44613	3	0.44603	3

(a) Criteo

Model and implementation	parameters	training time (seconds)	public set logloss	public set rank	private set logloss	private set rank
LM-SG	$\eta = 0.2, \lambda = 0, t = 10$	1360	0.39023	61	0.38833	64
LM-LIBLINEAR-CD	$s = 7, c = 1$	3045	0.39131	115	0.38943	119
LM-LIBLINEAR-Newton	$s = 0, c = 1$	4090	0.39269	182	0.39079	183
Poly2-SG	$\eta = 0.2, \lambda = 0, B = 10^7, t = 10$	8,808	0.38544	10	0.38342	10
Poly2-LIBLINEAR-Hash-CD	$s = 7, c = 1$	7,288	0.38510	10	0.38298	9
Poly2-LIBLINEAR-Hash-Newton	$s = 0, c = 1$	108,184	0.38633	11	0.38430	11
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 40, t = 8$	2,700	0.38616	11	0.38409	11
FM	$\eta = 0.05, \lambda = 2 \times 10^{-5}, k = 100, t = 9$	3,612	0.38666	11	0.38463	11
LIBFM	$\lambda = 40, k = 40, t = 20$	18,712	0.39137	122	0.38963	127
LIBFM	$\lambda = 40, k = 40, t = 50$	41,720	0.39786	935	0.39635	943
LIBFM	$\lambda = 40, k = 100, t = 20$	39,719	0.39644	747	0.39470	755
LIBFM	$\lambda = 40, k = 100, t = 50$	91,210	0.40740	1,129	0.40585	1,126
FFM	$\eta = 0.2, \lambda = 2 \times 10^5, k = 4, t = 4$	4,766	0.38395	6	0.38205	6

(b) Avazu

Results and Discussion

相较于 LM、Poly 2 以及 FM 模型，FFM 在数据集上表现更好，拥有更高的准确率。

「About Feature Engineering」

Step 1: Exploratory Data Analysis(EDA)

原始特征

- 用户特征
  - `user_id(msno)`
  - `city`
  - `gender`
  - `bd`
- 音乐特征
  - `song_id`
  - `song_length`
  - `genre_ids`
  - `language`
  - `name`
  - `artist_name`
  - `composer`
  - `lyricist`
  - `isrc`
- 交互特征
- 上下文特征
  - `registered_via`
  - `registration_init_time`
  - `exipration_date`
  - `source_system_tab`
  - `source_screen_name`
  - `source_type`

Step 2: Feature Engineering

Feature Engineering 是把 raw data 转换成 features 的整个过程的总称。基本上特征工程就是个手艺活，制作的好坏全凭人的功夫，往细了讲，便是创造力与经验。

以推荐系统为例，数据集中的特征可以分成以下四种：

- 用户特征：用户本身的各种属性，例如 user id、性别、所在的城市等
- 音乐特征：音乐本身的各种属性，例如 item id、歌曲名、演唱者、作曲家、作词家、音乐风格分类等
- 交互特征：用户对音乐做出的某项行为，该行为的 aggregation 或交叉特征，例如最近听的歌曲的曲风分布或喜爱的歌手的类型
- 上下文特征：用戶对音乐做出的某项行，该行为的 metadata，例如注册的时间、使用的设备等

有些特征是在资料 EDA 阶段就可以拿到，有些特征则需要额外的步骤（例如如透过外部的 API 或者其他模型）才能取得。

联想特征

- 用户特征
  - `user_days_between_registration_today`：该用户的注册时间距离今天过了几天
  - `user_days_between_exipration_today`：该用户的退订时间距离今天过了几天
- 音乐特征
  - `song_id`
- 交互特征
- 上下文特征
  - `als_model_prediction`：来自 ALS 模型的预测值，该用户对某音乐的偏好程度
  - `gbdt_model_index`: 来自 GBDT 模型的 tree index，某 observation 的自动特征

2.1 Missing Value Imputation 缺失值处理

最简单暴力的做法当然就是直接 drop 掉那些含有缺失值的 rows。

- 针对 numerical 特征的缺失值，可以用以下方式取代：
  - `0`，缺点是可能会混淆其他本来就是 0 的数值
  - `-999`，用某个正常情况下不会出现的数值代替，但是选得不好可能会变成异常值，要特别对待
  - Mean，平均数
  - Median，中位数，跟平均数相比，不会被异常值干扰
- 针对 categorical 特征的缺失值，可以用以下方式取代：
  - Mode，众数，最常见的值
  - 改成 "Others" 之类的值

假设你要填补 `age` 这个特征，然后你有其他例如 `gender` 这样的特征，你可以分别计算男性和女性的 `age` 的 mean、median 和 mode 来填补缺失值；更复杂一点的方式是，你可以把没有缺失值的数据挑出来，用它们来训练一个 regression 或 classification 模型，用这个模型来预测缺失值。

不过其实有些算法是可以容许缺失值的，这时候可以新增一个`has_missing_value` 栏位（称为 NA indicator column）。

2.2 Outliers Detection 野点处理

发现离群值最直观的方式就是画图表，针对单一特征可以使用 box plot；两两特征则可以使用 scatter plot。

处置离群值的方式通常是直接删除或是做变换（例如 log transformation 或 binning），当然你也可以套用处理缺失值的方式。

2.3 Duplicate Entries Removal 异常值处理

Duplicate 或 redundant 尤其指的是那些 features 都一样，但是 target variable 却不同的数据。

2.4 Feature Scaling 特征缩放

- **2.4.1 Standardization 标准化**

原始数据集中，因为各个特征的含义和单位不同，每个特征的取值范围可能会差异很大。例如某个二元特征的范围是 0 或 1，另一个特征的范围可能是 [0, 1000000]，由于取值范围相差过大导致了模型可能会更偏向于取值范围较大的那个特征。解决的办法就是把各种不同 scale 的特征转换成同样的 scale，称为标准化或正规化。

狭义来说，标准化专门指的是通过计算 z-score，让数据的 mean 为 0、 variance 为 1。

- **2.4.2 Normalization 归一化**

归一化是指把每个样本缩放到单位范数（每个样本的范数为 1），适用于计算 dot product 或者两个样本之间的相似性。除了标准化、归一化之外，其他还有通过最大、最小值，把数据的范围缩放到 [0, 1] 或 [-1, 1] 的区间缩放法，不过这个方法容易受异常值的影响。

标准化是分别对单一特征进行（针对 column）；归一化是对每个 observation 进行（针对 row）。

- **对 SVM、logistic regression 或其他使用 squared loss function 的演算法来说，需要 standardization；**
- **对 Vector Space Model 来说，需要 normalization；**
- **对 tree-based 的算法，基本上都不需要标准化或归一化，它们对 scale 不敏感。**

2.5 Feature Transformation 特征变换

针对连续值特征：

- **2.5.1 Rounding**

  某些精度有到小数点后第 n 位的特征，如果你其实不需要那么精确，可以考虑 `round(value * m)` 或 `round(log(value))` 这样的做法，甚至可以把 round 之后的数值当成 categorical 特征。

- **2.5.2 Log Transformation**

  因为 x 越大，log(x) 增长的速度就越慢，所以取 log 的意义是可以 compress 大数和 expand 小数，换句话说就是压缩 "long tail" 和展开 "head"。假设 x 原本的范围是 [100, 1000]，log(x, 10) 之后的范围就变成 [2, 3] 了。也常常使用 log(1 + x) 或 log(x / (1 - x))。

  另外一种类似的做法是 square root 平方根或 cube root 立方根（可以用在负数）。

- **2.5.3 Binarization**

  对数值型的数据设定一个 threshold，大于就赋值为 1、小于就赋值为 0。例如 `score`，如果你只关心「及格」或「不及格」，可以直接把成绩对应到 1（`score >= 60`）和 0（`score < 60`）。或是你要做啤酒销量分析，你可以新增一个 `age >= 18` 的特征来标示出已成年。

  你有一个 `color` 的 categorical 特征，如果你不在乎实际上是什么颜色的话，其实也可以改成 `has_color`。

- **2.5.4 Binning**

  也称为 bucketization。以 `age` 这样的特征为例，你可以把所有年龄拆分成 n 段，0-20 岁、20-40 岁、40-60 岁等或是 0-18 岁、18-40 岁、40-70 岁等（等距或等量），然后把个别的年龄对应到某一段，假设 26 岁是对应到第二个 bucket，那新特征的值就是 2。这种方式是人为地指定每个 bucket 的边界值，还有另外一种拆分法是根据数据的分布来拆，称为 quantization 或 quantile binning，你只需要指定 bucket 的数量即可。

同样的概念应用到其他地方，可以把 datetime 特征拆分成上午、中午、下午和晚上；如果是 categorical 特征，则可以先 SELECT count() ... GROUP BY，然后把出现次数小于某个 threshold 的值改成 "Other" 之类的。或者是你有一个 occupation 特征，如果你其实不需要非常准确的职业资讯的话，可以把 "Web Developer"、"iOS Developer" 或 "DBA" 这些个别的资料都改成 "Software Engineer"。

binarization 和 binning 都是对 continuous 特征做 discretization 离散化，增强模型的非线性泛化能力。

- **2.5.5 Integer Encoding**

  也称为 label encoding。把每个 category 对应到数字，一种做法是随机对应到 0, 1, 2, 3, 4 等数字；另外一种做法是依照该值出现的频率大小的顺序来给值，例如最常出现的值给 0，依序给 1, 2, 3 等等。如果是针对一些在某种程度上有次序的 categorical 特征（称为 ordinal），例如「钻石会员」「白金会员」「黄金会员」「普通会员」，直接 mapping 成数字可能没什么问题，但是如果是类似 `color` 或 `city` 这样的没有明显大小的特征的话，还是用 one-hot encoding 比较合适。不过如果用的是 tree-based 的算法就无所谓了。

  有些 categorical 特征也可能会用数字表示（例如 id），跟 continuous 特征的差别是，数值的差异或大小对 categorical 特征来说没有太大的意义。

- **2.5.6 One-hot Encoding(OHE)**

  如果某个特征有 m 种值（例如 Taipei, Beijing, Tokyo），那它 one-hot encode 之后就会变成长度为 m 的向量：

  - Taipei: [1, 0 ,0]
  - Beijing: [0, 1, 0]
  - Tokyo: [0, 0, 1]

  你也可以改用 Dummy coding，這樣就只需要產生長度為 m -1 的向量：

  - Taipei: [1, 0]
  - Beijing: [0, 1]
  - Tokyo: [0, 0]

  OHE 的缺点是容易造成特征的维度大幅增加和没办法处理之前没见过的值。

- **2.5.7 Bin-counting**

  例如在 Computational Advertising 中，如果你有针对每个 user 的「广告曝光数（包含点击和未点击）」和「广告点击数」，你就可以算出每个 user 的「点击率」，然后用这个机率来表示每个 user，反之也可以对 ad id 使用类似的做法。

```
ad_id   ad_views  ad_clicks  ad_ctr
412533  18339     1355       0.074
423334  335       12         0.036
345664  1244      132        0.106
349833  35387     1244       0.035
```

  换个思路，如果你有一个 brand 的特征，然后你可以从 user 的购买记录中找出购买 A 品牌的人，有 70% 的人会购买 B 品牌、有 40% 的人会购买 C 品牌；购买 D 品牌的人，有 10% 的人会购买 A 品牌和 E 品牌，你可以每个品牌表示成这样：

```
brand  A    B    C    D    E
A      1.0  0.7  0.4  0.0  0.0
B      ...
C      ...
D      0.1  0.0  0.0  1.0  0.1
E      ...
```

- **2.5.8 LabelCount Encoding**

  类似 Bin-cunting 的做法，一样是利用现有的 count 或其他统计上的资料，差别在于 LabelCount Encoding 最后用的是次序而不是数值本身。优点是对异常值不敏感。

```
ad_id   ad_clicks  ad_rank
412533  1355       1
423334  12         4
345664  132        3
349833  1244       2
```

- **2.5.9 Count Vectorization**

  除了可以用在 text 特征之外，如果你有 comma-seperated 的 categorical 特征也可以使用这个方法。例如电影类型 `genre`，里头的值长这样 `Action,Sci-Fi,Drama`，就可以先用 `RegexTokenizer` 转成 `Array("action", "sci-fi", "drama")`，再用 `CountVectorizer` 转成 vector。

- **2.5.10 Feature Hashing**

  以 user id 为例，透过一个 hash function 把每一个 user id 映射到 `(hashed_1, hashed_2, ..., hashed_m)` 的某个值。指定 m << user id 的取值范围，所以缺点是会有 collision（如果你的 model 足够 robust，倒也是可以不管），优点是可以良好地处理之前没见过的值和罕见的值。当然不只可以 hash 单一值，也可以 hash 一个 vector。

  你可以把 feature hashing 表示为单一栏位的数值（例如 2）或是类似 one-hot encoding 那样的多栏位的 binary 表示法（例如 [0, 0, 1]）。

- **2.5.11 Category Embedding**

- **2.5.12 User Profile**

  使用用户画像来表示每个 user id，例如用户的年龄、性别、职业、收入、居住地、偏好的各种 tag 等，把每个 user 表示成一个 feature vector。除了单一维度的特征之外，也可以建立「用户听过的歌都是哪些曲风」、「用户（30 天内）浏览过的文章都是什么分类，以 TF-IDF 的方式表达。或者是把用户所有喜欢文章对应的向量的平均值作为此用户的 profile。比如某个用户经常关注与推荐系统有关的文章，那么他的 profile 中 "CB"、"CF" 和 "推荐系统" 对应的权重值就会较高。

- **2.5.13 Rare Categorical Varibales**

  先计算好每一种 category 的数量，然后把小于某个 threshold 的 category 都改成 "Others" 之类的值。或是使用 clustering 演算法来达到同样的目的。你也可以直接建立一个新的 binary feature 叫做 rare，要来标示那些相对少见的资料点。

- **2.5.14 Unseen Categorical Variables**

当你用 training set 的资料 fit 了一个 `StringIndexer`（和 `OneHotEncoder`），把它拿去用在 test set 上时，有一定的机率你会遇到某些 categorical 特征的值只在 test set 出现，所以对只见过 training set 的 transformer 来说，这些就是所谓的 unseen values。

对付 unseen values 通常有几种做法：

1. 用整个 training set + test set 来编码 categorical 特征
2. 直接舍弃含有 unseen values 的那条记录
3. 把 unseen values 改成 "Others" 之类的已知值。

如果采用第一种方式，一但你把这个 transformer 拿到 production 去用时，无可避免地还是会遇到 unseen values。不过通常线上的 feature engineering 会有别的做法，例如事先把 user 或 item 的各项特征都算好（定期更新或是 data 产生的时候触发），然后以 id 为 key 存进 Redis 之类的 NoSQL 里，model 要用的时候直接用 user id / item id 拿到处理好的 feature vector。

- **2.5.15 Large Categorical Variables**

  针对那种非常大的 categorical 特征（例如 id 类的特征），如果你用的是 logistic regression，其实可以硬上 one-hot encoding。不然就是利用上面提到的 feature hashing 或 bin counting 等方式；如果是 GBDT 的话，甚至可以直接用 id 硬上，只要 tree 足够多。

2.6 Feature Construction 特征构建

特征构建指的是从原有的特征中，人工地创造出新的特征，通常用来解决一般的线性模型没办法学到非线性特征的问题。其中一个重点是能否通过某些办法，在特征中加入某些「额外的资讯」，虽然也得小心数据偏见的问题。

如果你有很多 user 购物的资料，除了可以 aggregate 得到 total spend 这样的 feature 之外，也可以变换一下，变成 spend in last week、spend in last month 和 spend in last year 这种可以表示「趋势」的特征。

例如：

1. `author_avg_page_view`: 该文章作者的所有文章的平均浏览数
2. `user_visited_days_since_doc_published`: 该文章发布到该用户访问经过了多少天
3. `user_history_doc_sim_categories`: 用户读过的所有文章的分类和该篇文章的分类的 TF-IDF 的相似度
4. `user_history_doc_sim_topics`: 用户读过的所有文章的内文和该篇文章的内文的 TF-IDF 的相似度

- **2.6.1 Temporal Features 时间特征**

  对于 date / time 类型的资料，除了转换成 timestamp 和取出 day、month 和 year 做成新的栏位之外，也可以对 hour 做 binning（分成上午、中午、晚上之类的）或是对 day 做 binning（分成工作日、周末）；或是想办法查出该日期当天的天气、节日或活动等讯息，例如 `is_national_holiday` 或 `has_sport_events`。

  更进一步，用 datetime 类的资料通常也可以做成 `spend_hours_last_week` 或 `spend_money_last_week` 这种可以用来表示「趋势」的特征。

- **2.6.2 Text Features 文字特征**
- **2.6.3 Spatial Features 地理特征**
- **2.6.4 Cyclical Features**

2.7 Feature Interaction 特征交互

假设你有 `A` 和 `B` 两个 continuous 特征，你可以用 `A + B`、`A - B`、`A * B` 或 `A / B` 之类的方式建立新的特征。例如 `house_age_at_purchase = house_built_date - house_purchase_date` 或是 `click_through_rate = n_clicks / n_impressions`。

还有一种类似的作法叫 Polynomial Expansion 多项式展开，当 degree 为 2 时，可以把 `(x, y)` 两个特征变成 `(x, x * x, y, x * y, y * y)` 五个特征。

2.8 Feature Combination 特征组合

特征组合主要是针对 categorical 特征，特征交互则是适用于连续值特征。但是两者的概念是差不多的，就是把两个以上的特征透过某种方式结合在一起，变成新的特征。通常用来解决一般的线性模型没办法学到非线性特征的问题。

假设有 `gender` 和 `wealth` 两个特征，分别有 2 和 3 种取值，最简单的方式就是直接 string concatenation 组合出一个新的特征 `gender_wealth`，共有 2 x 3 = 6 种取值。因为是 categorical 特征，可以直接对 `gender_wealth` 使用 `StringIndexer` 和 `OneHotEncoder`。你当然也可以一起组合 continuous 和 categorical 特征，例如 `age_wealth` 这样的特征，只是 vector 里的值就不是 0 1 而是 age 本身了。

假设 C 是 categorical 特征，N 是 continuous 特征，以下有几种有意义的组合：

```
user_id  age  gender  wealth  gender_wealth  gender_wealth_ohe   age_wealth
1        56   male    rich    male_rich      [1, 0, 0, 0, 0, 0]  [56, 0, 0]
2        30   male    middle  male_middle    [0, 1, 0, 0, 0, 0]  [0, 30, 0]
3        19   female  rich    female_rich    [0, 0, 0, 1, 0, 0]  [19, 0, 0]
4        62   female  poor    female_poor    [0, 0, 0, 0, 0, 1]  [0, 0, 62]
5        78   male    poor    male_poor      [0, 0, 1, 0, 0, 0]  [0, 0, 78]
6        34   female  middle  female_middle  [0, 0, 0, 0, 1, 0]  [0, 34, 0]
```

2.9 Feature Extraction 特征提取

通常就是指 dimensionality reduction。

- **Principal Component Analysis (PCA)**
- **Latent Dirichlet Allocation (LDA)**
- **Latent Semantic Analysis (LSA)**

2.10 Feature Selection 特征选择

特征选择是指通过某些方法自动地从所有的特征中挑选出有用的特征。

- **Filter Method**

  采用某一种评估指标（发散性、相关性或 Information Gain 等），单独地衡量个别特征跟 target variable 之间的关系，常用的方法有 Chi Square Test（卡方检验）。这种特征选择方式没有任何模型的参与。

  以相关性来说，也不见得跟 target variable 的相关性越高就越好。

- **Wrapper Method**

会采用某个模型来预测你的 target variable，把特征选择想成是一个组合优化的问题，想办法找出一组特征子集能够让模型的评估结果最好。缺点是太耗时间了，实际上不常用。

- **Embedded Method**

  通常会采用一个会为特征赋予 coefficients 或 importances 的演算法，例如 Logistic Regression（特别是使用 L1 penalty）或 GBDT，直接用权重或重要性对所有特征排序，然后取前 n 个作为特征子集。

2.11 Feature Learning 特征学习

也称为 Representation Learning 或 Automated Feature Engineering。

- **GBDT**
- **Neural Network: Restricted Boltzmann Machines**
- **Deep Learning: Autoencoder**