

机器学习导论-贝叶斯分类器

Introduction to Machine Learning-Bayes Classifier

李文斌

南京大学 智能科学与技术学院

www.liwenbin.cn, liwenbin@nju.edu.cn

2025年03月31日

贝叶斯分类器

□ 贝叶斯决策论 (Bayesian decision theory)

✓ 概率框架下实施决策的基本理论

给定 N 个类别, 令 λ_{ij} 代表将第 j 类样本误分类为第 i 类所产生的损失, 则基于后验概率将样本 \mathbf{x} 分到第 i 类的条件风险为:

$$R(c_i|\mathbf{x}) = \sum_{j=1}^N \lambda_{ij} P(c_j|\mathbf{x})$$

贝叶斯判定准则 (Bayes decision rule):

$$h^*(\mathbf{x}) = \underset{c \in \mathcal{Y}}{\operatorname{argmin}} R(c|\mathbf{x})$$

- h^* 称为**贝叶斯最优分类器** (Bayes optimal classifier), 其总体风险称为**贝叶斯风险** (Bayes risk)
- $1 - R(h^*)$ 反映了**学习性能的理论上限**

贝叶斯分类器

□ 判别式 VS. 生成式

✓ $P(c|x)$ 在现实中通常难以获得

- 从这个角度来看，机器学习所要实现的是基于有限的训练样本尽可能准确地评估出后验概率

✓ 两种基本策略

判别式 (discriminative) 模型

思路：直接对 $P(c|x)$ 建模

代表：

- 决策树
- BP 神经网络
- SVM

生成式 (generative) 模型

思路：先对联合概率分布 $P(x, c)$ 建模，再由此获得 $P(c|x)$

$$P(c | x) = \frac{P(x, c)}{P(x)}$$

代表：贝叶斯分类器

贝叶斯分类器

□ 贝叶斯定理

$$\checkmark P(c|\mathbf{x}) = \frac{P(\mathbf{x},c)}{P(\mathbf{x})}$$

- 根据贝叶斯定理，有：

$$P(c|\mathbf{x}) = \frac{P(c)P(\mathbf{x}|c)}{P(\mathbf{x})}$$

先验概率 (prior)

样本空间中各类样本所占的比例，可通过各类样本出现的频率估计（大数定律）

样本相对于类标记的类条件概率 (class-conditional probability), 亦称 似然 (likelihood)

证据 (evidence) 因子，与类别无关

主要困难在于估计似然 $P(\mathbf{x}|c)$



Thomas Bayes
(1701?-1761)

贝叶斯分类器

□ 极大似然估计

✓ 先假设某种概率分布形式，再基于训练样例对参数进行估计

- 假定 $P(x|c)$ 具有确定的概率分布形式，且被参数 θ_c 唯一确定，则任务就是利用训练集 D 来估计参数 θ_c
- θ_c 对于训练集 D 中第 c 类样本组成的集合 D_c 的似然(likelihood)为

$$P(D_c|\theta_c) = \prod_{x \in D_c} P(x|\theta_c)$$

- 连乘易造成下溢，因此通常使用对数似然 (log-likelihood)

$$LL(\theta_c) = \log P(D_c|\theta_c) = \sum_{x \in D_c} \log P(x|\theta_c)$$

- 于是， θ_c 的极大似然估计为： $\theta_c = \underset{\theta_c}{\operatorname{argmax}} LL(\theta_c)$

估计结果的准确性严重依赖于所假设的概率分布形式是否符合潜在的真实分布

贝叶斯分类器

□ 极大似然估计 (续)

- 例如，假设连续属性的概率密度函数 $P(\mathbf{x}|c) \sim N(\boldsymbol{\mu}_c, \boldsymbol{\sigma}_c^2)$ ，则参数 $\boldsymbol{\mu}_c$ 和 $\boldsymbol{\sigma}_c^2$ 的极大似然估计为

$$\hat{\boldsymbol{\mu}}_c = \frac{1}{|D_c|} \sum_{\mathbf{x} \in D_c} \mathbf{x}$$
$$\hat{\boldsymbol{\sigma}}_c^2 = \frac{1}{|D_c|} \sum_{\mathbf{x} \in D_c} (\mathbf{x} - \hat{\boldsymbol{\mu}}_c)(\mathbf{x} - \hat{\boldsymbol{\mu}}_c)^T$$

- 即，通过极大似然法得到的：

正态分布均值就是样本均值；

方差就是 $(\mathbf{x} - \hat{\boldsymbol{\mu}}_c)(\mathbf{x} - \hat{\boldsymbol{\mu}}_c)^T$ 的均值

估计结果的准确性严重依赖于所假设的概率分布形式是否符合潜在的真实分布

贝叶斯分类器

□ 朴素贝叶斯分类器 (naïve Bayes classifier)

$$P(c|\mathbf{x}) = \frac{P(c)P(\mathbf{x}|c)}{P(\mathbf{x})}$$

- 主要障碍：所有属性上的联合概率难以从有限训练样本估计获得
 - ◆ 组合爆炸；样本稀疏
- 基本思路：假设属性相互独立？

$$P(c|\mathbf{x}) = \frac{P(c)P(\mathbf{x}|c)}{P(\mathbf{x})} = \frac{P(c)}{P(\mathbf{x})} \prod_{i=1}^d P(x_i|c)$$

d 为属性数, x_i 为 \mathbf{x} 在第 i 个属性上的取值

- $P(\mathbf{x})$ 对所有类别相同, 于是

$$h_{nb}(\mathbf{x}) = \operatorname{argmax}_{c \in \mathcal{Y}} P(c) \prod_{i=1}^d P(x_i|c)$$

贝叶斯分类器

□ 朴素贝叶斯分类器 (naïve Bayes classifier)

- 估计 $P(c)$: $P(c) = \frac{|D_c|}{|D|}$

- 估计 $P(x|c)$:

- ◆ 对离散属性, 令 D_{c,x_i} 表示 D_c 中在第 i 个属性上取值为 x_i 的样本组成的集合, 则

$$P(x_i|c) = \frac{|D_{c,x_i}|}{|D_c|}$$

- ◆ 对连续属性, 考虑概率密度函数, 假定 $P(x_i|c) \sim N(\mu_{c,i}, \sigma_{c,i}^2)$

$$P(x_i|c) = \frac{1}{\sqrt{2\pi}\sigma_{c,i}} \exp\left(-\frac{(x_i - \mu_{c,i})^2}{2\sigma_{c,i}^2}\right)$$

贝叶斯分类器

□ 一个例子

✓ (青绿; 稍蜷; 浊响; 清晰)

✓ -好瓜 or 坏瓜?

$$P(\text{青绿}|\text{好瓜}) = 3/8 \quad P(\text{青绿}|\text{坏瓜}) = 3/9$$

$$P(\text{稍蜷}|\text{好瓜}) = 3/8 \quad P(\text{稍蜷}|\text{坏瓜}) = 4/9$$

$$P(\text{浊响}|\text{好瓜}) = 6/8 \quad P(\text{浊响}|\text{坏瓜}) = 4/9$$

$$P(\text{清晰}|\text{好瓜}) = 7/8 \quad P(\text{清晰}|\text{坏瓜}) = 2/9$$

$$P(\text{好瓜}=\text{yes}) = 8/17 \quad P(\text{好瓜}=\text{no}) = 9/17$$

$$\begin{aligned} &P(\text{青绿}|\text{好瓜}) P(\text{稍蜷}|\text{好瓜}) P(\text{浊响}|\text{好瓜}) P(\text{清晰}|\text{好瓜}) P(\text{好瓜}=\text{yes}) \\ &= 3/8 * 3/8 * 6/8 * 7/8 * 8/17 \end{aligned}$$

$$\begin{aligned} &P(\text{青绿}|\text{坏瓜}) P(\text{稍蜷}|\text{坏瓜}) P(\text{浊响}|\text{坏瓜}) P(\text{清晰}|\text{坏瓜}) P(\text{好瓜}=\text{no}) \\ &= 3/9 * 4/9 * 4/9 * 2/9 * 9/17 \end{aligned}$$

好瓜!

编号	色泽	根蒂	敲声	纹理	好瓜
1	青绿	蜷缩	浊响	清晰	是
2	乌黑	蜷缩	沉闷	清晰	是
3	乌黑	蜷缩	浊响	清晰	是
4	青绿	蜷缩	沉闷	清晰	是
5	浅白	蜷缩	浊响	清晰	是
6	青绿	稍蜷	浊响	清晰	是
7	乌黑	稍蜷	浊响	稍糊	是
8	乌黑	稍蜷	浊响	清晰	是
9	乌黑	稍蜷	沉闷	稍糊	否
10	青绿	硬挺	清脆	清晰	否
11	浅白	硬挺	清脆	模糊	否
12	浅白	蜷缩	浊响	模糊	否
13	青绿	稍蜷	浊响	稍糊	否
14	浅白	稍蜷	沉闷	稍糊	否
15	乌黑	稍蜷	浊响	清晰	否
16	浅白	蜷缩	浊响	模糊	否
17	青绿	蜷缩	沉闷	稍糊	否

贝叶斯分类器

□ 拉普拉斯修正 (Laplacian correction)

- 若某个属性值在训练集中没有与某个类同时出现过，则直接计算会出现问题，因为概率连乘将“抹去”其他属性提供的信息
 - 例如，若训练集中未出现“敲声=清脆”的好瓜，则模型在遇到“敲声=清脆”的测试样本时
- 令 N 表示训练集 D 中可能的类别数， N_i 表示第 i 个属性可能的取值数

$$\hat{P}(c) = \frac{|D_c| + 1}{|D| + N}, \quad \hat{P}(x_i|c) = \frac{|D_{c,x_i}| + 1}{|D_c| + N_i}$$

假设了属性值与类别的均匀分布，这是额外引入的 bias

贝叶斯分类器

□ 朴素贝叶斯分类器的使用

✓ 对预测速度要求高

- ✓ 预计算所有概率估值，使用时“查表”

✓ 若数据更替频繁

- ◆ 不进行任何训练，收到预测请求时再估值
(懒惰学习, lazy learning)

✓ 若数据不增加

- ◆ 基于现有估值，对新样本涉及的概率估值进行修正
(增量学习, incremental learning)

贝叶斯分类器

□ 半朴素贝叶斯分类器 (semi-naïve Bayes classifier)

✓ 朴素贝叶斯分类器的“属性独立性假设”在现实中往往难以成立

✓ 半朴素贝叶斯分类器的基本思路：

◆ 适当考虑一部分属性间的相互依赖信息

✓ 最常用策略

◆ 独依赖估计 (One-Dependent Estimator, ODE)

假设每个属性类别之外最多仅依赖一个其他属性

$$P(c|\mathbf{x}) \propto P(c) \prod_{i=1}^d p(x_i|c, \boxed{pa_i}) \quad x_i \text{ 的 “父属性”}$$

关键是如何确定父属性

贝叶斯分类器

□ 两种常见方法

✓ SPODE (Super-Parent ODE)

- ◆ 假设所有属性都依赖于同一属性，称为“超父” (Super-Parent)，然后通过交叉验证等模型选择方法来确定超父属性

✓ TAN (Tree Augmented naïve Bayes)

- ◆ 以属性间的条件“互信息” (mutual information) 为边的权重，构建完全图，再利用最大带权生成树算法，仅保留强相关属性间的依赖性

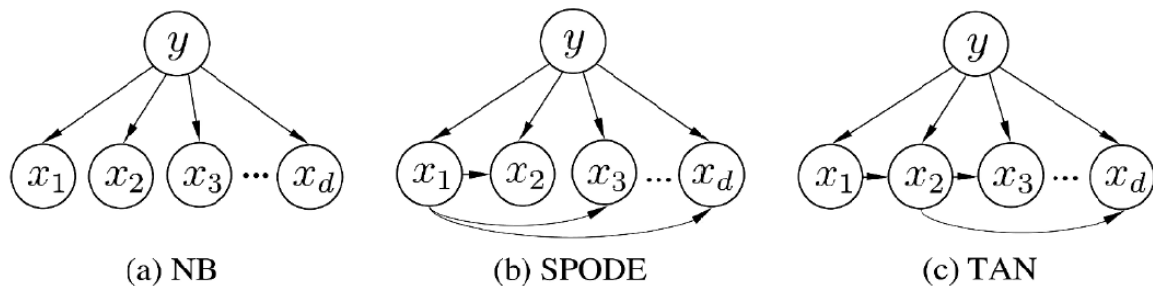


图 7.1 朴素贝叶斯与两种半朴素贝叶斯分类器所考虑的属性依赖关系

贝叶斯分类器

□ AODE (Averaged One-Dependent Estimator)

- ✓ 尝试将每个属性作为超父构建SPODE
- ✓ 将拥有足够训练数据支撑的SPODE集成作为最终结果

$$P(c | \mathbf{x}) \propto \sum_{\substack{i=1 \\ |D_{x_i}| \geq m'}}^d P(c, x_i) \prod_{j=1}^d P(x_j | c, x_i)$$

其中 D_{x_i} 是在第 i 个属性上取值为 x_i 的样本的集合, m' 为阈值常数

$$\hat{P}(c, x_i) = \frac{|D_{c, x_i}| + 1}{|D| + N \times N_i}, \quad \hat{P}(x_j | c, x_i) = \frac{|D_{c, x_i, x_j}| + 1}{|D_{c, x_i}| + N_j}$$

N 为 D 中可能的类别数, N_i 为第 i 个属性上可能的取值数

D_{c, x_i, x_j} 表示类别为 c 且在第 i 和第 j 个属性上取值分别为 x_i 和 x_j 的样本集合



Geoff Webb
澳大利亚Monash大学

贝叶斯分类器

□ 高阶依赖

✓ 能否通过考虑属性间的高阶依赖来进一步提升泛化性能？

◆ 例如最简单的做法：ODE \rightarrow kDE

- 将父属性 $P a_i$ 替换为包含k个属性的集合 $P a_i$

✓ 明显障碍：随着 k 的增加，估计 $P(x_i|y, P a_i)$ 所需的样本数将以指数级增加

◆ 训练样本非常充分 \rightarrow 性能可能提升

◆ 有限训练样本 \rightarrow 高阶联合概率估计困难

考虑属性间的高阶依赖，需要其他办法

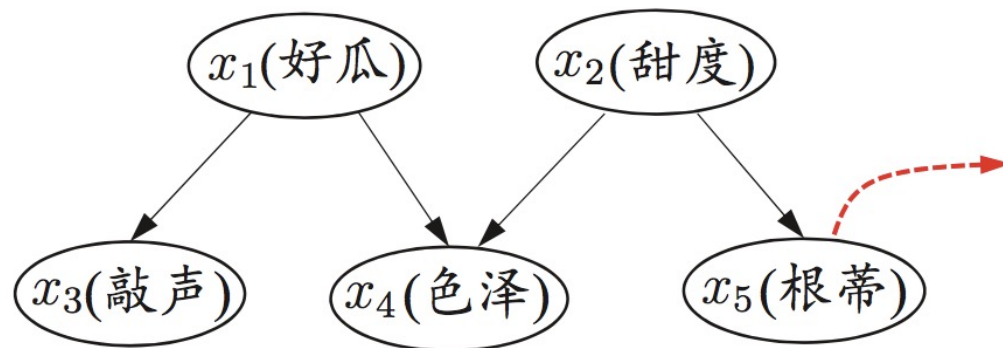
贝叶斯分类器

□ 贝叶斯网 (Bayesian network; Bayes network)

亦称 “信念网” (belief network)

有向无环图(DAG, Directed Acyclic Graph)

条件概率表 (CPT, Conditional Probability Table)



		根蒂	
		硬挺	蜷缩
甜度	高	0.1	0.9
	低	0.7	0.3

贝叶斯网 $B = \langle G, \theta \rangle$

结构 参数

1985年 J. Pearl 命名为贝叶斯网, 为了强调:

- 输入信息的主观本质
- 对贝叶斯条件的依赖性
- 因果与证据推理的区别

概率图模型(Probabilistic graphical model) → 第14章

- 有向图模型 → 贝叶斯网
- 无向图模型 → 马尔可夫网



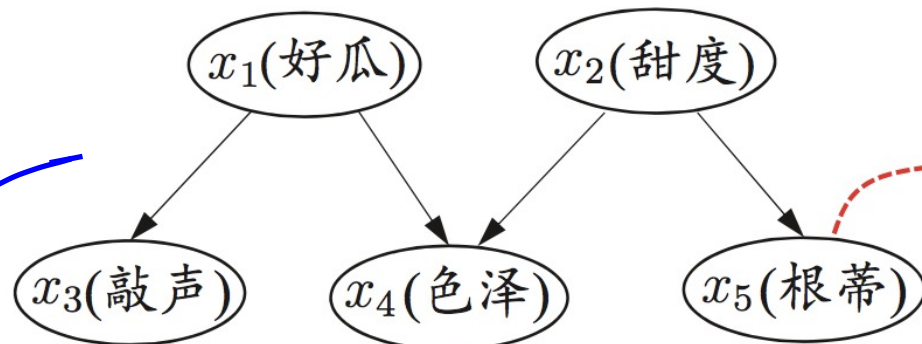
Judea Pearl
朱迪亚·珀尔(1936 -)
2011 图灵奖

贝叶斯分类器

□ 贝叶斯网 (Bayesian network; Bayes network)

有向无环图(DAG, Directed Acyclic Graph)

条件概率表 (CPT, Conditional Probability Table)



		根蒂	
		硬挺	蜷缩
甜度	高	0.1	0.9
	低	0.7	0.3

$$x_3 \perp x_4 \mid x_1$$

$$x_4 \perp x_5 \mid x_2$$

给定父结点集，贝叶斯网假设每个属性与其非后裔属性独立

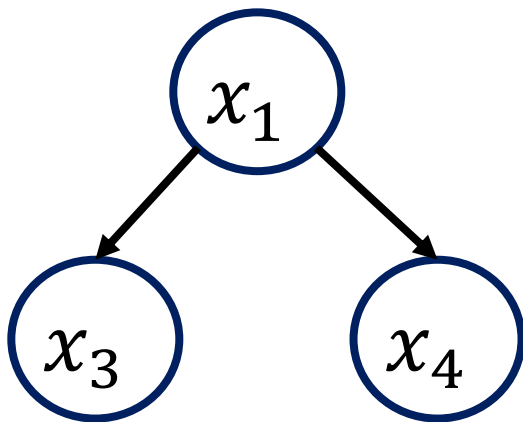
$$P_B(x_1, x_2, \dots, x_d) = \prod_{i=1}^d P_B(x_i \mid \pi_i) = \prod_{i=1}^d \theta_{x_i} \mid \pi_i$$

父结点集

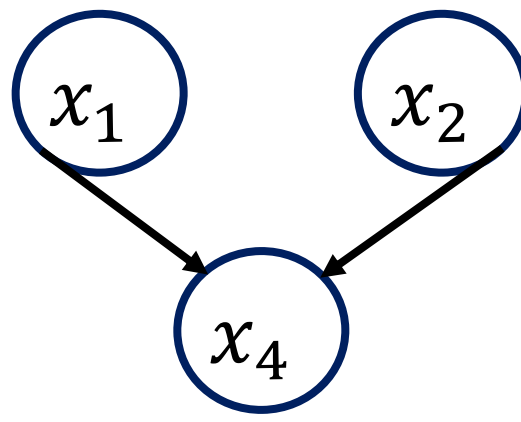
$$P(x_1, x_2, x_3, x_4, x_5) = P(x_1)P(x_2)P(x_3|x_1)P(x_4|x_1, x_2)P(x_5|x_2)$$

贝叶斯分类器

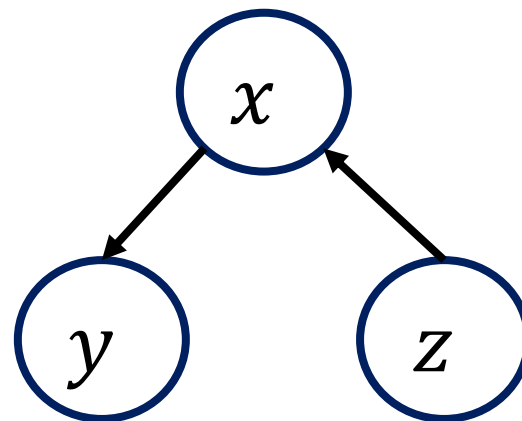
□ 三变量间的典型依赖关系



同父结构
条件独立性
 $x_3 \perp x_4 | x_1$



V型结构
边缘独立性
 $x_1 \perp\!\!\!\perp x_2$



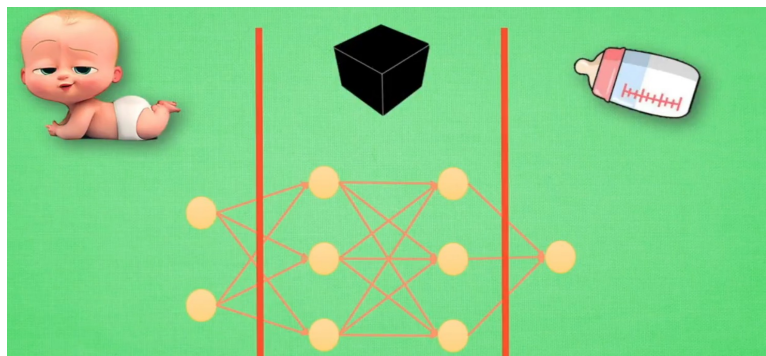
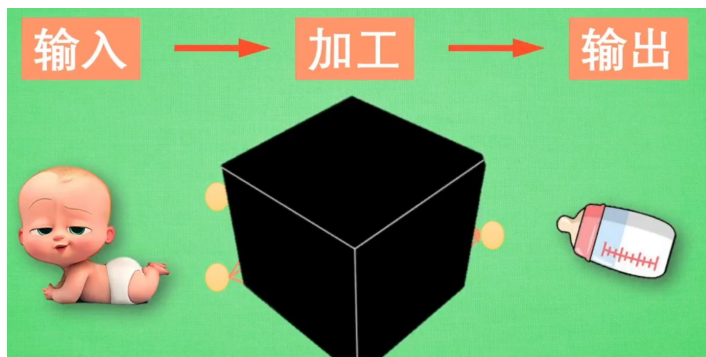
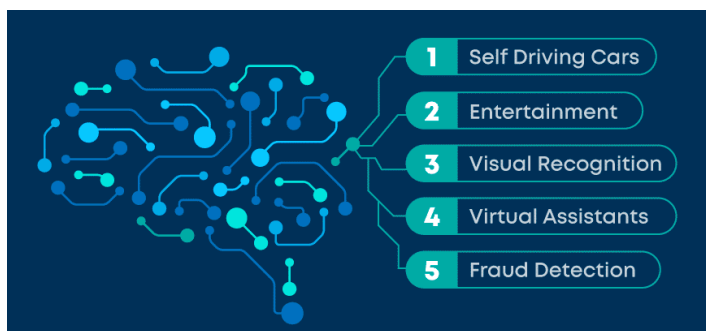
顺序结构
条件独立性
 $y \perp z | x$

- 若 x_4 已知, 则 x_1 与 x_2 不独立
- 若 x_4 未知, 则 x_1 与 x_2 独立

深度学习中的贝叶斯

□ 深度学习时代的贝叶斯方法

背景与动因



贝叶斯方法的优势

- ◆ 不确定性建模
- ◆ 正则化能力
- ◆ 小样本学习
- ◆ 模型可解释性

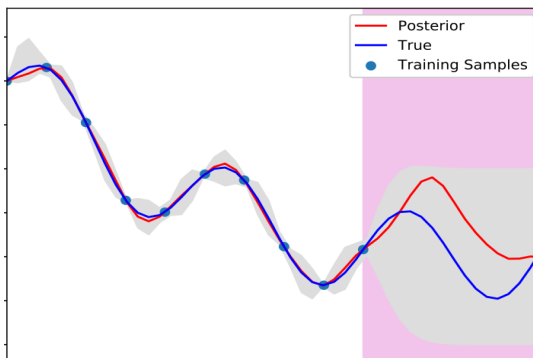
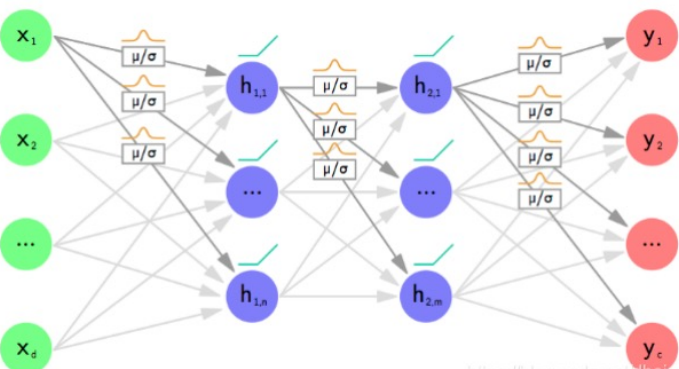
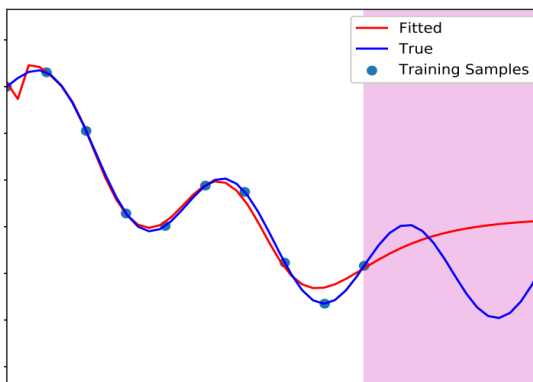
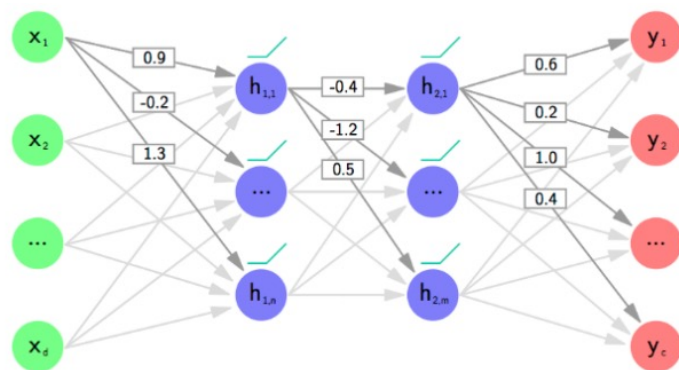
贝叶斯方法的应用

- ◆ 贝叶斯神经网络 (BNNS)
- ◆ 变分推断与MC Dropout
- ◆ 贝叶斯优化 (BO)

深度学习在多个领域取得了巨大的成功，但是传统的深度神经网络缺乏不确定性建模能力，容易过拟合，不懂装懂，难以解释。

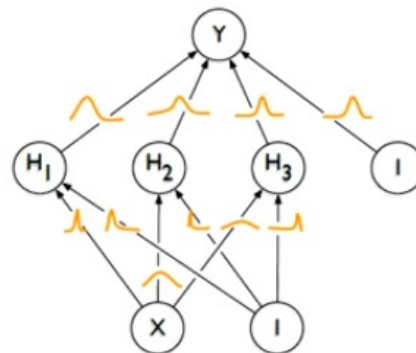
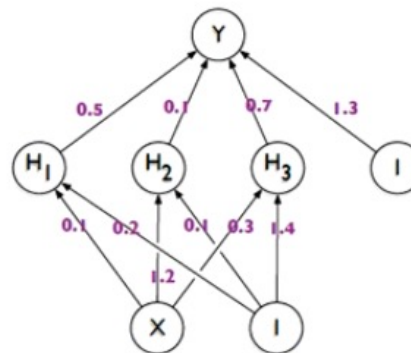
深度学习中的贝叶斯

□ 贝叶斯神经网络



频率理论 vs 贝叶斯理论

- ◆ 概率是一个确定的数
- ◆ 大数定律
- ◆ 没有先验概率
- ◆ 概率是一个分布
- ◆ 对可信度的衡量
- ◆ 有先验概率



传统神经网络（上）与贝叶斯神经网络（下）

[\[2006.12024\] Bayesian Neural Networks: An Introduction and Survey \(arxiv.org\)](#)

Case Studies in Applied Bayesian Data Science: CIRM Jean-Morlet Chair, Fall 2018 (书籍)

深度学习中的贝叶斯

□ 贝叶斯推断

定义：根据贝叶斯定理进行知识更新（使用贝叶斯定理推理数据的种群分布或概率分布的性质过程）

$$P(A|B) = \frac{P(B|A)P(A)}{P(B)}$$

似然函数 先验概率
后验概率 边缘似然（数据的整体概率）

核心步骤

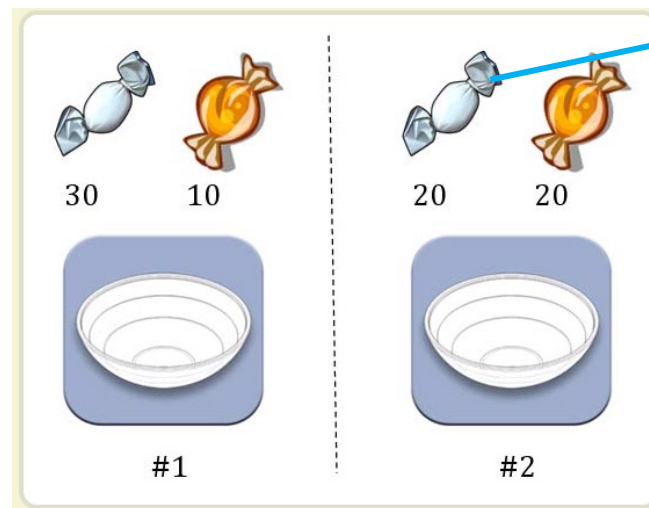
- ◆ 定义先验分布
- ◆ 构建似然函数
- ◆ 计算后验分布

常用的推断方法

- ◆ 马尔可夫链蒙特卡洛 (MCMC)
- ◆ 变分推断
- ◆ 拉普拉斯近似

贝叶斯推断的优势

- ◆ 不确定性量化，先验知识融合
- ◆ 在线学习能力（后验分布可作为新数据的先验）

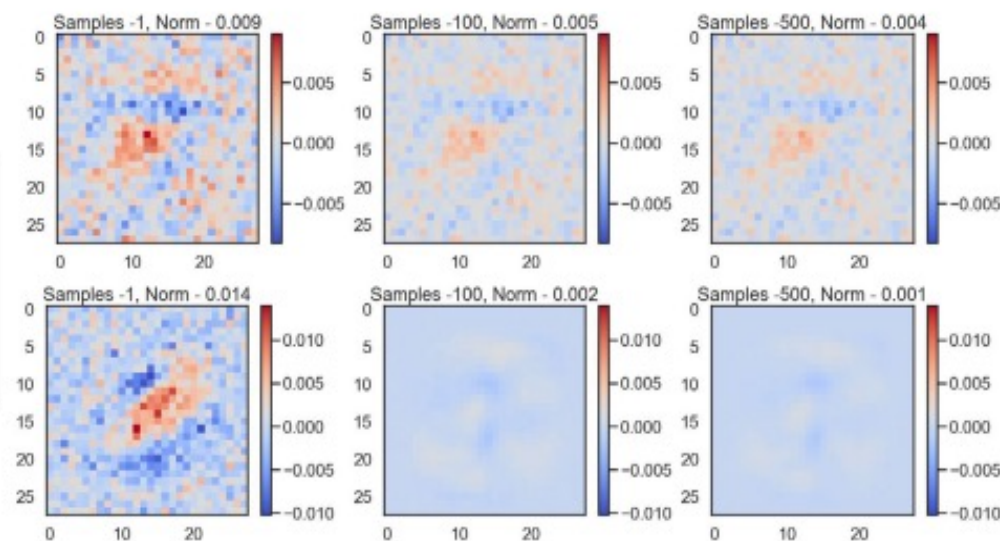
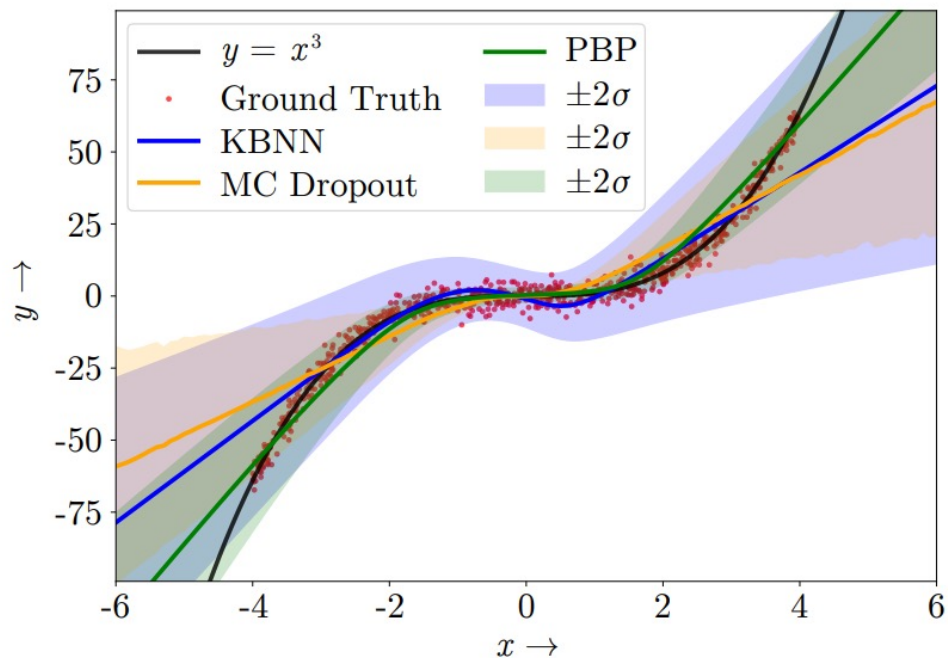


水果糖案例

随机选择一个碗，从中摸出一颗糖，发现是水果糖。请问这颗水果糖来自一号碗的概率有多大？

深度学习中的贝叶斯

□ 最新的应用

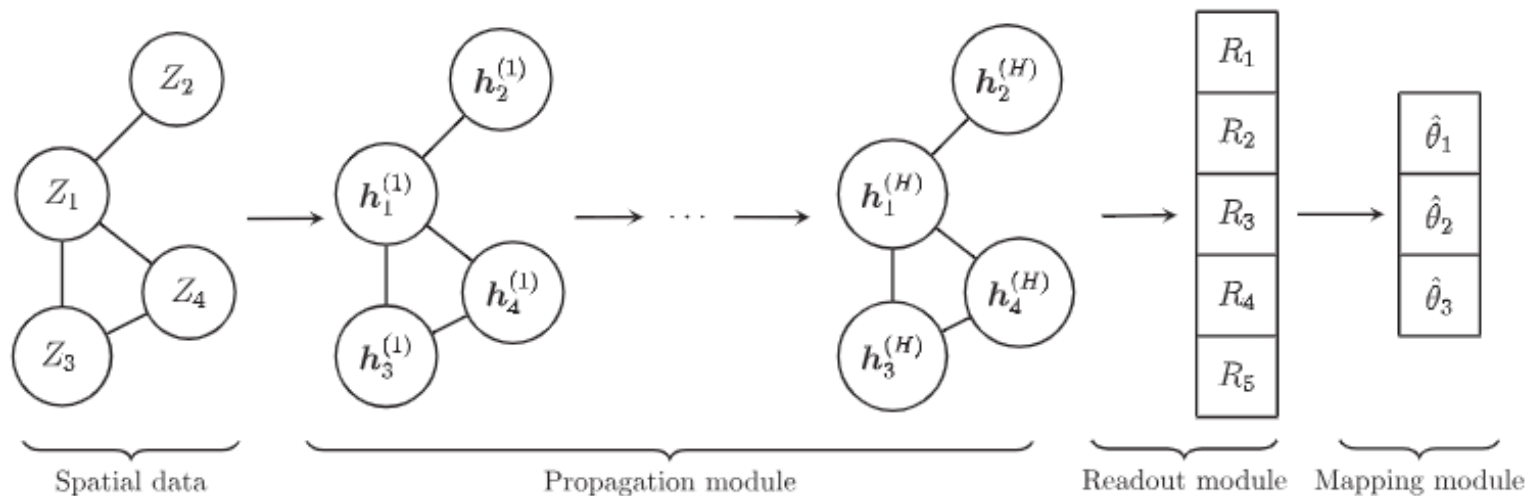


Kalman Bayesian Neural Networks for Closed-Form Online Learning. AAAI 2023 提出了一种通过闭式贝叶斯推理来训练网络权值的方法，无需反复遍历整个数据集。（图左为KBNN、MC Dropout和PBP方法在回归任务 $y = x^3 + \epsilon$ 上训练一个周期后的预测结果）

On the Robustness of Bayesian Neural Networks to Adversarial Attacks. TNNLS2024 证明了BNNs在保持干净数据上高准确率的同时，也能对基于梯度和非梯度的方法的对抗攻击表现出较强的鲁棒性。（图右为训练的贝叶斯神经网络（BNNs）的期望损失梯度的输入梯度图）

深度学习中的贝叶斯

□ 最新的应用



Bayesian deep neural networks for spatio-temporal probabilistic optimal power flow with multi-source renewable energy 2024AE 使用BNNS解决了POPF 的实际应用的输入场景不准确和计算开销巨大问题。（上图为基于GNN的针对单个空间场的BNNS架构图）

贝叶斯分类器实践内容

作业要求：参考“**贝叶斯实践内容-自主练习**”完成**思考题目**

提交要求：不作为小作业，自行练习为主

负责助教：李兵

答疑邮箱：5222023330043@smail.nju.edu

提交邮箱：~~nju_ml@163.com~~

提交时间：~~2025年*月**日晚24:00~~

谢谢！

联系方式： liwenbin@nju.edu.cn

更多信息： www.liwenbin.cn

HuggingFace介绍

目录

□ **前言**

□ **Models**

□ **Datasets**

□ **演示**

目录

□ 前言

□ Models

□ Datasets

□ 演示

前言

□ 前言

- 模型的网络结构之间的区别，训练技巧之间的区别，损失函数之间的区别。
- 当前的模型主要比拼的是数据量以及参数量。
- 刷屏的模型，主流模型拥有极其恐怖量的参数。
- 我们能做的，训练模型？海量的数据？

□ 问题

- 想要快速上手，是否需要深入学习传统等算法？
- 现在的公开课和教材太多，难以快速上手。
- 如今的难题渐渐不再依赖于传统方法，一些交给transformer是完全足够的。
- 模型的领域有无数个算法和模型，我们是否需要一个一个来进行学习来进行实验？

前言

□ Huggingface是什么

- 一个库，包含了基本所有的NLP领域的核心模型（BERT，GPT等等）。
- 只需几行代码就能调用当前的主流预训练模型。
- 微调自己的模型只需要处理好自己的数据，继续训练模型即可。
- 即使不擅长数学，代码，数据也能高效使用模型。
- NLP领域大佬的舞台，通过其开源模型，宣传论文和研究成果。

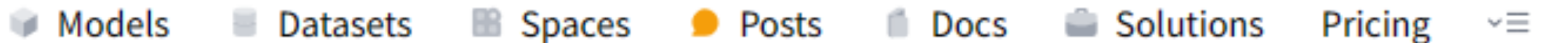


HUGGING FACE

前言

□ Huggingface组成

- **Models**: 提供预训练的模型, 如BERT, GPT, T5等。
- **Datasets**: 提供成千上万的NLP, CV以及其它领域的数据集, 用户也可以上传自己的数据集。
- **Spaces**: 用户可以使用Gradio以及Streamlit搭建机器学习模型的在线演示, 方便分享互动模型以及可视化成果。
- **Posts**: Hugging Face 社区的博客和技术文章部分, 用户可以查看最新的技术教程、项目介绍及社区新闻。
- **Docs**: 提供 Hugging Face 各个库 (如 Transformers、Datasets、Diffusers 等) 的使用文档, 帮助用户快速上手和掌握相关工具。
- **Solutions**: 介绍 Hugging Face 针对企业的解决方案, 包括 API、云服务以及专用的模型部署方案, 满足商业场景的需求。
- **pricing**: 展示平台的各项服务收费标准, 包括免费版和企业版的功能对比, 帮助用户根据需求选择合适的计划。



目录

□ 前言

□ **Models**

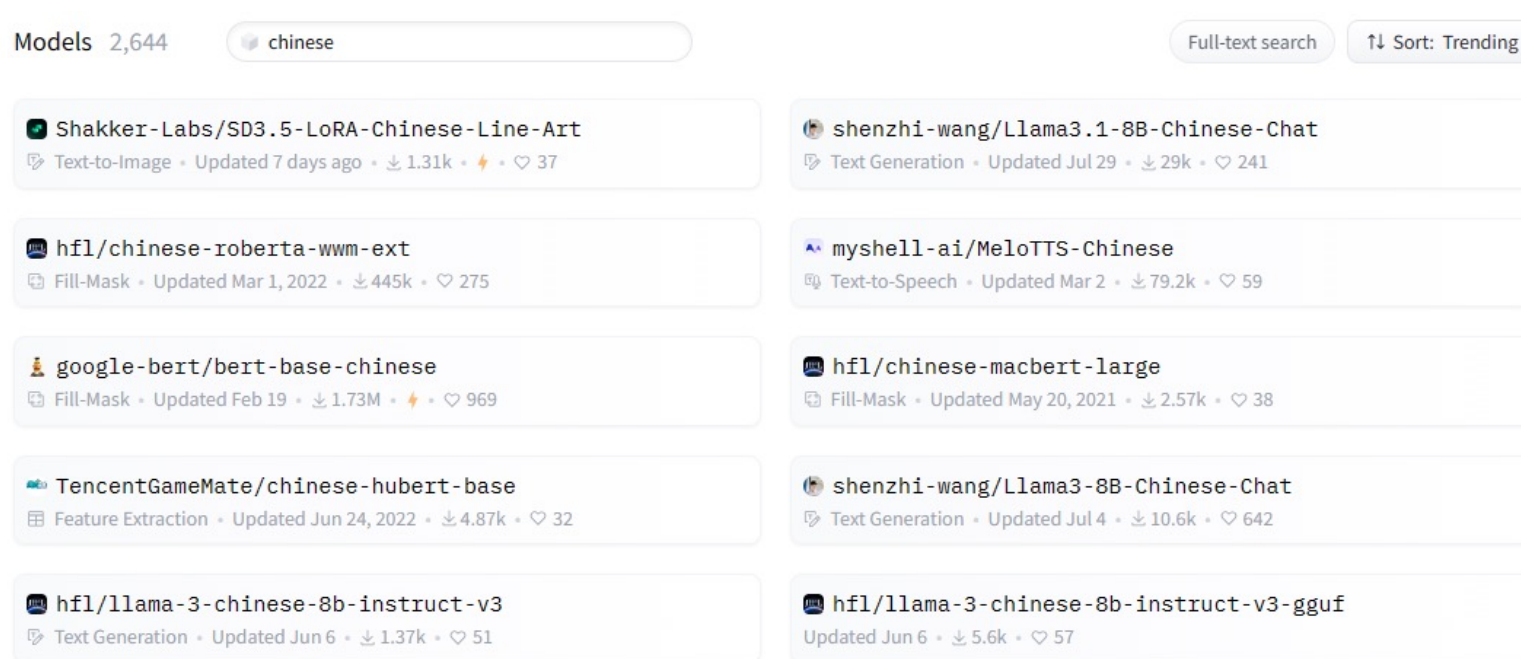
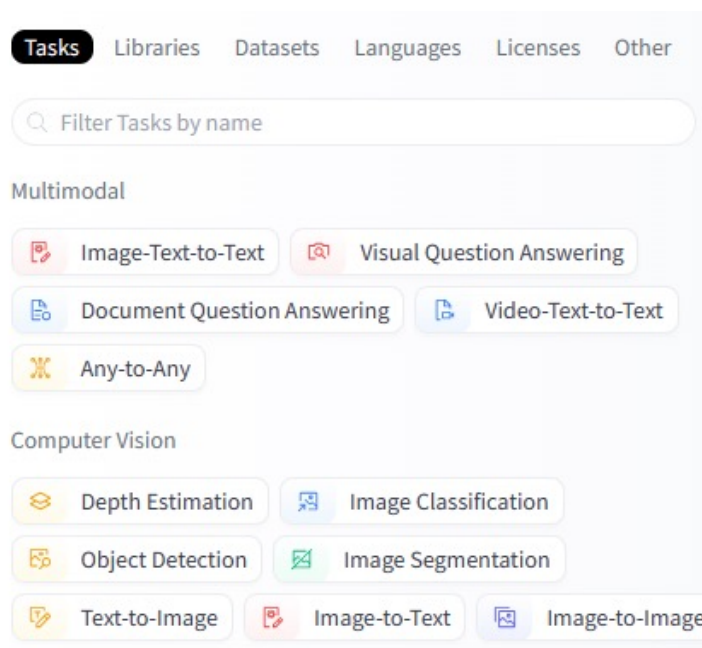
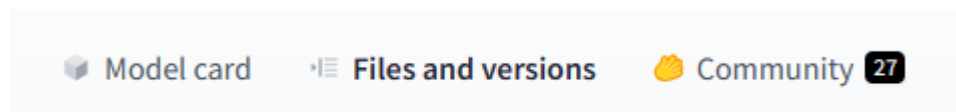
□ **Datasets**

□ 演示

Models

□ Models

- 通过关键字查找自己需要的模型



Models

□ Models

- Model card (怎么去使用这个模型)

≡ Bert-base-chinese

Table of Contents

- [Model Details](#)
- [Uses](#)
- [Risks, Limitations and Biases](#)
- [Training](#)
- [Evaluation](#)
- [How to Get Started With the Model](#)

Training

Training Procedure

- `type_vocab_size: 2`
- `vocab_size: 21128`
- `num_hidden_layers: 12`

Training Data

[More Information Needed]





































How to Get Started With the Model

```
from transformers import AutoTokenizer  
  
tokenizer = AutoTokenizer.from_pretrained('bert-base-chinese')  
  
model = AutoModelForMaskedLM.from_pretrained('bert-base-chinese')
```

Models

□ Models

- Files and versions (权重数据, 参数文件, 版本信息)

 .gitattributes		445 Bytes	
 README.md		1.85 kB	
 config.json		624 Bytes	
 flax_model.msgpack		409 MB 	
 model.safetensors	 	412 MB 	
 pytorch_model.bin	 	412 MB 	
 tf_model.h5		478 MB 	
 tokenizer.json		269 kB	
 tokenizer_config.json		49 Bytes	
 vocab.txt		110 kB	

目录

□ 前言

□ Models

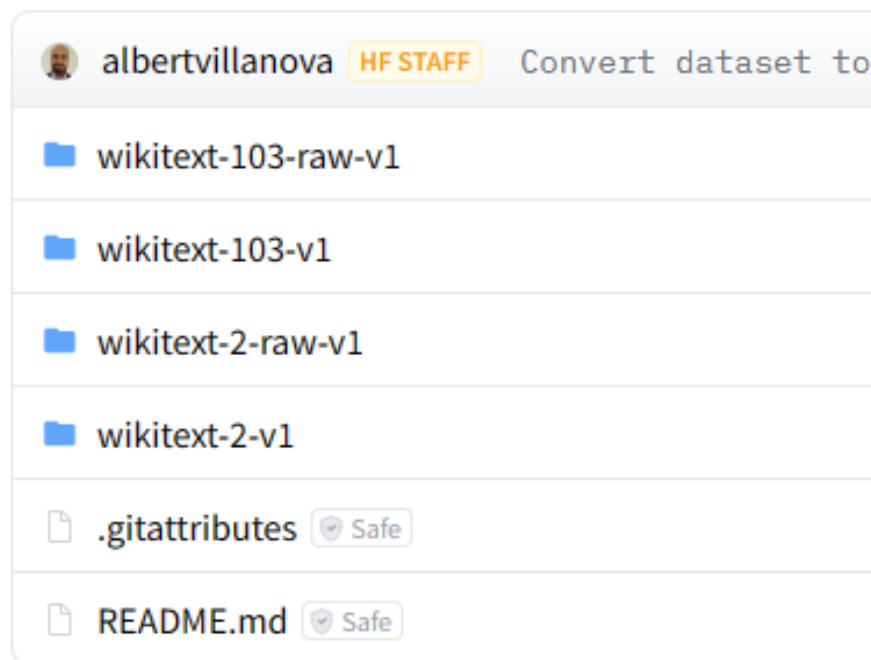
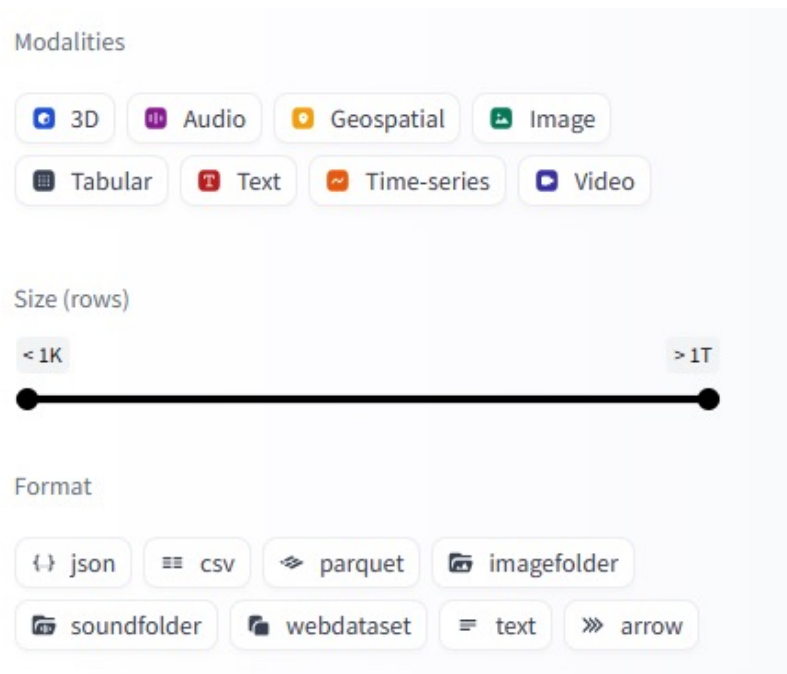
□ **Datasets**

□ 演示

Datasets

□ Datasets

- 通过关键字查找自己需要的数据



目录

□ 前言

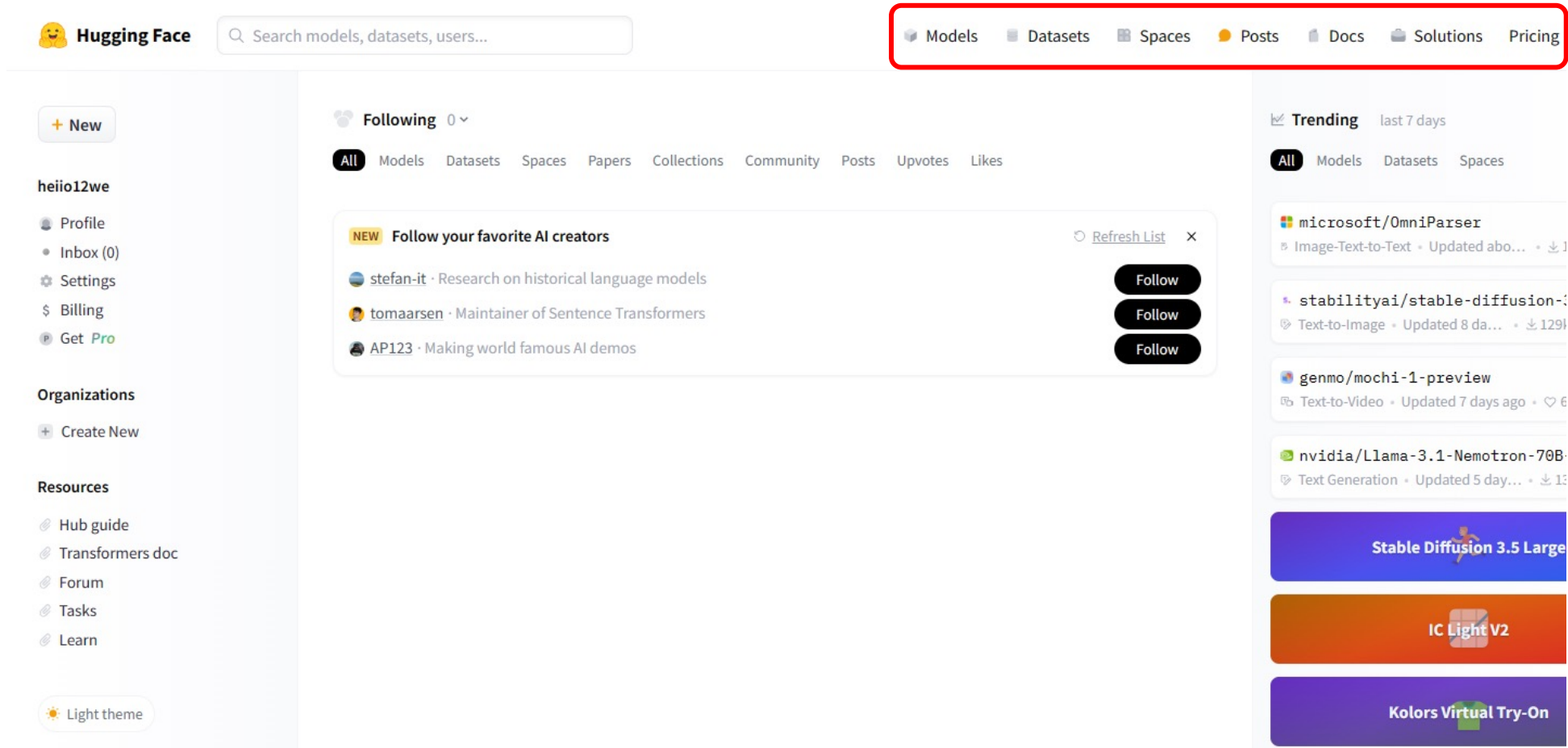
□ Models

□ Datasets

□ 演示

演示

□ Huggingface界面



演示

□ 演示（句子编码）

- 环境（python pytorch transformers datasets）

```
# 编码句子
from transformers import BertTokenizer

tokenizer = BertTokenizer.from_pretrained(
    pretrained_model_name_or_path = 'bert-base-chinese',
    cache_dir = None,
    force_download = False,
)
sents = [
    '选择珠江花园的原因就是方便。',
    '笔记本的键盘就是爽。',
    '房间太小，其他的都一般',
    '今天才知道这本书还有第六卷，真有带你郁闷',
    '机器背面似乎被撕了张什么标签，残胶还在。'
]
```

```
out = tokenizer.encode_plus(
    text = sents[0],
    text_pair = sents[1],

    truncation = True,
    padding = 'max_length',
    add_special_token = True,
    max_length = 30,
    return_tensor = None,

    return_token_type_ids = True, #第一个句子和特殊符号的位置是0，第二个句子的位置是1
    return_attention_mask = True,
    return_special_tokens_mask = True,
)

for k,v in out.items():
    print(k,':',v)

tokenizer.decode(out['input_ids'])
```

演示

□ 演示 (句子编码)

- 环境 (python pytorch transformers datasets)

```
warnings.warn(
Keyword arguments {'add_special_token': True, 'return_tensor': None} not recognized.
Keyword arguments {'add_special_token': True, 'return_tensor': None} not recognized.
[101, 6848, 2885, 4403, 3736, 5709, 1736, 4638, 1333, 1728, 2218, 3221, 3175, 912, 511, 102, 5011, 6381, 3315, 4638, 7241, 4669, 2218, 3221, 4272, 511, 102, 0, 0, 0]
[CLS] 选择珠江花园的原因就是方便。 [SEP] 笔记本的键盘就是爽。 [SEP] [PAD] [PAD] [PAD]
```

句子开始

内容

新句子开始

填充

演示

□ 演示（数据集操作）

- 数据集下载，排序，打乱顺序，选择，切分，分桶
- Sort shuffle select split shard

```
print("数据集")
print(datasets)
print(datasets[1])

# 数据集排序
sort_dataset = datasets.sort('label')
print("数据集排序")
print(sort_dataset['label'][:10])
print(sort_dataset['label'][-10:])

#数据集打乱顺序
shuffled_dataset= sort_dataset.shuffle(seed=42)
print("打乱顺序")
print(shuffled_dataset['label'][:10])

#选择
datasets.select([0,10,20,30,40,50])
print("选择")
print(datasets.select([0,10,20,30,40,50]))

#切分和分桶
datasets.train_test_split(test_size = 0.1)
print("切分")
print(datasets.train_test_split(test_size = 0.1))
datasets.shard(num_shards=4,index=0)
print("分桶")
print(datasets.shard(num_shards=4,index=0))
```

```
数据集
Dataset({
  features: ['label', 'text'],
  num_rows: 9600
})
{'label': 1, 'text': '15.4寸笔记本的键盘确实爽，基本跟台式机差不多了，蛮喜欢数字小键盘，'}
数据集排序
[0, 0, 0, 0, 0, 0, 0, 0, 0, 0]
[1, 1, 1, 1, 1, 1, 1, 1, 1, 1]
打乱顺序
[0, 1, 0, 0, 1, 0, 1, 0, 1, 0]
选择
Dataset({
  features: ['label', 'text'],
  num_rows: 6
})
切分
DatasetDict({
  train: Dataset({
    features: ['label', 'text'],
    num_rows: 8640
  })
  test: Dataset({
    features: ['label', 'text'],
    num_rows: 960
  })
})
分桶
Dataset({
  features: ['label', 'text'],
  num_rows: 2400
})
```

演示

□ 演示（模型下载使用）

```
1 from transformers import AutoTokenizer, AutoModelForMaskedLM
2 import torch
3
4
5 tokenizer = AutoTokenizer.from_pretrained("bert")
6 model = AutoModelForMaskedLM.from_pretrained("bert")
7
8 input_text = '江苏省是[MASK]国的一个省'
9 inputs = tokenizer(input_text, return_tensors='pt')
10
11
12 with torch.no_grad():
13     outputs = model(**inputs)
14
15 logits = outputs.logits
16 masked_index = (inputs['input_ids'] == tokenizer.mask_token_id).nonzero(as_tuple=True)[1]
17 masked_logits = logits[0, masked_index]
18 predicted_index = masked_logits.argmax(dim=-1)
19 predicted_token = tokenizer.decode(predicted_index)
20
21 print(predicted_token)
22
```

- This IS expected if you are initializing from a BertForPreTraining model).
- This IS NOT expected if you are initializing orSequenceClassification model).
中

谢谢!