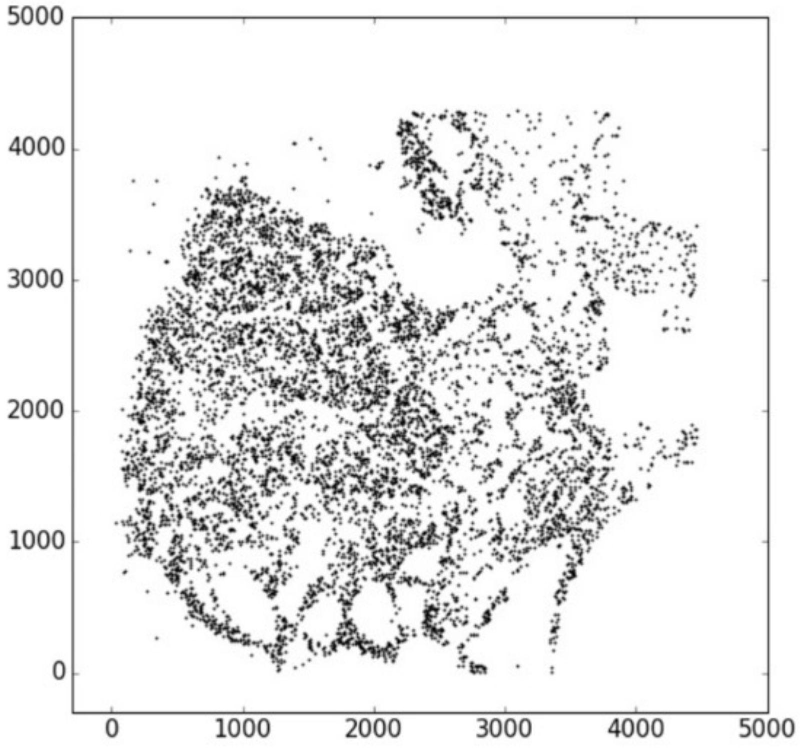
连续特征嵌入：离散桶与傅立叶特征

INSIS时空数据挖掘组-论文分享

2023.9 林彦

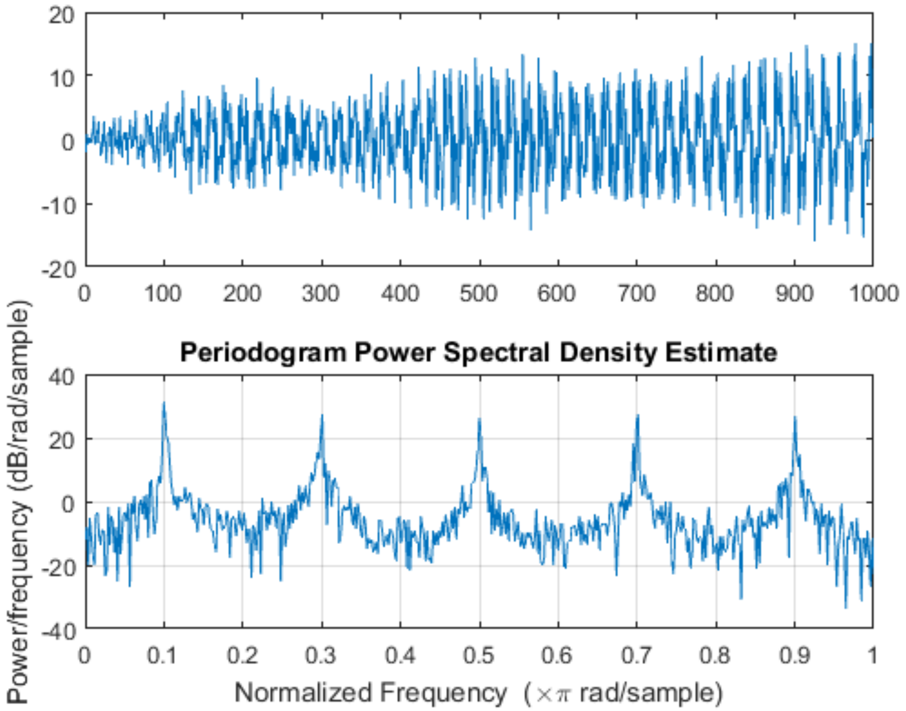
时空数据中的连续特征

- 连续特征在时空数据中占主导地位，同时时空连续特征的表达较为复杂
- 连续值的嵌入在许多任务中至关重要



空间特征通常由高频和低频信息混合构成

https://www.nature.com/articles/s41598-019-41951-2/figures/1



时间序列在时间维度上通常具有周期性

https://stats.stackexchange.com/questions/191471/periodicity-and-seasonality-of-a-time-series

连续特征嵌入传统方案

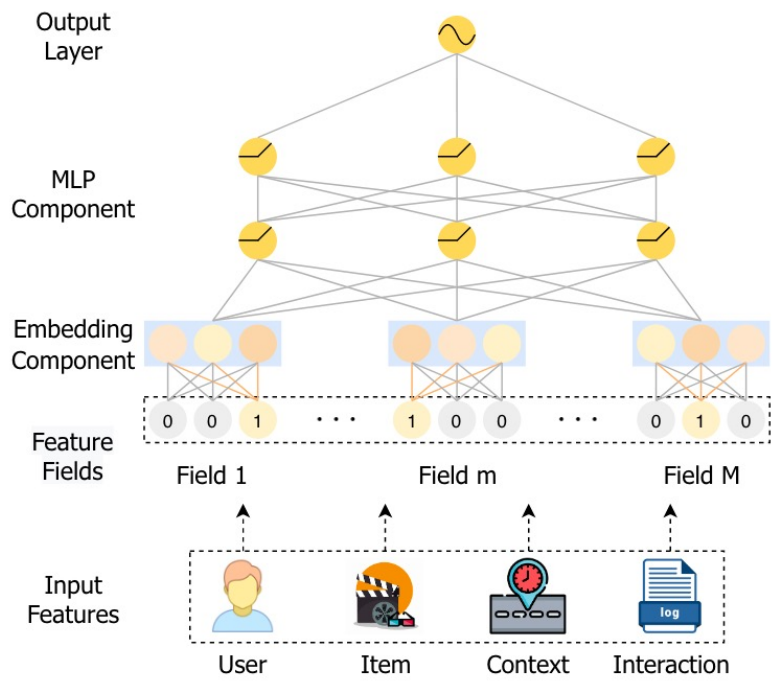
➢ 无嵌入

- 直接将连续特征作为神经网络模型的输入特征
- 缺乏表达能力

➢ 领域嵌入

- 根据连续特征所属领域，为每个领域分配嵌入向量$\boldsymbol{z} \in \mathbb{R}^d$
- 连续值的嵌入计算为$x\boldsymbol{z}$
- 表达能力依然不足，同时在高频数据上尤其表现不佳

推荐系统中常用的领域嵌入



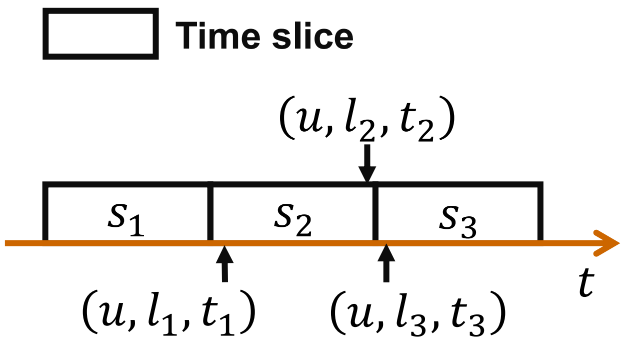
[1] Zhao X, Liu H, Liu H, et al. Memory-efficient embedding for recommendations[J]. arXiv preprint arXiv:2006.14827, 2020.
[2] Grinsztajn L, Oyallon E, Varoquaux G. Why do tree-based models still outperform deep learning on typical tabular data?[J]. Advances in Neural Information Processing Systems, 2022, 35: 507-520.

分享内容大纲

- 连续特征嵌入
  - 桶嵌入
    - 规则式离散化
    - 融合线性的PLE离散化
    - AutoDis可优化的离散化
  - 傅立叶特征
    - 随机傅立叶特征
    - 位置编码
    - 可学习的傅立叶特征

规则式离散化 & 离散嵌入

- 按预定义的规则将连续值分配至离散桶内，结合离散特征嵌入
- 二阶段方案 - 预定义的规则无法随梯度下降优化
- 相似值被拆分 - 可能将语义上相似的值划分到不同桶内，导致嵌入不相似
- 不相似值被合并 - 可能将语义上不相似的值划分到同一桶内



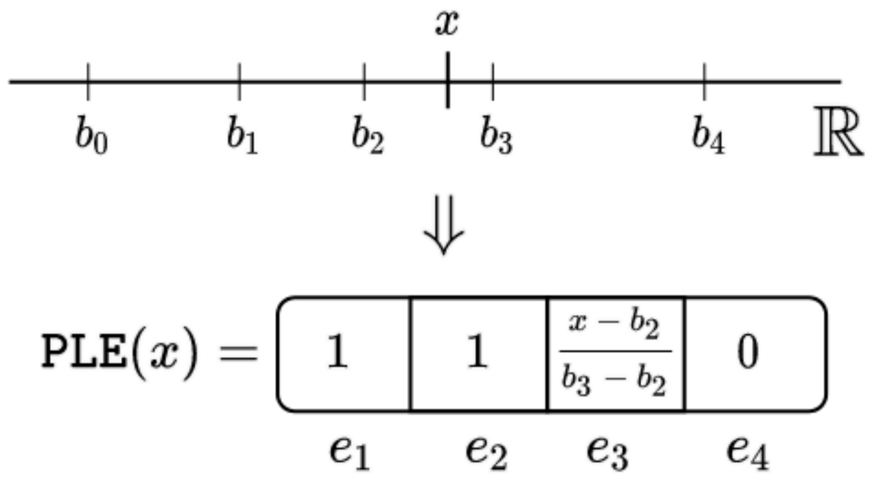
预定义的规则可能产生不准确的结果，且难以随神经网络模型优化

PLE: 连续值的线性离散化

- 依然按规则离散化连续值，但设计更加线性的离散方案
- 相比简单的离散化，能够缓解相似值拆分和不相似值合并的问题

$$\mathrm{PLE}(x) = [e_1, \ldots, e_T] \in \mathbb{R}^T$$

$$e_t = \begin{cases} 0, & x < b_{t-1} \texttt{ AND } t > 1 \\ 1, & x \geq b_t \texttt{ AND } t < T \\ \frac{x - b_{t-1}}{b_t - b_{t-1}}, & \text{otherwise} \end{cases} \quad (1)$$



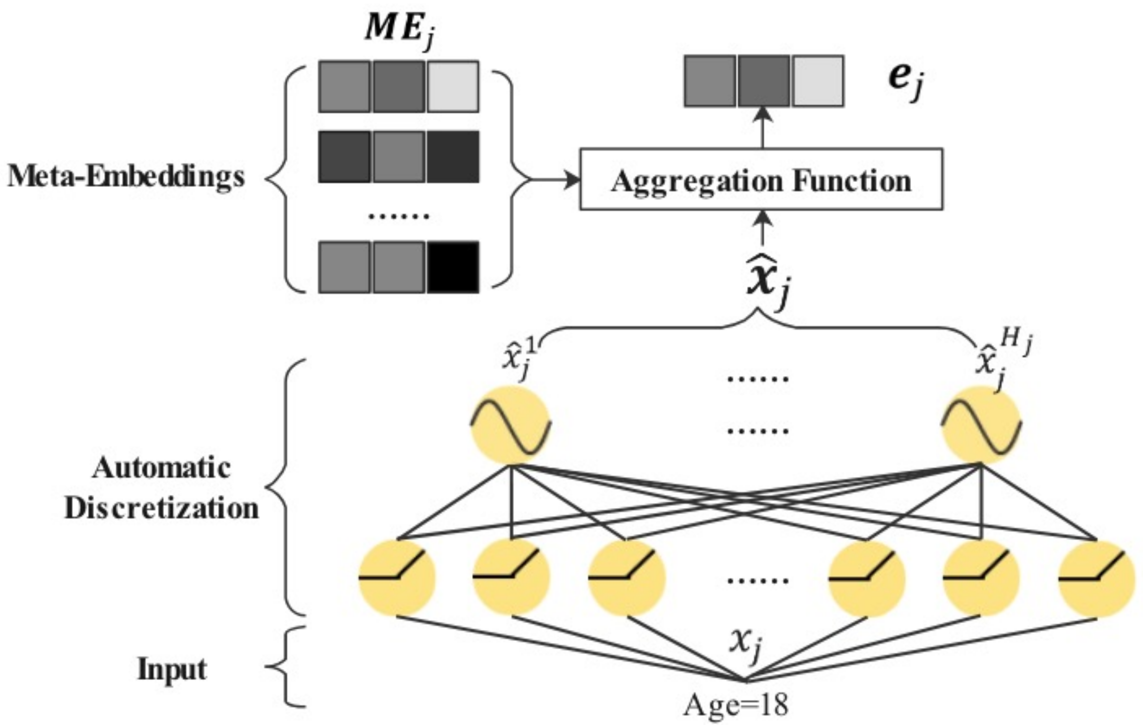
Gorishniy Y, Rubachev I, Babenko A. On embeddings for numerical features in tabular deep learning[J]. Advances in Neural Information Processing Systems, 2022, 35: 24991-25004.

AutoDis: 可优化的连续值离散嵌入

- 借助MLP，构建可反向传播的离散化方案
- 技术上接近全连接嵌入与领域嵌入的组合

$$\widehat{\mathbf{x}}_j = d_j^{Auto}(x_j) = [\widehat{x}_j^1, \ldots, \widehat{x}_j^h, \ldots, \widehat{x}_j^{H_j}].$$

$$\boldsymbol{e}_j = \sum \widehat{x}_j \cdot \boldsymbol{E}_j$$



Guo H, Chen B, Tang R, et al. An embedding learning framework for numerical features in ctr prediction[C]//Proceedings of the 27th ACM SIGKDD Conference on Knowledge Discovery & Data Mining. 2021: 2910-2918.

Fourier Features: 多级周期性嵌入

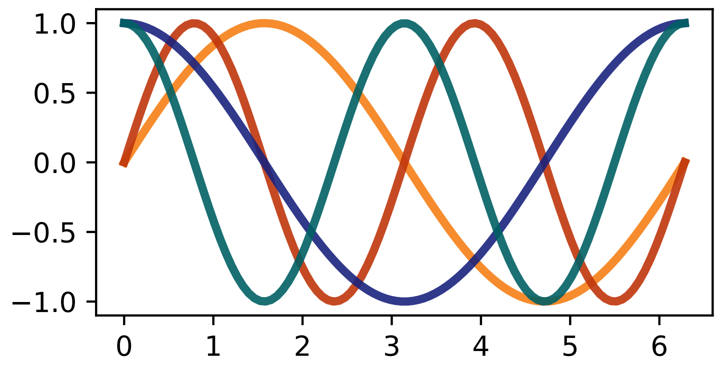
➢ Random Fourier Features

- 由三角函数和随机系数构成的指定维度编码
- 从核函数定义的概率中取样系数，决定某个维度的周期频率

➢ Positional Encoding

- 由单调函数计算并固定系数，周期频率确定不变

三角函数周期性



Compute the Fourier transform p of the kernel k: $p(\omega) = \frac{1}{2\pi}\int e^{-j\omega'\Delta}k(\Delta)\, d\Delta$.
Draw D iid samples $\omega_1, \cdots, \omega_D \in \mathcal{R}^d$ from p.
Let $\mathbf{z}(\mathbf{x}) \equiv \sqrt{\frac{1}{D}}\left[\cos(\omega_1'\mathbf{x}) \;\cdots\; \cos(\omega_D'\mathbf{x}) \; \sin(\omega_1'\mathbf{x}) \;\cdots\; \sin(\omega_D'\mathbf{x})\right]'$.

Random Fourier Features计算算法

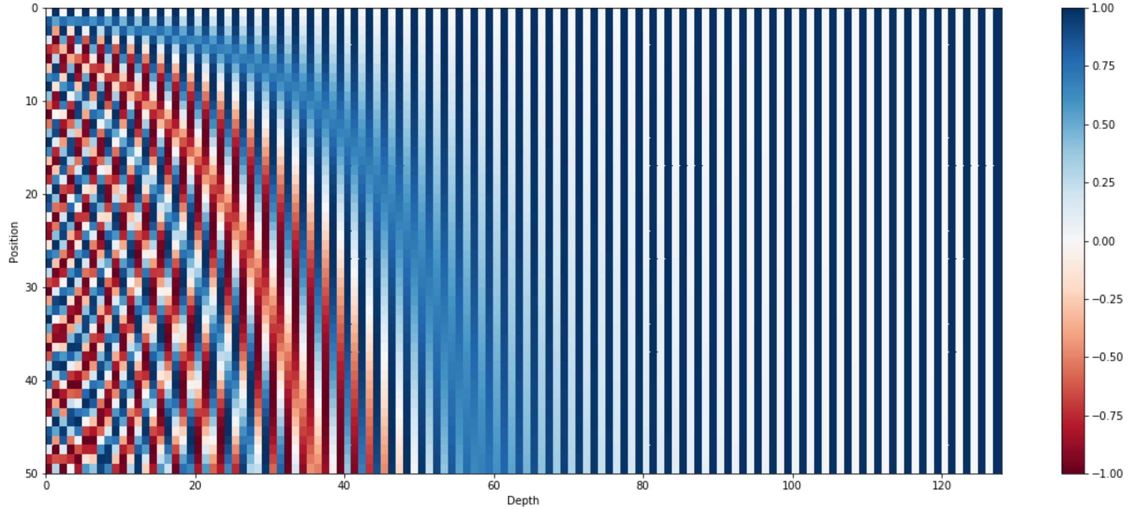
$$PE_{(pos,2i)} = sin(pos/10000^{2i/d_{\text{model}}})$$
$$PE_{(pos,2i+1)} = cos(pos/10000^{2i/d_{\text{model}}})$$

Transformer中的位置编码

[1] Rahimi A, Recht B. Random features for large-scale kernel machines[J]. Advances in neural information processing systems, 2007, 20.
[2] Vaswani A, Shazeer N, Parmar N, et al. Attention is all you need[J]. Advances in neural information processing systems, 2017, 30.

Fourier Features: 周期性与平移不变性

0 :	0	0	0	0	8 :	1	0	0	0
1 :	0	0	0	1	9 :	1	0	0	1
2 :	0	0	1	0	10 :	1	0	1	0
3 :	0	0	1	1	11 :	1	0	1	1
4 :	0	1	0	0	12 :	1	1	0	0
5 :	0	1	0	1	13 :	1	1	0	1
6 :	0	1	1	0	14 :	1	1	1	0
7 :	0	1	1	1	15 :	1	1	1	1

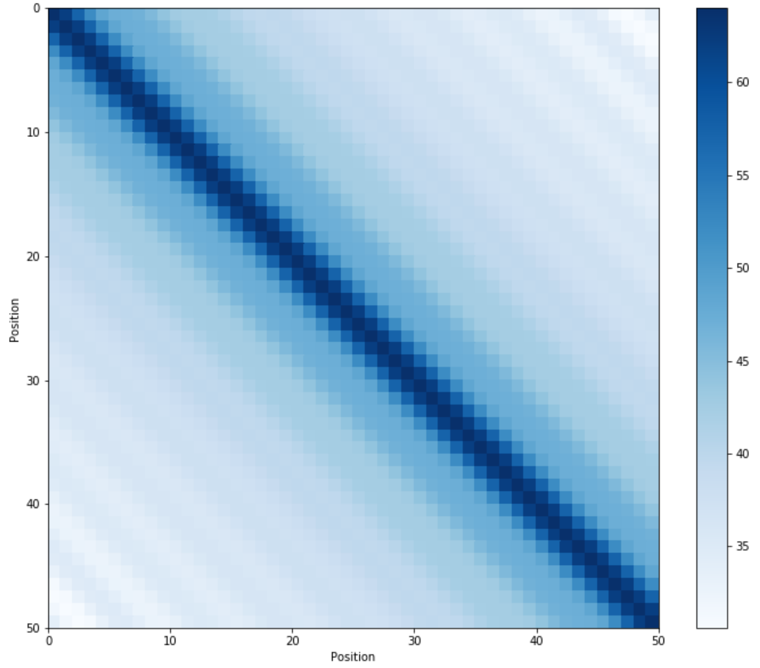


类比二进制编码理解周期性

https://kazemnejad.com/blog/transformer_architecture_positional_encoding/

$$\cos(a-b) = \cos a \cos b + \sin a \sin b$$

$$\mathrm{PE}(x) \cdot \mathrm{PE}(x+\delta) = \sum_{i=1}^{d} \cos(\omega_i \delta)$$



编码向量内积的平移不变形

可学习的Fourier Features

- 将固定系数替换为可学习参数，引入可学习的周期频率和距离信息

$$r_x = \frac{1}{\sqrt{D}}[\cos xW_r^T \parallel \sin xW_r^T]$$

$$\Downarrow$$

$$r_x \cdot r_y = \frac{1}{D}\mathrm{sum}\big(\cos((x-y)W_r^T)\big)$$

$$\gamma(\mathbf{v}) = \left[a_1 \cos(2\pi \mathbf{b}_1^{\mathrm{T}}\mathbf{v}), a_1 \sin(2\pi \mathbf{b}_1^{\mathrm{T}}\mathbf{v}), \ldots, a_m \cos(2\pi \mathbf{b}_m^{\mathrm{T}}\mathbf{v}), a_m \sin(2\pi \mathbf{b}_m^{\mathrm{T}}\mathbf{v})\right]^{\mathrm{T}}$$

$$\Downarrow$$

$$\gamma(\mathbf{v}_1)^{\mathrm{T}}\gamma(\mathbf{v}_2) = \sum_{j=1}^{m} a_j^2 \cos\left(2\pi \mathbf{b}_j^{\mathrm{T}}(\mathbf{v}_1 - \mathbf{v}_2)\right)$$

[1] Tancik M, Srinivasan P, Mildenhall B, et al. Fourier features let networks learn high frequency functions in low dimensional domains[J]. Advances in Neural Information Processing Systems, 2020, 33: 7537-7547.

[2] Li Y, Si S, Li G, et al. Learnable fourier features for multi-dimensional spatial positional encoding[J]. Advances in Neural Information Processing Systems, 2021, 34: 15816-15829.

扩展阅读

- Positional Encoding讨论：[Transformer Architecture: The Positional Encoding - Amirhossein Kazemnejad's Blog]
- 基于Bochner’s theorem推导的Time Encoding: [Inductive Representation Learning on Temporal Graphs]
- 关于MLP结构在高频数据上表现欠佳的讨论： [Why do tree-based models still outperform deep learning on typical tabular data]

实现代码

- 两种Learnable Fourier Features：tancik/fourier-feature-networks / willGuimont/learnable_fourier_positional_encoding
- PLE与AutoDis：自行实现代码，详见附件

```python
class AutoDis(nn.Module):
    def __init__(self, num_buckets, d_model, skip_factor):
        super().__init__()

        self.input_mlp = nn.Sequential(nn.Linear(1, num_buckets, bias=False), nn.LeakyReLU())
        self.hidden_mlp = nn.Linear(num_buckets, num_buckets)
        self.meta_embeds = nn.Parameter(torch.normal(0, 1, size=(num_buckets, d_model)), requires_grad=True)
        self.skip_factor = skip_factor

    def forward(self, x):
        x = x.unsqueeze(-1)
        h = self.input_mlp(x)
        buckets = F.softmax(self.hidden_mlp(h) + self.skip_factor * h, dim=-1)  # (B, L, N)
        embed = repeat(buckets, 'B L N -> B L N 1') * repeat(self.meta_embeds, 'N E -> 1 1 N E')
        embed = embed.sum(-2)
        return embed
```

```python
class PLE(nn.Module):
    def __init__(self, num_buckets, min_val, max_val):
        super().__init__()

        self.num_buckets = num_buckets
        self.min_val = min_val
        self.max_val = max_val

    def forward(self, x):
        bucket_index = (x - self.min_val) / (self.max_val - self.min_val) * self.num_buckets  # (B, L)
        offset = bucket_index - torch.floor(bucket_index)
        buckets = (repeat(torch.arange(self.num_buckets).to(x.device), 'N -> 1 1 N')
                    < bucket_index.unsqueeze(-1)).float()  # (B, L, N)
        embed = torch.where(repeat(torch.arange(self.num_buckets).to(x.device), 'N -> B L N', B=x.size(0), L=x.size(1)) ==
                            torch.ceil(bucket_index).unsqueeze(-1),
                            repeat(offset, 'B L -> B L N', N=self.num_buckets),
                            buckets)
        return embed
```

感谢聆听！