Function Basis Encoding of Numerical Features in Factorization Machines

Anonymous authors
Paper under double-blind review

Abstract

Factorization machine (FM) variants are widely used for large scale real-time content recommendation systems, since they offer an excellent balance between model accuracy and low computational costs for training and inference. These systems are trained on tabular data with both numerical and categorical columns. Incorporating numerical columns poses a challenge, and they are typically incorporated using a scalar transformation or binning, which can be either learned or arbitrarily chosen. In this work, we provide a systematic and theoretically-justified way to incorporate numerical features into FM variants by encoding them into a vector of function values for a set of functions of one's choice.

We view FMs as approximators of *segmentized* functions, namely, functions from a field's value to the real numbers, assuming the remaining fields are assigned some given constants, which we refer to as the segment. From this perspective, we show that our technique yields a model that learns segmentized functions of the numerical feature spanned by the set of functions of one's choice, namely, the spanning coefficients vary between segments. Hence, to improve model accuracy we advocate the use of functions known to have powerful approximation capabilities, and offer the B-Spline basis due to its well-known approximation power, widespread availability in software libraries and its efficiency in terms of computational resources and memory usage. Our technique preserves fast training and inference, and requires only a small modification of the computational graph of an FM model. Therefore, incorporating it into an existing system to improve its performance is easy. Finally, we back our claims with a set of experiments that include a synthetic experiment, performance evaluation on several data-sets, and an A/B test on a real online advertising system which shows improved performance. We have made the code to reproduce the experiments available at `https://anonymous.4open.science/r/continuous_features-9639/`.

1 Introduction

Traditionally, online content recommendation systems rely on predictive models to choose a set of items to display by predicting the affinity of the user towards a set of candidate items. These models are usually trained on feedback gathered from a log of interactions between users and items from the recent past. For systems such as online ad recommenders with billions of daily interactions, speed is crucial. The training process must be fast to keep up with changing user preferences and quickly deploy a fresh model. Similarly, model inference, which amounts to computing a score for each item, must be rapid to select a few items to display out of a vast pool of candidate items, all within a few milliseconds. Factorization machine (FM) variants, such as Rendle (2010); Juan et al. (2016); Pan et al. (2018); Sun et al. (2021), are widely used in these systems due to their ability to train incrementally, and strike a good balance between being able to produce accurate predictions, while facilitating fast training and inference.

The training data consists of past interactions between users and items, and is typically given in tabular form, where the table's columns, or *fields*, have either categorical or numerical features. For example, “gender” or “time since last visit” are fields, whereas “Male” and “10 hours” are corresponding features. In recommendation systems that rely on FM variants, each row in the table is typically encoded as a concatenation of field encoding vectors. Categorical fields are usually one-hot encoded, whereas numerical fields are conventionally binned to a finite set of intervals to form a categorical field, and one-hot encoding is subsequently applied. A large number of works are devoted to the choice of intervals, e.g. Dougherty et al. (1995); Peng et al. (2009); Liu et al. (2002); Gama & Pinto (2006). Regardless of the choice, the model's output is a *step function* of the value of a given numerical field, assuming the remaining fields are kept constant, since the same interval is chosen independently of where the value falls in a given interval. For example, consider a model training on a data-set with "age", "device type", and "time the user spent on our site". For the segment of 25-years old users using an iPhone the model will learn some step function for different spending time values, whereas for the segment of 37-years old users using a laptop the model may learn a (possibly) different step function.

However, the optimal segmentized functions the model aims to learn, which describe the user behavior, aren't necessarily step functions. Typically, such functions are continuous or even smooth, and there is a gap between the approximating step functions the model learns, and the optimal ones. In theory, a potential solution is simply to increase the number of bins. This increases the approximation power of step functions, and given an infinite amount of data, would indeed help. However, the data is finite, and this can lead to a *sparsity* issue - as the number of learning samples assigned to each bin diminishes, it becomes increasingly challenging to learn a good representation of each bin, even with large data-sets, especially because we need to represent all segments simultaneously. This situation can lead to a degradation in the model's performance despite having increased the theoretical approximation power of the model. Therefore, there is a limit to the accuracy we can achieve with binning on a given data-set as demonstrated in Figure 1.

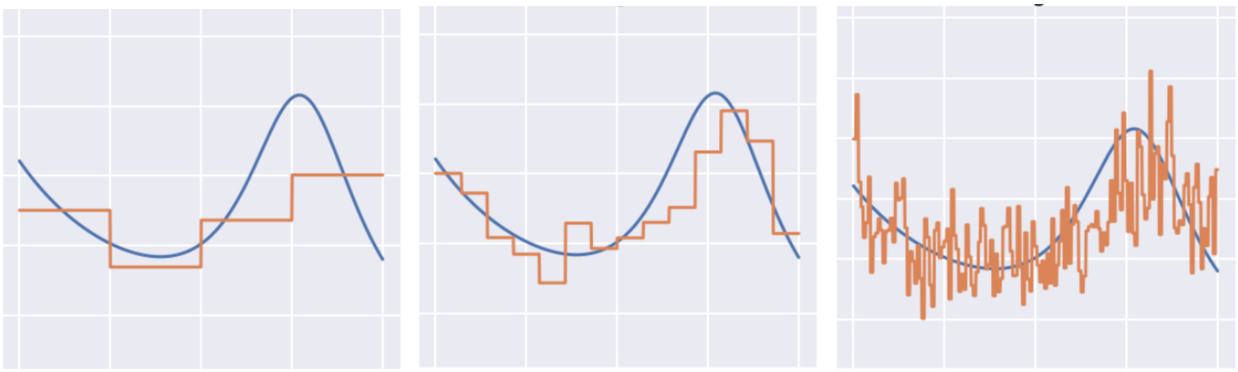


Figure 1: For a given segment, learned segmentized step function approximations (orange) of a true function (blue) which was used to generate a synthetic data-set. On the left - too few bins, "bad" approximation. In the middle - a balanced number of bins, "moderate" approximation. On the right - many bins, but approximation gets even "worse" due to a sparsity issue. Refer to Figure 4 for additional synthetic evaluation results with loss measurements.

In this work, we propose a technique to improve the accuracy of FM variants by reducing the approximation gap *without* sparsity issues, while preserving training and inference speeds. In other words, on the approximation-estimation balance, we aim to reduce both approximation and estimation error. Our technique is composed of encoding a numerical field using a vector of *basis functions*, and a minor modification to the computational graph of an FM variant. The idea of using nonlinear basis functions to model nonlinear functions is, of course a standard practice with linear models. This work is about analyzing the *interplay* between nonlinear basis functions and the FM family, to yield a surprisingly powerful technique for recommender systems, and tabular datasets in general.

Indeed, we present an elementary Lemma showing that the resulting model learns a *segmentized* output function spanned by the chosen basis, meaning that spanning coefficients depend on the values of the remaining fields. This is, of course, an essential property for recommendation systems, since indeed users with different demographic or contextual properties may behave differently. To the best of our knowledge, the idea of using arbitrary basis functions with FMs and the insights we present regarding the representation power of such a combination are new.

Based on the generic observation above, we offer the B-Spline basis (de Boor, 2001, pp 87) defined on the unit interval on uniformly spaced break-points, composed onto a transformation that maps the feature to the unit interval. The number of break-points (a.k.a knots) is a system hyper-parameter which can be further optimized. The strong approximation power of splines (de Boor, 2001, pp 149) ensures that we do not need a large number of break-points. Hence, we can closely approximate the optimal segmentized functions, without introducing sparsity issues. Moreover, to make integration of our idea easier in a practical production-grade recommendation system, we present a technique to seamlessly integrate a model trained using our proposed scheme into an existing recommendation system that employs binning, albeit with a controllable reduction in accuracy. Although a significant part of this work considers the B-Spline basis, the techniques we present can be used with an arbitrary basis.

To summarize, the main contributions of this work are: (a) *Basis encoding* We propose encoding numerical features using a vector of basis functions, while introducing a minor change to the computational graph of FM variants; (b) *Spanning properties* We show that using our method, any model from a family that includes many popular FM variants, learns a *segmentized* function spanned by the basis of our choice, and inherits the approximation power of that basis; (c) *B-Spline basis* We justify the use of the B-Spline basis from both theoretical and practical aspects, and demonstrate their benefits via numerical evaluation; (d) *Ease of integration into an existing system* We show how to integrate a model trained according to our method into an existing recommender system which currently employs numerical feature binning, to significantly reduce integration costs; (e) *Simplified numerical feature engineering* the strong approximation power of the cubic B-Spline basis allows building accurate models without investing time and effort to manually tune bin boundaries, and use simple uniform break-points instead.

1.1 Related work

Putting aside the FM variants, there is a large body of work dealing with neural networks training over tabular data Arik & Pfister (2021); Badirli et al. (2020); Gorishniy et al. (2021); Huang et al. (2020); Popov et al. (2020); Somepalli et al. (2022); Song et al. (2019); Hollmann et al. (2022). Neural networks have the potential to achieve high accuracy and can be incrementally trained on newly arriving data using transfer learning techniques. Additionally, due to the *universal approximation* theorem Hornik et al. (1989), neural networks are capable of representing a segmentized function from any numerical feature to a real number. However, the time required to train and make inferences using neural networks is significantly greater than that required for factorization machines. Even though some work has been done to alleviate this gap for neural networks by using various embedding techniques Gorishniy et al. (2022), they have not been able to outperform other model types. As a result, in various practical applications, FMs are preferred over NNs. For this reason, in this work we focus on FMs, and specifically on decreasing the gap between the representation power of FMs and NNs without introducing a significant computational and conceptual complexity.

A very simple but related approach to ours was presented in Covington et al. (2016). The work uses neural networks and represents a numerical value z as the triplet $(z, z^2, \sqrt{z})$ in the input layer, which can be seen as a variant of our approach using a basis of three functions. Another approach which bears similarity to ours also comes from works which use deep learning Cheng (2022); Gorishniy et al. (2021); Song et al. (2019); Guo et al. (2021). In these works, first-order splines are used in the input layer to represent continuity, and the representation power of a neural network compensates for the weak approximation power of first-order splines. Here we do the opposite - we use the stronger approximation power of cubic splines to compensate for the weaker representation power of FMs.

The recent work of David (2022) uses a technique which appears to resemble ours, but takes the perspective of function approximation on a bounded domain, applies to regular FM models only, and expands towards higher orders of interaction. In contrast, our work takes the perspective of recommender systems and has a significantly broader theoretical and practical scope: it applies to a wide range of FM variants, includes handling of unbounded domains and interaction with categorical features, and reports A/B test results on a real-world ad recommendation system.

Finally, any comprehensive discussion on tabular data would be incomplete without mentioning *gradient boosted decision trees* (GBDT) Chen & Guestrin (2016); Ke et al. (2017); Prokhorenkova et al. (2018), which are known to achieve state-of-the-art results Gorishniy et al. (2021); Shwartz-Ziv & Armon (2022). However, GBDT models aren't useful in a variety of practical applications, primarily due to significantly slower inference speeds,namely, it is challenging and costly to use GBDT models to rank hundreds of thousands of items in a matter of milliseconds.

2 Formal problem statement

We consider tabular data-sets with m fields, $f_1, \dots, f_m$, each can be either numerical or categorical field. We denote a row in the dataset with $(z_1, \dots, z_m)$, each z_i is a feature (value) associated with its corresponding field f_i. Each feature z_i is mapped to a vector of values by applying its corresponding encoding function enc_i:

$$\boldsymbol{x} = \mathbf{enc}(z_1, \dots z_m) \equiv \begin{bmatrix} \mathbf{enc}_1(z_1) \\ \vdots \\ \mathbf{enc}_m(z_m) \end{bmatrix},$$

Each $\mathbf{enc}_i$ is assumed to encode the corresponding feature into a vector. For example, enc_i be a one-hot encoding of a categorical field f_i. The vector $\boldsymbol{x}$ is then fed into a model. The indices in $\boldsymbol{x}$ holding the encoded vector of a given field f are denoted by $\mathcal{I}(f)$, and the field whose value is mapped to x_i is denoted by $f(i)$. With a slight abuse of notation, we will denote by ϕ_f the model's segmentized output, which is the output as a function of the value of a field f, assuming the remaining fields are some given constants. As in any supervised learning task, ϕ_f aims to approximate some unknown optimal segmentized function.

In the vast majority of cases, a numerical field f is either encoded using a scalar transform $\mathbf{enc}_f : \mathbb{R} \to \mathbb{R}$, or using binning - the numerical domain is partitioned into a finite set of intervals $\mathbf{enc}_f(z_f)$ is a one-hot encoding of the interval z_f belongs to. Since for any z_f in a given binning interval we have the same encoding, a deterministic model will produce the same output for the entire interval, and therefore ϕ_f is a step function.

Step functions are considered weak approximators in the sense that many binning intervals are required to achieve good approximation accuracy. For example, Lipschitz-continuous functions can be approximated by a step function on ℓ intervals up to an error of only $O(\frac{1}{\ell})$ (de Boor, 2001, pp 149). Hence, with binning we need to strike an intricate balance between the theoretically-achievable approximation accuracy and our ability to achieve it because of sparsity issues, as illustrated in Figure 1. Note, that the above observation does not depend on whether the bins are chosen a-priori or learned. In this work we propose an alternative to binning, by encoding numerical features in a way that allows achieving more of the theoretical accuracy before sparsity issues take effect.

2.1 The factorization machine family

In this work we consider several model variants, which we refer to as the factorization machine family. The family includes the celebrated *factorization machine* (FM) Rendle (2010), the *field-aware factorization machine* (FFM) Juan et al. (2016), the *field-weighted factorization machine* (FwFM) Pan et al. (2018), and the *field-matrixed factorization machine* (FmFM) Sun et al. (2021); Pande (2021), that generalized the former variants.

As a convention, we denote by $f(i)$ the field whose value is used to encode the i^{th} component of the feature vector $\boldsymbol{x}$, and denote arbitrary fields by f, e. We also denote by $\langle \boldsymbol{x}, \boldsymbol{y} \rangle_{\boldsymbol{P}} = \boldsymbol{x}^T \boldsymbol{P} \boldsymbol{y}$ the "inner product"[1] associated with some matrix $\boldsymbol{P}$. The FmFM model computes

$$\phi_{\text{FmFM}}(\boldsymbol{x}) = w_0 + \langle \boldsymbol{x}, \boldsymbol{w} \rangle + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle x_i \boldsymbol{v}_i, x_j \boldsymbol{v}_j \rangle_{\boldsymbol{M}_{f(i),f(j)}}, \tag{1}$$

where $w_0 \in \mathbb{R}$, $\boldsymbol{w} \in \mathbb{R}^n$, and $\boldsymbol{v}_i \in \mathbb{R}^{k_i}$ are learned parameters. The *field-interaction matrices* $\boldsymbol{M}_{f(i),f(j)} \in \mathbb{R}^{k_{f(i)} \times k_{f(j)}}$ can be either learned or predefined, and may have a special structure. FMs are typically employed

[1] FmFMs do not require it to be a real inner product. For it to be a true inner product, the matrix has to be square and positive definite

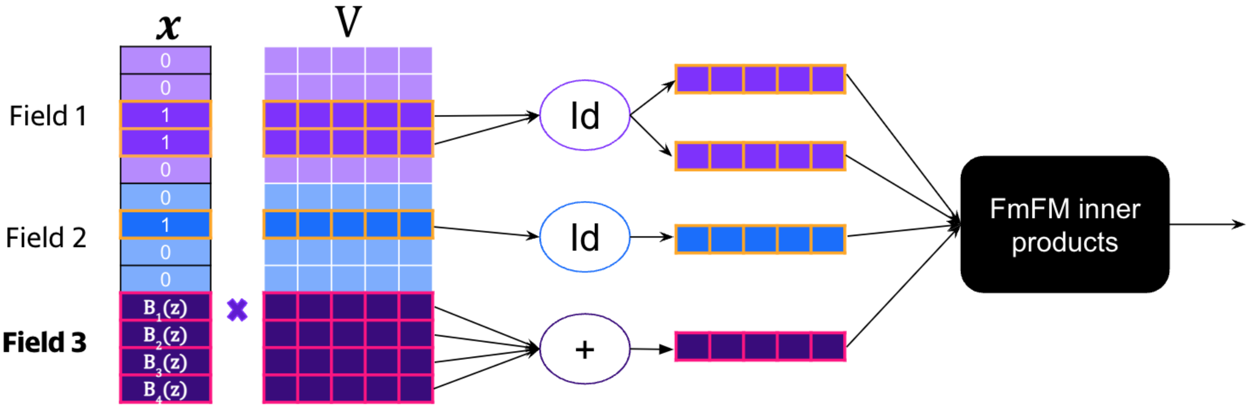


Figure 2: The computational graph with continuous numerical fields. Field 3 is a continuous numerical field whose value is z. The vectors $\boldsymbol{v}_1, \ldots, \boldsymbol{v}_n$ in the rows of $\boldsymbol{v}$ are multiplied by the input vector $\boldsymbol{x}$. Then, a reduction is applied to each field. Most fields have the identity (Id) reduction, whose output is identical to its input, whereas field 3 uses the sum reduction. The resulting vectors are then passed to the pairwise interaction module. An analogous process happens with the $\boldsymbol{w}$ vector.

for supervised learning tasks on a dataset $\{(\boldsymbol{x}_t, y_t)\}_{t=1}^N$ with labels y_t . Note that this model family allows a different embedding dimension for each field, since the matrices $\boldsymbol{M}_{f(i),f(j)}$ do not have to be square. For completeness, we present the above FM variants in Appendix B, and explain why FmFMs generalize the entire family. Given this property , we use ϕ_{FmFM} in equation 1 to describe and prove properties which should hold for the entire family.

3 The basis function encoding approach

To describe our approach we need to introduce a slight modification to the FmFM computational graph formulated in equation 1. Before feeding the vectors $x_i \boldsymbol{v}_i$ and scalars $x_i w_i$ into the FM variant of choice, we pass the vectors and, respectively, scalars belonging to each field through a linear reduction associated with field f_i. We consider two reductions: (a) the identity reduction, which just returns its input; and (b) a sum reduction, which computes $\sum_{i \in \mathcal{I}(f_i)} x_i \boldsymbol{v}_i$ and, respectively, $\sum_{i \in \mathcal{I}(f_i)} x_i w_i$. Clearly, choosing the identity function for all fields reduces back to the regular FmFM model.

For a *continuous numerical* field f , we choose a set of functions $B_1^f, \ldots, B_{\ell_f}^f$, and encode the field as $\mathbf{enc}_f(z) = (B_1^f(z), \ldots, B_{\ell_f}^f(z))^T$, followed by a subsequent summing reduction . When the field f under consideration if clear from the context, we omit it for clarity, and write $B_1, \ldots, B_\ell$. After the summing reduction, the reduced vector of field f having the value z is $\sum_{i \in \mathcal{I}(f)} B_i(z) \boldsymbol{v}_i$, and the reduced scalar is $\sum_{i \in \mathcal{I}(f)} B_i(z) w_i$. For the remaining fields, we use the identity reduction. The process to calculate the reduced vectors before feeding them into the FM model is depicted in Figure 2. Note that each field can have its own set of basis functions, and in particular, the size of the bases may differ . For a formal description using matrix notation we refer the readers to Appendix A.1.

3.1 Spanning properties

To show why our modeling choices improve the approximation power of the model, we prove two technical Lemmas that establish a relationship between the basis of choice and the model's output.

Lemma 1 (Spanning property). *Let*

$$\mathbf{enc}_f(z) = (B_1(z), \cdots, B_\ell(z))^T$$

be the encoding associated with a continuous numerical field f*, and suppose* ϕ_{FmFM} *is computed according to equation 1. Then for every assignment of the remaining fields, there exist* $\alpha_1, \ldots, \alpha_\ell, \beta \in \mathbb{R}$*, which depend on* *the values of the remaining fields and the model's parameters but* not *on z, such that ϕ_{FmFM} as a function of z can be written as*

$$\phi_{\text{FmFM}}(z) = \sum_{i=1}^{\ell} \alpha_i B_i(z) + \beta.$$

An elementary formal proof can be found in Appendix A.2, but to provide insight, we present an informal explanation. The vector stemming from a numerical field, after the summing reduction, is a linear combination of the basis functions, while the remaining post-reduction vectors do no depend on z. Thus, when pairwise inner products are computed, we obtain a linear combination of the basis functions plus some scalar β.

Another interesting result can be obtained by looking at ϕ_{FmFM} as a function of *two* continuous numerical fields. It turns out that we obtain a function in the span of a tensor product of the two bases chosen for the two fields.

Lemma 2 (Pairwise spanning property)**.** *Let e, f be two continuous numerical fields. Suppose that $\mathbf{enc}_e(z_e) = (B_1(z_e), \cdots B_\ell(z_e))^T$ and $\mathbf{enc}_f(z_f) = (C_1(z_f), \cdots C_\kappa(z_f))^T$ be field encoding functions, and suppose ϕ_{FmFM} is computed according to equation 1. Define $B_0(z) = C_0(z) = 1$. Then for every assignment of the remaining fields, there exist $\alpha_{i,j}, \beta \in \mathbb{R}$ for $i \in \{0, \ldots, \ell\}$ and $j \in \{0, \ldots, \kappa\}$, which depend on the values of the remaining fields and the model's parameters but* not *on z_e, z_f, such that ϕ_{FmFM} as a function of z_e, z_f can be written as*

$$\phi_{\text{FmFM}}(z_e, z_f) = \sum_{i=0}^{\ell} \sum_{j=0}^{\kappa} \alpha_{i,j} B_i(z_e) C_j(z_f) + \beta.$$

The proof can be found in Appendix A.3. Note, that the model learns $O(\ell \cdot \kappa)$ segmentized coefficients *without* actually learning $O(\ell \cdot \kappa)$ parameters, but instead it learns only $O(\ell + \kappa)$ parameters in the form of $\ell + \kappa$ embedding vectors.

Essentially, the model learns a *segmentized*, or a *personalized* approximation of an optimal user behavior function for each continuous numerical field in the affine span of the chosen basis, or the tensor product basis in case of two fields. So it is natural to ask ourselves - which basis should we choose? Ideally, we should aim to choose a basis that is able to approximate functions well with a small number of basis elements, to keep training and inference efficient, and avoid over-fitting.

3.2 Splines and the B-Spline basis

Spline functions Schoenberg (1946) are piece-wise polynomial functions of degree d with up to $d-1$ continuous derivatives defined on some interval $[a, b]$. The interval is divided into disjoint sub-intervals at a set of break-points $a = t_0 < t_1 < \cdots < t_{\ell-d} = b$, where each polynomial piece is defined on $[t_j, t_{j+1}]$. It is well-known that spline functions of degree d defined on $\ell - d + 1$ break-points can be written as weighted sums of the celebrated B-Spline basis (de Boor, 2001, pp 87) comprising of exactly ℓ functions. For brevity, we will not elaborate their explicit formula in this paper, and point out that it's available in a variety of standard scientific computing packages, e.g., the `scipy.interpolate.BSpline` class of the SciPy package Virtanen et al. (2020).

The cases of $d = 2$ and $d = 3$ are known as *quadratic* and *cubic splines*, respectively. Here we concentrate on splines with uniformly spaced break points, and at this stage assume that the values of our numerical field lie in a compact interval $[a, b]$, which we assume w.l.o.g is $[0, 1]$. . We discuss the more generic cases in the following sub-section.

It is known (de Boor, 2001, pp 149), that splines of degree d can approximate an arbitrary function g with $k \leq 1 + d$ continuous derivatives up to an error bounded by $O(\|g^{(k)}\|_\infty / \ell^k)^2$, where $g^{(k)}$ is the k^{th} derivative of g. The spanning property (Lemma 1) ensures that the model's segmentized outputs are splines spanned by the same basis, and therefore are powerful enough to approximate the optimal segmentized outputs. Therefore, assuming that the functions we aim to approximate are smooth enough and vary "slowly", in the

[2] For a function ϕ defined on S, its infinity norm is $\|\phi\|_\infty = \max_{x \in S} |\phi(x)|$

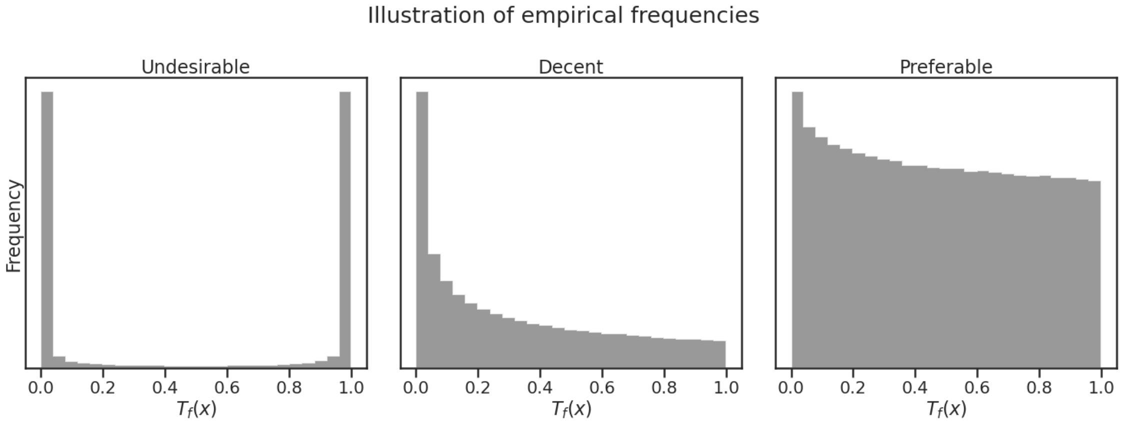


Figure 3: Examples of empirical frequencies of a field f under the transformation T_f. Left - an undesirable histogram, since the mid-range has almost no data. In the middle - decent, since there is a reasonable amount of data throughout the interval. Right - preferable, since the distribution is close to uniform.

sense that their high-order k^{th} derivatives are small, the approximation error goes down at the rate of $O(\frac{1}{\ell^k})$, whereas with binning the rate is $O(\frac{1}{\ell})$.

A direct consequence is that we can obtain a theoretically good approximation which is also achievable in practice, since we can be accurate with a small number of basis functions, and this significantly decreases the chances of sparsity and over-fitting issues. This is in contrast to binning, where high-resolution binning is required to for a good theoretical approximation accuracy, but it may not be achievable in practice.

Yet another important property of the B-Spline basis of degree d is that at any point only $1+d$ basis functions are non-zero. Thus, regardless of the number of basis functions ℓ we use, computing the model's output remains efficient, since the reduction described in Figure 2 requires a weighted sum of only $1+d$ vectors, regardless of the size of the basis.

3.3 Continuous numerical fields with arbitrary domain and distribution

Splines approximate functions on a compact interval. Thus, numerical fields with unbounded domains pose a challenge. Moreover, the support of each B-Spline function is only a sub-interval of the domain defined by $1+d$ consecutive knots. Thus, even if a numerical field f is bounded in $[a,b]$, a highly skewed distribution may cause "starvation" of the support of some basis functions: if $\mathbb{P}(z_f \in \text{support}(B_i))$ is extremely small, there will be little training data to effectively learn a useful representation of B_i.

As a remedy to both challenges, we recommend first transforming a numerical field f using a function $T_f : \mathbb{R} \to [0,1]$. The desired property of such a transformation is making sure that $\mathbb{P}(z_f \in \text{support}(B_i))$ is non-negligible for every i, namely, the data is spread on the interval such that no region of the interval is "starved". This is illustrated in Figure 3. In fact, this is the *feature engineering* part of our method.

In theory, if we knew the distribution of z_f, we could use its cumulative distribution function (CDF) as the transform of choice, since the transformed values would be uniformly distributed in $[0,1]$ David & Johnson (1948). In practice, T_f can be any function which roughly resembles the empirical CDF of the field's values, as described next. We recommend fitting a distribution with a simple closed-form CDF, such as Normal or Student-T, to a sub-sample of the field's values, and using its CDF as T_f. Alternatively, we can use the empirical CDF represented using a high-resolution step function, such as the one provided by the `QuantileTransform` class in the Scikit-Learn package Buitinck et al. (2013).

Note, that our method does not eliminate the need for data analysis and feature engineering, but only streamlines it; just fit a few distributions[3], and see (even visually) whose CDF *roughly* resembles the empirical

[3] Easy to do using `scipy.stats.{some_dist}.fit()`

CDF. The rest of the "heavy lifting" is done by the strong approximation power splines, and the computational efficiency facilitated by the B-Spline basis.

3.4 Integration into an existing system by simulating binning

Suppose we would like to obtain a model which employs binning of the field f into a large number N of intervals, e.g. $N = 1000$. As we discussed in the introduction, in most cases we cannot directly learn such a model because of sparsity issues. However, we can *generate* such a model from another model trained using our scheme to make initial integration easier.

The idea is best explained by referring, again, to Figure 2. For any value z of the numerical field of our choice, the vector corresponding to that field after the reduction stage (field 3 in the figure) is the weighted sum $\boldsymbol{v}_f(z) = \sum_{i \in \mathcal{I}(f)} B_i^f(z) \boldsymbol{v}_i$. Given a set of N "bins"$([z_{j+1}, z_j))_{j=0}^N$ that we would like to simulate, we simply compute N corresponding embedding vectors by evaluating $\boldsymbol{v}_f$ at the mid-point of each interval, as in

$$\boldsymbol{v}_f(\tfrac{z_{j+1}+z_j}{2}) \quad j = 0, \ldots, N$$

The resulting model now has an embedding vector for each bin, as we desire.

The choice of the set of bins may vary between fields. For example, in many practical situations a geometric progression of bin boundaries is applicable to fields with unbounded domains. Another natural choice is utilizing the transformation T_f from the previous section, that resembles the data CDF, and using its inverse computed at uniformly spaced points as the bin boundaries, i.e., $\{z_j = T_f^{-1}(\frac{j}{N})\}_{j=0}^N$.

4 Evaluation

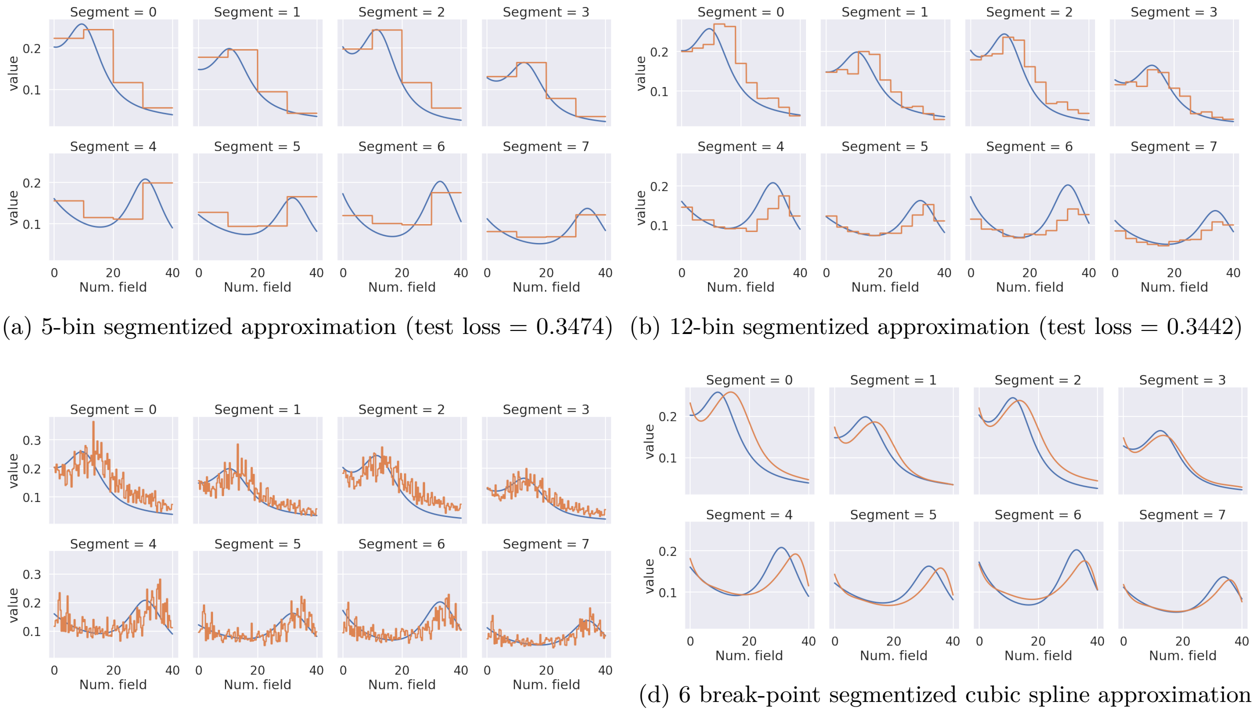
We divide this section into three parts. First, we use a synthetically generated data-set to show that our theory holds - the model learns segmentized output functions that resemble the ground truth. Then, we compare the accuracy obtained with binning versus splines on several data-sets. The code to reproduce these experiments is available in the supplemental material. Finally, we report the results of a web-scale A/B test conducted on a major online advertising platform serving real traffic.

4.1 Learning artificially chosen functions

We used a synthetic toy click-through rate prediction data-set with four fields, and zero-one labels (click / non-click). Naturally, the cross-entropy loss is used to train models on such tasks. We have three categorical fields each having two values each, and one numerical field in the range $[0, 40]$. For each of the eight segment configurations defined by the categorical fields, we defined functions $p_0, \ldots, p_7$ (see Figure 4) describing the CTR as a function of the numerical field. Then, we generated a data-set of 25,000 rows, such that for each row i we chose a segment configuration $s_i \in \{0, ..., 7\}$ of the categorical fields uniformly at random, the value of the numerical field $z_i \sim \text{Beta-Binomial}(40, 0.9, 1.2)$, and a label $y_i \sim \text{Bernoulli}(p_{s_i}(z_i))$.

We trained an FFM Juan et al. (2016) provided by Yahoo-Inc (2023) using the binary cross-entropy loss on the above data, both with binning of several resolutions and with splines defined on 6 sub-intervals . The numerical field was naïvely transformed to $[0, 1]$ by simple normalization. We plotted the learned curve for every configuration in Figure 4. Indeed, low-resolution binning approximates poorly, a higher resolution approximates better, and a too-high resolution cannot be learned because of sparsity. However, Splines defined on only six sub-intervals approximate the synthetic functions $\{p_i\}_{i=0}^7$ quite well.

Next, we compared the test cross-entropy loss on 75,000 samples generated in the same manner with for several numbers of intervals used for binning and cubic Splines. For each number of intervals we performed 15 experiments to neutralize the effect of random model initialization. As is apparent in Figure 5, Splines consistently outperform in this theoretical setting. The test loss obtained by both strategies increases if the number of intervals becomes too large, but the effect is much more significant in the binning solution.



(a) 5-bin segmentized approximation (test loss = 0.3474)

(b) 12-bin segmentized approximation (test loss = 0.3442)

(c) 120-bin segmentized approximation (test loss = 0.3478)

(d) 6 break-point segmentized cubic spline approximation (0.3432)

Figure 4: Results of segmentized approximations of four FFM models trained on synthetic data. In each plot, a family of segmentized functions on the interval $[0, 40]$, plotted in blue, are approximated by the model, plotted in orange. 12 bins are more accurate than 5, but 120 bins are even less accurate than 5 due to sparsity. With splines we achieve best accuracy.

4.2 Public tabular data-sets

Since our approach works on any tabular dataset, and isn't specific to recommender systems, we mainly test our approach versus binning on several tabular data-sets with abundant numerical features that have a strong predictive power: the California housing Pace & Barry (1997) , adult income Kohavi (1996), Higgs Baldi et al. (2014) (we use the 98K version from OpenML Vanschoren et al. (2014)), and song year prediction Bertin-Mahieux et al. (2011). For the first two data-sets we used an FFM, whereas for the last two we used an FM, both provided by Yahoo Yahoo-Inc (2023), since FFMs are significantly more expensive to train when there are many columns, and even more so with hyper-parameter tuning.

We use Optuna Akiba et al. (2019) for hyper-parameter tuning, namely, step-size, batch-size, number of intervals, and embedding dimension, separately for each strategy. For binning, we also tuned the choice of uniform or quantile bins. In addition, 20% of the data was held out for validation, and regression targets were standardized. Finally, for the adult income data-set, 0 has a special meaning for two columns, and was treated as a categorical value.

We ran 20 experiments with the tuned configurations to neutralize the effect of random initialization, and report the mean and standard deviation of the metrics on the test set in Table 1, where it is apparent that our approach outperforms binning on these datasets. These datasets were chosen since they contain several numerical fields, and are small enough to run many experiments to neutralize the effect of hyper-parameter choice and random initialization at a reasonable computational cost, or time. They were also used in other works on tabular data, such as Gorishniy et al. (2021; 2022).

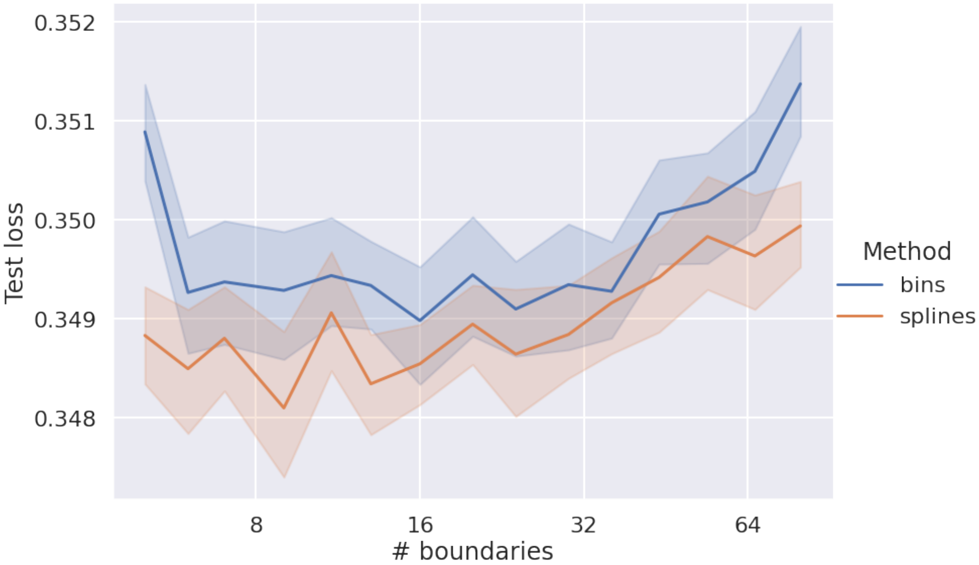


Figure 5: Comparison of the test cross-entropy loss obtained with Splines and bins. Both methods suffer from sparsity issues as the number of intervals grows, but Splines are able to utilize their approximation power with a small number of intervals, before sparsity takes effect. The bands are 90% bootstrap confidence intervals based on multiple experiment repetitions: for each number of boundaries and numerical field type we ran 15 experiments to neutralize the effect of random initialization.

Table 1: Comparison of binning vs. splines. Standard deviations are reported as in parentheses as % of the mean.

	Cal. housing	Adult income	Higgs	Song year
# rows	20640	48842	98050	515345
# cat. fields	0	8	0	0
# num. fields	8	6	28	88
Metric type	RMSE	Cross-Entropy	Cross-Entropy	RMSE
Binning metric (std%)	0.4730 (0.36%)	0.2990 (0.47%)	0.5637 (0.22%)	0.9186 (0.4%)
Splines metric (std%)	**0.4294** (0.57%)	**0.2861** (0.28%)	**0.5448** (0.12%)	**0.8803** (0.2%)
Splines vs. binning lift	9.2%	4.3%	3.36%	4.16%

4.3 The Criteo dataset

To further demonstrate the benefits of our approach over on a real-world recommendation dataset, we evaluate it with an FwFM on the Criteo display advertising challenge dataset cri (2014). It has plenty of numerical features, but these features are *integers*. Before presenting the experiments, it's important to discuss the unique challenges posed by integer features, and especially features representing counts, e.g. the number of visits of the user in the last week.

Integers possess properties of both discrete and continuous nature. For example, their CDF function is a *step* function that may have large jumps, and it's not trivial to transform them to $[0, 1]$ using a CDF approximation in a reasonable manner. This is because large jumps produce large gaps in the $[0, 1]$ interval that are not covered by any data. Moreover, features that represent counts, such as the number of times the user interacted with some category of items, pose an additional difficulty stemming from this hybrid discrete-continuous nature. Smaller values are more 'discrete', while larger values are more 'continuous', i.e. the difference between users who never visited our site and users who visited it once may be large, the

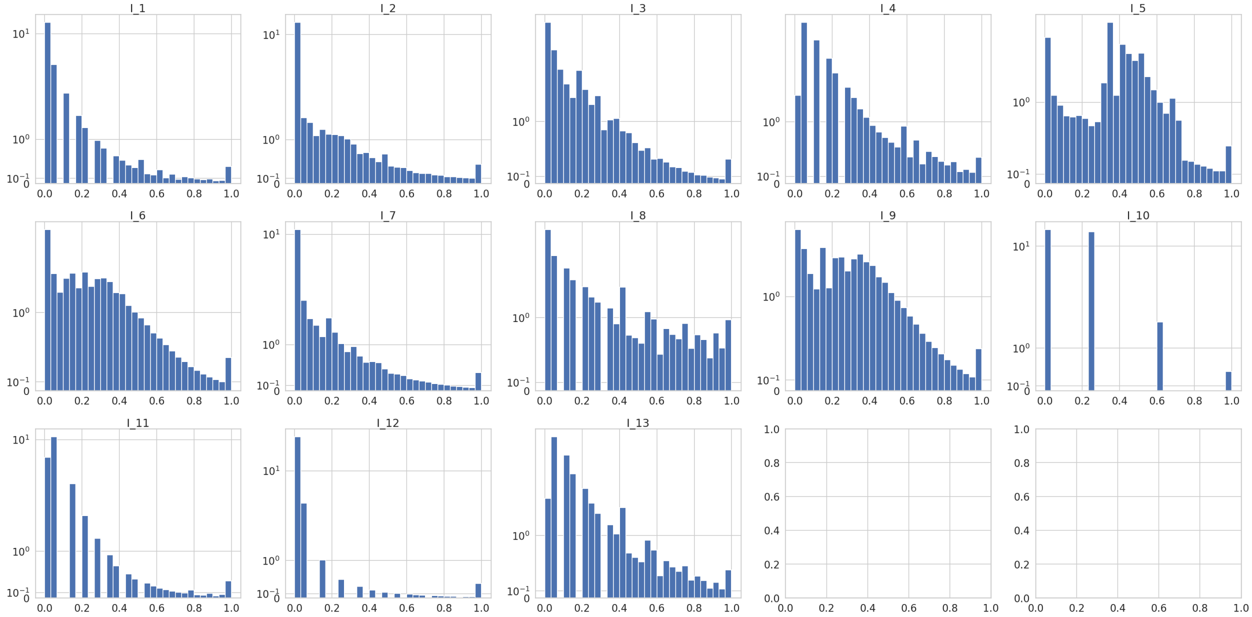


Figure 6: Histograms of columns `I_1`, ..., `I_13` transformed by $T_f(x)$ defined by the squared inverse hyperbolic sine, followed by min-max scaling.

difference between one and two visits will be large as well, but the difference in user behavior between 98 and 99 visits is probably small.

Despite these challenges, a work on algorithms for recommender systems cannot go without a benchmark on a well-known recommendation dataset, and the Criteo dataset was chosen due to its popularity and the abundance of numerical fields, even though they are integers. We point out that other classical public recommendation datasets, such as MovieLens GorupLens, or Avazu ava (2014), do not contain continuous or similar integer numerical fields, and therefore were incompatible for the analysis of this work[4].

The data-set is a log of 7 days of ad impressions and clicks in chronological order. Thus, we split into training, validation, and test in the following manner: the first 5/7 of the data-set is the training set, the next 1/7 is the validation set for hyper-parameter tuning, and the last 1/7 is the test set. Categorical features with less than 10 occurrences were replaced by a special "INFREQUENT" value for every field . When employing binning, we use a similar strategy to the winners of the Criteo challenge - the bin index is floor($\ln^2(z)$) for $z \geq 1$. Namely, the bin boundaries are $\{\exp(\sqrt{i})\}_{i=0}^{\ell}$ for $z \geq 1$. Values smaller than 1 are treated as categorical values.

For splines, we stand on the shoulders of giants, and use a similar transform to the winners in order to transform a field into $[0, 1]$: $x \to \operatorname{arcsinh}^2(x)$, followed min-max scaling. The reason is that $\operatorname{arcsinh}(x)$ mimics the logarithm, but is also defined for zero. The min-max scaling is learned from the training set only, and if the validation or the test set contain values above the maximum observed, they are mapped to 1. Negative values are, similarly, treated as categorical values. We note, that this transform is far from being the empirical CDF, and due to the reasons discussed above, the empirical CDF is typically not applicable to integers. The histograms of the transformed columns `I_1`, ..., `I_13` are plotted in Figure 6. We can see that some of the columns appear to be quite discrete. For example, `I_10` has only *four* distinct values. The columns `I_11` and `I_12` appear discrete as well. Thus, when conducting an experiment with cubic splines, we used them for all columns, except for the above three.

We conduct experiments with $k \in 8, 16, \dots, 64$ as embedding dimensions, and each experiment is conducted using 50 trials of Optuna Akiba et al. (2019) with its default configuration to tune the learning rate and

[4] We could, in theory, engineer such features using the well-known Target Encoding or Count Encoding techniques, but this is out of the scope of this work.

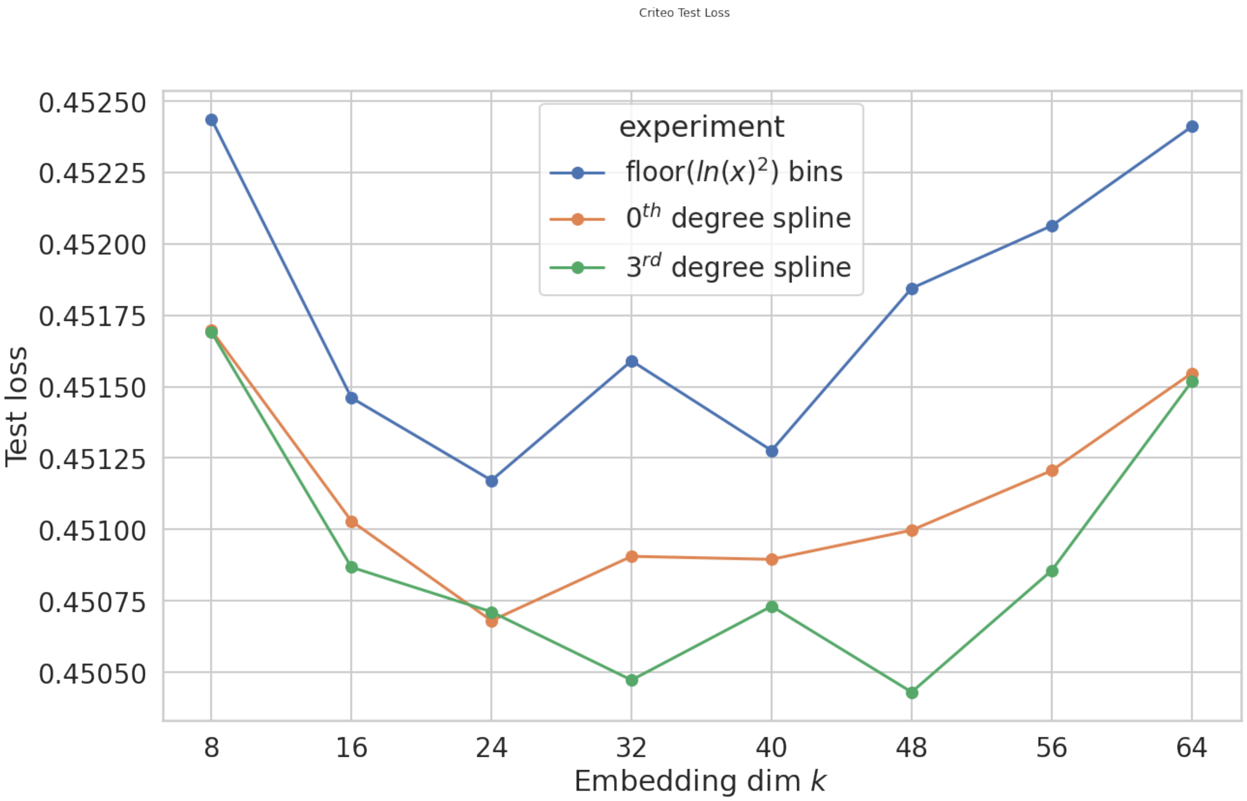


Figure 7: Test LogLoss for the Criteo experiment for various embedding dimensions.

the L_2 regularization coefficient. The models were trained using the AdamW optimizer Loshchilov & Hutter (2019) optimizer. As an ablation study, to make sure that cubic splines contribute to the the improvement we observe, we also conduct experiments with 0^{th} order splines applied after the above transformation, since it may be the case that the transformation itself yields an improvement. We remind the readers that 0^{th} order splines are just uniform bins. For splines, we used 20 knots, which is roughly half the number of bins obtained by the strategy employed by the Criteo winners. The obtained test losses are summarized in Figure 7. It is apparent that cubic splines outperform both uniform binning (0^{th} order splines) and the original binning procedure of the Criteo winners for most embedding dimensions. Moreover, we can see that cubic splines perform best when the embedding dimension is slightly larger than the best one for binning. We conjecture that, at least for the Criteo data-set, cubic splines require more expressive power yielded by a slightly higher embedding dimension to show their full potential.

4.4 A/B test results on an online advertising system

Here we report an online performance improvement measured using an A/B test, serving real traffic of a major online advertising platform. The platform applies a proprietary CTR prediction model that is closely related to FwFM Sun et al. (2021). The model, which provides CTR predictions for merchant catalogue-based ads, has a *recency* field that measures the time (in hours) passed since the user viewed a product at the advertiser's site. We compared an implementation using our approach of continuous feature training and high-resolution binning during serving time described in Section 3.4 with a fine grained geometric progression of 200 bin break points, versus the "conventional" binned training and serving approach used in the production model at that time. The new model is only one of the rankers[5] that participates in our ad auction. Therefore, a mis-prediction means the ad either unjustifiably wins or loses the auction, both leading to revenue losses.

[5] Usually each auction is conducted among several models (or "rankers"), that rank their ad inventories and compete over the incoming impression.

We conducted an A/B test against the production model at that time, when our new model was serving 40% of the traffic for over six days. The new model dramatically reduced the CTR prediction error, measured as $(\frac{\text{Average predicted CTR}}{\text{Measured CTR}} - 1)$ on an hourly basis, from an average of 21% in the baseline model, to an average of 8% in the new model. The significant increase in accuracy has resulted in this model being adopted as the new production model.

5 Discussion

We presented an easy to implement approach for improving the accuracy of the factorization machine family whose input includes numerical fields. Our scheme avoids increasing the number of model parameters and over-fitting, by relying on the approximation power of *why?* splines. This is explained by the spanning property along with the spline approximation theorems. Moreover, the discretization strategy described in Section 3.4 allows our idea to be integrated into an existing recommendation system without introducing major changes to the production code that utilizes the model to rank items, i.e., inference. .

It is easy to verify that our idea can be extended to factorization machine models of higher order Blondel et al. (2016). In particular, the spanning property in Lemma 1 still holds, and the pairwise spanning property in Lemma 2 becomes q-wise spanning property from machines of order q. However, to keep the paper focused and readable, we keep the analysis out of the scope of this paper.

With many advantages, our approach is not without limitations. We do not eliminate the need for data research and feature engineering, which is often required when working with tabular data, since the data still needs to be analyzed to fit a function that roughly resembles the empirical CDF. We believe that feature engineering becomes easier and more systematic, but some work still has to be done.

Finally, we would like to note two drawbacks. First, our approach slightly reduces interpretability, since we cannot associate a feature with a corresponding learned latent vector. Second, our approach may not be applicable to all kinds of numerical fields. For example, consider a product recommendation system with a product price field. Usually higher prices mean a different category of products, leading to a possibly different trend of user preferences. In that case, the optimal segmentized output as a function of the product's price is probably far from having small (higher order) derivatives, and thus cubic splines may perform poorly, and possibly even worse than simple binning.

References

Avazu Advertising Challenge. `https://www.kaggle.com/c/avazu-ctr-prediction`, 2014. [Online; accessed 01-Apr-2024].

Criteo Display Advertising Challenge. `https://www.kaggle.com/c/criteo-display-ad-challenge`, 2014. [Online; accessed 01-Apr-2024].

Takuya Akiba, Shotaro Sano, Toshihiko Yanase, Takeru Ohta, and Masanori Koyama. Optuna: A next-generation hyperparameter optimization framework. In *Proceedings of the 25th ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, 2019.

Sercan Ö. Arik and Tomas Pfister. Tabnet: Attentive interpretable tabular learning. *Proceedings of the AAAI Conference on Artificial Intelligence*, 35(8):6679–6687, May 2021. doi: 10.1609/aaai.v35i8.16826. URL `https://ojs.aaai.org/index.php/AAAI/article/view/16826`.

Sarkhan Badirli, Xuanqing Liu, Zhengming Xing, Avradeep Bhowmik, Khoa Doan, and Sathiya S. Keerthi. Gradient boosting neural networks: Grownet, 2020.

Pierre Baldi, Peter Sadowski, and Daniel Whiteson. Searching for exotic particles in high-energy physics with deep learning. *Nature communications*, 5(1):4308, 2014.

Thierry Bertin-Mahieux, Daniel PW Ellis, Brian Whitman, and Paul Lamere. The million song dataset. 2011.

Mathieu Blondel, Akinori Fujino, Naonori Ueda, and Masakazu Ishihata. Higher-order factorization machines. In D. Lee, M. Sugiyama, U. Luxburg, I. Guyon, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 29. Curran Associates, Inc., 2016. URL https://proceedings.neurips.cc/paper_files/paper/2016/file/158fc2ddd52ec2cf54d3c161f2dd6517-Paper.pdf.

Lars Buitinck, Gilles Louppe, Mathieu Blondel, Fabian Pedregosa, Andreas Mueller, Olivier Grisel, Vlad Niculae, Peter Prettenhofer, Alexandre Gramfort, Jaques Grobler, Robert Layton, Jake VanderPlas, Arnaud Joly, Brian Holt, and Gaël Varoquaux. API design for machine learning software: experiences from the scikit-learn project. In *ECML PKDD Workshop: Languages for Data Mining and Machine Learning*, pp. 108–122, 2013.

Tianqi Chen and Carlos Guestrin. Xgboost: A scalable tree boosting system. In *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, KDD '16, pp. 785–794, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450342322. doi: 10.1145/2939672.2939785. URL https://doi.org/10.1145/2939672.2939785.

Yuan Cheng. Dynamic explicit embedding representation for numerical features in deep ctr prediction. In *Proceedings of the 31st ACM International Conference on Information & Knowledge Management*, pp. 3888–3892, 2022.

Paul Covington, Jay Adams, and Emre Sargin. Deep neural networks for youtube recommendations. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pp. 191–198, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959190. URL https://doi.org/10.1145/2959100.2959190.

F. N. David and N. L. Johnson. The probability integral transformation when parameters are estimated from the sample. *Biometrika*, 35(1/2):182–190, 1948. ISSN 00063444. URL http://www.jstor.org/stable/2332638.

Rügamer David. Additive higher-order factorization machines, 2022.

Carl de Boor. *A Practical Guide to Splines*, volume 27 of *Applied Mathematical Sciences*. Springer-Verlag New York, 2001. ISBN 978-0-387-95366-3.

James Dougherty, Ron Kohavi, and Mehran Sahami. Supervised and unsupervised discretization of continuous features. In Armand Prieditis and Stuart Russell (eds.), *Machine Learning Proceedings 1995*, pp. 194–202. Morgan Kaufmann, San Francisco (CA), 1995. ISBN 978-1-55860-377-6. doi: https://doi.org/10.1016/B978-1-55860-377-6.50032-3. URL https://www.sciencedirect.com/science/article/pii/B9781558603776500323.

João Gama and Carlos Pinto. Discretization from data streams: Applications to histograms and data mining. In *Proceedings of the 2006 ACM Symposium on Applied Computing*, SAC '06, pp. 662–667, New York, NY, USA, 2006. Association for Computing Machinery. ISBN 1595931082. doi: 10.1145/1141277.1141429. URL https://doi.org/10.1145/1141277.1141429.

Yury Gorishniy, Ivan Rubachev, Valentin Khrulkov, and Artem Babenko. Revisiting deep learning models for tabular data. In M. Ranzato, A. Beygelzimer, Y. Dauphin, P.S. Liang, and J. Wortman Vaughan (eds.), *Advances in Neural Information Processing Systems*, volume 34, pp. 18932–18943. Curran Associates, Inc., 2021. URL https://proceedings.neurips.cc/paper_files/paper/2021/file/9d86d83f925f2149e9edb0ac3b49229c-Paper.pdf.

Yury Gorishniy, Ivan Rubachev, and Artem Babenko. On embeddings for numerical features in tabular deep learning. In *Advances in Neural Information Processing Systems*, volume 35, pp. 24991–25004, 2022.

GorupLens. GroupLens datasets. https://grouplens.org/datasets/movielens/. [Online; accessed 01-Apr-2024].

Huifeng Guo, Bo Chen, Ruiming Tang, Weinan Zhang, Zhenguo Li, and Xiuqiang He. An embedding learning framework for numerical features in ctr prediction. In *Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining*, pp. 2910–2918, 2021.

Noah Hollmann, Samuel Müller, Katharina Eggensperger, and Frank Hutter. TabPFN: A transformer that solves small tabular classification problems in a second. In *NeurIPS 2022 First Table Representation Workshop*, 2022. URL `https://openreview.net/forum?id=eu9fVjVasr4`.

Kurt Hornik, Maxwell Stinchcombe, and Halbert White. Multilayer feedforward networks are universal approximators. *Neural Networks*, 2(5):359–366, 1989. ISSN 0893-6080. doi: https://doi.org/10.1016/0893-6080(89)90020-8. URL `https://www.sciencedirect.com/science/article/pii/0893608089900208`.

Xin Huang, Ashish Khetan, Milan Cvitkovic, and Zohar Karnin. Tabtransformer: Tabular data modeling using contextual embeddings, 2020.

Yuchin Juan, Yong Zhuang, Wei-Sheng Chin, and Chih-Jen Lin. Field-aware factorization machines for ctr prediction. In *Proceedings of the 10th ACM Conference on Recommender Systems*, RecSys '16, pp. 43–50, New York, NY, USA, 2016. Association for Computing Machinery. ISBN 9781450340359. doi: 10.1145/2959100.2959134. URL `https://doi.org/10.1145/2959100.2959134`.

Guolin Ke, Qi Meng, Thomas Finley, Taifeng Wang, Wei Chen, Weidong Ma, Qiwei Ye, and Tie-Yan Liu. Lightgbm: A highly efficient gradient boosting decision tree. In I. Guyon, U. Von Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 30. Curran Associates, Inc., 2017. URL `https://proceedings.neurips.cc/paper_files/paper/2017/file/6449f44a102fde848669bdd9eb6b76fa-Paper.pdf`.

Ron Kohavi. Scaling up the accuracy of naive-bayes classifiers: A decision-tree hybrid. In *Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, KDD'96, pp. 202–207. AAAI Press, 1996.

Huan Liu, Farhad Hussain, Chew Lim Tan, and Manoranjan Dash. Discretization: An enabling technique. *Data mining and knowledge discovery*, 6:393–423, 2002.

Ilya Loshchilov and Frank Hutter. Decoupled weight decay regularization, 2019. URL `https://arxiv.org/abs/1711.05101`.

R Kelley Pace and Ronald Barry. Sparse spatial autoregressions. *Statistics & Probability Letters*, 33(3): 291–297, 1997.

Junwei Pan, Jian Xu, Alfonso Lobos Ruiz, Wenliang Zhao, Shengjun Pan, Yu Sun, and Quan Lu. Field-weighted factorization machines for click-through rate prediction in display advertising. In *Proceedings of the 2018 World Wide Web Conference*, WWW '18, pp. 1349–1357, New York, NY, USA, 2018. Association for Computing Machinery. ISBN 9781450356398. doi: 10.1145/3178876.3186040. URL `https://doi.org/10.1145/3178876.3186040`.

Harshit Pande. Field-embedded factorization machines for click-through rate prediction, 2021.

Liu Peng, Wang Qing, and Gu Yujia. Study on comparison of discretization methods. In *2009 International Conference on Artificial Intelligence and Computational Intelligence*, volume 4, pp. 380–384, 2009. doi: 10.1109/AICI.2009.385.

Sergei Popov, Stanislav Morozov, and Artem Babenko. Neural oblivious decision ensembles for deep learning on tabular data. In *International Conference on Learning Representations*, 2020. URL `https://openreview.net/forum?id=r1eiu2VtwH`.

Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett (eds.), *Advances in Neural Information Processing Systems*, volume 31. Curran Associates, Inc., 2018. URL `https://proceedings.neurips.cc/paper_files/paper/2018/file/14491b756b3a51daac41c24863285549-Paper.pdf`.

Steffen Rendle. Factorization machines. In *2010 IEEE International Conference on Data Mining*, pp. 995–1000, 2010. doi: 10.1109/ICDM.2010.127.

Isaac Jacob Schoenberg. Contributions to the problem of approximation of equidistant data by analytic functions. part b. on the problem of osculatory interpolation. a second class of analytic approximation formulae. *Quarterly of Applied Mathematics*, 4(2):112–141, 1946.

Ravid Shwartz-Ziv and Amitai Armon. Tabular data: Deep learning is not all you need. *Information Fusion*, 81:84–90, 2022.

Gowthami Somepalli, Avi Schwarzschild, Micah Goldblum, C. Bayan Bruss, and Tom Goldstein. SAINT: Improved neural networks for tabular data via row attention and contrastive pre-training. In *NeurIPS 2022 First Table Representation Workshop*, 2022. URL `https://openreview.net/forum?id=FiyUTAy4sB8`.

Weiping Song, Chence Shi, Zhiping Xiao, Zhijian Duan, Yewen Xu, Ming Zhang, and Jian Tang. Autoint: Automatic feature interaction learning via self-attentive neural networks. In *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, CIKM '19, pp. 1161–1170, New York, NY, USA, 2019. Association for Computing Machinery. ISBN 9781450369763. doi: 10.1145/3357384.3357925. URL `https://doi.org/10.1145/3357384.3357925`.

Yang Sun, Junwei Pan, Alex Zhang, and Aaron Flores. Fm2: Field-matrixed factorization machines for recommender systems. In *Proceedings of the Web Conference 2021*, WWW '21, pp. 2828–2837, New York, NY, USA, 2021. Association for Computing Machinery. ISBN 9781450383127. doi: 10.1145/3442381.3449930. URL `https://doi.org/10.1145/3442381.3449930`.

Joaquin Vanschoren, Jan N Van Rijn, Bernd Bischl, and Luis Torgo. Openml: networked science in machine learning. *ACM SIGKDD Explorations Newsletter*, 15(2):49–60, 2014.

Pauli Virtanen, Ralf Gommers, Travis E. Oliphant, Matt Haberland, Tyler Reddy, David Cournapeau, Evgeni Burovski, Pearu Peterson, Warren Weckesser, Jonathan Bright, Stéfan J. van der Walt, Matthew Brett, Joshua Wilson, K. Jarrod Millman, Nikolay Mayorov, Andrew R. J. Nelson, Eric Jones, Robert Kern, Eric Larson, C J Carey, İlhan Polat, Yu Feng, Eric W. Moore, Jake VanderPlas, Denis Laxalde, Josef Perktold, Robert Cimrman, Ian Henriksen, E. A. Quintero, Charles R. Harris, Anne M. Archibald, Antônio H. Ribeiro, Fabian Pedregosa, Paul van Mulbregt, and SciPy 1.0 Contributors. SciPy 1.0: Fundamental Algorithms for Scientific Computing in Python. *Nature Methods*, 17:261–272, 2020. doi: 10.1038/s41592-019-0686-2.

Yahoo-Inc. Fully-vectorized weighted field embedding bags for recommender systems. `https://github.com/yahoo/weighted_fields_recsys`, 2023. [Online; accessed 01-May-2023].

A Proof of the spanning properties

First, we write the FmFM model using matrix notation. Then, we use the matrix notation to prove our Lemmas.

A.1 Formalization using linear algebra

To formalize our approach, we denote by $\boldsymbol{v}$ the matrix whose *rows* are the vectors $\boldsymbol{v}_i$, and decompose the FmFM formula equation 1 into three sub-formulas:

$$\begin{aligned}
\boldsymbol{y} &= \operatorname{diag}(\boldsymbol{x})\boldsymbol{w}, \\
\boldsymbol{P} &= \operatorname{diag}(\boldsymbol{x})\boldsymbol{v}, \\
\phi_{\text{FmFM}} &= w_0 + \langle \mathbf{1}, \mathbf{y} \rangle + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle \boldsymbol{P}_i, \boldsymbol{P}_j \rangle_{M_{f_i, f_j}},
\end{aligned}$$

where the diag$(\cdot)$ operator creates a diagonal matrix with the argument on the diagonal, $\mathbf{1}$ is a vector whose components are all 1, and $\boldsymbol{P}_i$ is the ith row of $\boldsymbol{P}$. Next, we associate each field f with a *field reduction* matrix $\mathbf{R}_f$, concatenate them into one big block-diagonal reduction matrix . Note, that neither the block matrices R_f, nor the matrix R have to be square.

$$\mathbf{R} = \begin{bmatrix} \mathbf{R}_1 & \mathbf{0} & \dots & \mathbf{0} \\ \mathbf{0} & \mathbf{R}_2 & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \dots & \mathbf{R}_m \end{bmatrix},$$

and assuming $\mathbf{R}$ has $\hat{n}$ rows, we modify the FmFM formula as:

$$\begin{aligned}
\hat{\boldsymbol{y}} &= \mathbf{R} \operatorname{diag}(\boldsymbol{x})\boldsymbol{w}, \\
\hat{\boldsymbol{P}} &= \mathbf{R} \operatorname{diag}(\boldsymbol{x})\boldsymbol{v}, \\
\phi_{\text{FmFM}} &= w_0 + \langle \mathbf{1}, \hat{\boldsymbol{y}} \rangle + \sum_{i=1}^{\hat{n}} \sum_{j=i+1}^{\hat{n}} \langle \hat{\boldsymbol{P}}_i, \hat{\boldsymbol{P}}_j \rangle_{M_{f_i, f_j}}.
\end{aligned} \tag{2}$$

Setting $\mathbf{R}_f = \mathbf{I}$ for a field f amounts to the identity reduction, whereas setting $\mathbf{R}_f = \mathbf{1}^T$, will cause the scalars $x_i w_i$ and the vectors $x_i \boldsymbol{v}_i$ to be summed up, resulting in the summing reduction . This matrix notation is useful for the study of the theoretical properties of our technique, but in practice we will apply the field reductions manually, as efficiently as possible without matrix multiplication.

A.2 Proof of the spanning property (Lemma 1)

Proof. Recall that we need to rewrite equation 2 as a function of z. For some vector $\mathbf{q}$, denote by $\mathbf{q}_{a:b}$ the sub-vector $(q_a, \dots, q_b)$. Assume w.l.o.g. that $f = 1$, and that field 1 has the value z. By construction in equation 2, we have $\hat{y}_1 = \sum_{i=1}^{\ell} w_i B_i(z)$, while the remaining components $\hat{\boldsymbol{y}}_{2:\hat{n}}$ do not depend on z. Thus, the we have

$$w_0 + \langle \mathbf{1}, \hat{\boldsymbol{y}} \rangle = \sum_{i=1}^{\ell} w_i B_i(z) + \underbrace{w_0 + \langle \mathbf{1}, \hat{\boldsymbol{y}}_{2:\hat{n}} \rangle}_{\beta_1}. \tag{3}$$

Moreover, by equation 2 we have that $\hat{\boldsymbol{P}}_1 = \sum_{i=1}^{\ell} \boldsymbol{v}_i B_i(z)$, whereas the remaining rows $\hat{\boldsymbol{P}}_2, \ldots, \hat{\boldsymbol{P}}_{\hat{n}}$ do not depend on z. Thus,

$$\begin{aligned}
&\sum_{i=1}^{\hat{n}} \sum_{j=i+1}^{\hat{n}} \langle \hat{\boldsymbol{P}}_i, \hat{\boldsymbol{P}}_j \rangle_{M_{f_i,f_j}} \\
&\qquad = \sum_{j=2}^{\hat{n}} \langle \hat{\boldsymbol{P}}_1, \hat{\boldsymbol{P}}_j \rangle_{M_{1,f_j}} + \underbrace{\sum_{i=2}^{\hat{n}} \sum_{j=i+1}^{\hat{n}} \langle \hat{\boldsymbol{P}}_i, \hat{\boldsymbol{P}}_j \rangle_{M_{f_i,f_j}}}_{\beta_2} \\
&\qquad = \sum_{j=2}^{\hat{n}} \langle \sum_{i=1}^{\ell} \boldsymbol{v}_i B_i(z), \hat{\boldsymbol{P}}_j \rangle_{M_{1,f_j}} + \beta_2 \\
&\qquad = \sum_{i=1}^{\ell} \underbrace{\left(\sum_{j=2}^{\hat{n}} \langle \boldsymbol{v}_i, \hat{\boldsymbol{P}}_j \rangle_{M_{1,f_j}} \right)}_{\tilde{\alpha}_i} B_i(z) + \beta_2.
\end{aligned} \tag{4}$$

Combining equation 3 and equation 4, we obtain

$$\phi_{\mathrm{FmFM}}(z) = \sum_{i=1}^{\ell} (w_i + \tilde{\alpha}_i) B_i(z) + (\beta_1 + \beta_2),$$

which is of the desired form. □

A.3 Proof of the pairwise spanning property (Lemma 2)

Proof. Recall that we need to rewrite equation 2 as a function of z_e, z_f, which are the values of the fields e and f. Assume w.l.o.g. that $e = 1, f = 2$. By construction in equation 2, we have $\hat{y}_1 = \sum_{i=1}^{\ell} w_i B_i(z_1)$, and $\hat{y}_2 = \sum_{i=1}^{\kappa} w_i C_i(z_2)$, while the remaining components $\hat{\boldsymbol{y}}_{3:\hat{n}}$ do not depend on z. Thus, the we have

$$\begin{aligned}
w_0 + \langle \mathbf{1}, \hat{\boldsymbol{y}} \rangle &= \sum_{i=1}^{\ell} w_i B_i(z_1) + \sum_{i=1}^{\kappa} w_{i+\ell} C_i(z_2) + \underbrace{w_0 + \langle \mathbf{1}, \hat{\boldsymbol{y}}_{3:\hat{n}} \rangle}_{\beta_1} \\
&= \sum_{i=0}^{\ell} \sum_{j=0}^{\kappa} \hat{\alpha}_{i,j} B_i(z_1) C_j(z_2) + \beta_1,
\end{aligned} \tag{5}$$

where $\hat{\alpha}_{i,0} = w_i, \hat{\alpha}_{0,j} = w_{i+\ell}$ for all $i, j \geq 1$, for all other values of i, j we set $\hat{\alpha}_{i,j} = 0$.

Moreover, by equation 2 we have that $\hat{\boldsymbol{P}}_1 = \sum_{i=1}^{\ell} \boldsymbol{v}_i B_i(z_1)$ and $\hat{\boldsymbol{P}}_2 = \sum_{i=1}^{\kappa} \boldsymbol{v}_i C_{i+\ell}(z_2)$, whereas the remaining rows $\hat{\boldsymbol{P}}_3, \ldots, \hat{\boldsymbol{P}}_{\hat{n}}$ do not depend on z_1, z_2. First, let us rewrite the interaction between z_1, z_2 specifically:

$$\begin{aligned}
\langle p_1, p_2 \rangle_{M_{1,2}} &= \langle \sum_{i=1}^{\ell} \boldsymbol{v}_i B_i(z), \sum_{i=1}^{\kappa} \boldsymbol{v}_{i+\ell} C_i(z) \rangle_{M_{1,2}} \\
&= \sum_{i=1}^{\ell} \sum_{j=1}^{\kappa} \underbrace{\langle \boldsymbol{v}_i, \boldsymbol{v}_{j+\ell} \rangle_{M_{1,2}}}_{\gamma_{i,j}} B_i(z_1) C_j(z_2)
\end{aligned} \tag{6}$$

By following similar logic to 4, one can obtain a similar expression when looking at the partial sums that include the interaction between f (resp. e) and all other fields except e (resp. f). Observe that the interaction between all other values does not depend on z_1, z_2. Given all of this, we show how to rewrite the interaction as a function of z_1, z_2:

$$\begin{aligned}
&\sum_{i=1}^{\hat{n}} \sum_{j=i+1}^{\hat{n}} \langle \hat{\boldsymbol{P}}_i, \hat{\boldsymbol{P}}_j \rangle_{M_{f_i,f_j}} \\
&\quad = \langle \hat{\boldsymbol{P}}_1, \hat{\boldsymbol{P}}_2 \rangle_{M_{1,2}} + \sum_{j=3}^{\hat{n}} \langle \hat{\boldsymbol{P}}_1, \hat{\boldsymbol{P}}_j \rangle_{M_{1,f_j}} \\
&\qquad + \sum_{j=3}^{\hat{n}} \langle \hat{\boldsymbol{P}}_2, \hat{\boldsymbol{P}}_j \rangle_{M_{2,f_i}} + \underbrace{\sum_{i=3}^{\hat{n}} \sum_{j=i+1}^{\hat{n}} \langle \hat{\boldsymbol{P}}_i, \hat{\boldsymbol{P}}_j \rangle_{M_{f_i,f_j}}}_{\beta_2} \\
&\quad = \sum_{i=1}^{\ell} \sum_{j=1}^{\kappa} \gamma_{i,j} B_i(z_1) C_j(z_2) + \sum_{i=1}^{\ell} \tilde{\alpha}_i B_i(z_1) \\
&\qquad + \sum_{i=1}^{\kappa} \bar{\alpha}_i C_i(z_2) + \beta_2
\end{aligned}$$

where $\tilde{\alpha}_i, \bar{\alpha}_i$ are obtained similarly in equation 4's final step. □

B The factorization machine family

Factorization machines are formally described as models whose input is a feature vector $\boldsymbol{x}$ representing rows in tabular data-sets, as described in section 2. In the context of recommendation systems, the data-set contains past interactions between users and items, whose columns, often named *fields*, and whose values are *features*. The columns describe the context, such as the user's gender and age, the time of visit, or the article the user is currently reading, whereas others describe the item, such as product category, or item popularity.

The initial Factorization Machines (FMs), as proposed in Rendle (2010), compute

$$\Phi_{\text{FM}}(\boldsymbol{x}) = w_0 + \langle \boldsymbol{x}, \boldsymbol{w} \rangle + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle x_i \boldsymbol{v}_i, x_j \boldsymbol{v}_j \rangle.$$

The learned parameters are $w_0 \in \mathbb{R}$, $\boldsymbol{w} \in \mathbb{R}^n$, and $\boldsymbol{v}_1, \dots, \boldsymbol{v}_n \in \mathbb{R}^k$, where k is a hyper-parameter. The model can be thought of as a way to represent the quadratic interaction model

$$\Phi_{\text{quad}}(\boldsymbol{x}) = w_0 + \langle \boldsymbol{x}, \boldsymbol{w} \rangle + \sum_{i=1}^{n} \sum_{j=i+1}^{n} A_{i,j} x_i x_j,$$

where the coefficient matrix $\boldsymbol{A}$ is represented in factorized form. The vectors $\boldsymbol{v}_1, \dots, \boldsymbol{v}_n$ are the feature embedding vectors. Classical matrix factorization is recovered when we have only user id and item id fields, whose values are one-hot encoded. We note that this is a special case of the FmFM model in equation 1, with $\boldsymbol{M}_{f_i,f_j} = \boldsymbol{I}$.

The Φ_{FM} model does not represent the varying behavior of a feature belonging to some field when interacting with features from different fields. For example, genders may interact with ages differently than they interact with product categories. Initially, to explicitly encode this information into a model, the Field-aware Factorization Machine (FFM) was proposed in Juan et al. (2016). Each embedding vector $\boldsymbol{v}_i$ is modeled as a concatenation of field-specific embedding vectors for each of the m fields:

$$\boldsymbol{v}_i = \begin{bmatrix} \boldsymbol{v}_{i,1} \\ \vdots \\ \boldsymbol{v}_{i,m} \end{bmatrix},$$

where $\boldsymbol{v}_{i,f}$ is the embedding vector of feature i when interacting with another feature from a field f. The model computes

$$\Phi_{\text{FFM}}(\boldsymbol{x}) = w_0 + \langle \boldsymbol{x}, \boldsymbol{w} \rangle + \sum_{i=1}^{n} \sum_{j=i+1}^{n} \langle x_i \boldsymbol{v}_{i,f_j}, x_j \boldsymbol{v}_{j,f_i} \rangle$$

For any field f, let $\boldsymbol{P}_f$ be the matrix that extracts $\boldsymbol{v}_{i,f}$ from $\boldsymbol{v}_i$, namely, $\boldsymbol{P}_f \boldsymbol{v}_i = \boldsymbol{v}_{i,f}$. Then FFMs are also a special case of the FmFM model in equation 1 with $\boldsymbol{M}_{e,f} = \boldsymbol{P}_f^T \boldsymbol{P}_e$.

As pointed out by Pan et al. (2018); Juan et al. (2016), the FFM models are prone to over-fitting, since it learns a feature embedding vector for each feature x field pair. As a remedy, Pan et al. (2018) proposed the Field-Weighted Factorization Machine (FwFM) that models the varying behavior of field interaction using learned scalar field interaction intensities $r_{e,f}$ for each pair of fields e, f. The FwFM computes

$$\Phi_{\text{FwFM}}(\boldsymbol{x}) = w_0 + \langle \boldsymbol{x}, \boldsymbol{w} \rangle + \sum_{i=1}^{n} \sum_{j=i+1}^{n} r_{i,j} \langle x_i \boldsymbol{v}_i, x_j \boldsymbol{v}_j \rangle.$$

Letting $\boldsymbol{M}_{e,f} = r_{i,j} \boldsymbol{I}$, we recover the FmFM model in equation 1.