stokhēmac

xq

Contents

Introduction

Math

Probability

construction of a probability space

Probability distributions are used as a building block for:
- probabilistic modeling
- graphical models
- model selection

To tackle in statistical modeling and machine learning, one need to check the property of probability distributions: whether to model categorical (a discrete random variable) or continuous (a continuous random variable).

philosophical basis

probability theory can be considered a generalization of Boolean logic.

In the context of machine learning, it is often applied in this way to formalize the design of automated reasoning systems.

The philosophical basis of probability and how it should be somehow related to what we think should be true (in the logical sense) was studied by Cox (Jaynes, 2003). Another way to think about it is that if we are precise about our common sense we end up constructing probabilities. E. T. Jaynes (1922–1998) identified three mathematical criteria, which must apply to all plausibilities:

1. The degrees of plausibility are represented by real numbers.
2. These numbers must be based on the rules of common sense.
3. The resulting reasoning must be consistent, with the following three meanings of the word “consistent”:
   - Consistency or non-contradiction: When the same result can be reached through different means, the same plausibility value must be found in all cases.
   - Honesty: All available data must be taken into account.
   - Reproducibility: If our state of knowledge about two problems is the same, then we must assign the same degree of plausibility to both of them.

two major probabilities

Remark.

In machine learning and statistics, there are two major interpretations of probability:

- **Frequentist interpretation**: considers the relative frequencies of events of interest to the total number of events that occurred. The probability of an event is defined as the relative frequency of the event in the limit when one has infinite data.
- **Bayesian interpretation**: uses probability to specify the *degree of uncertainty* that the user has about an event. It is sometimes referred to as “subjective probability” or “degree of belief”.

probability and random variables

concepts of sample space, event space, and probability

There are three distinct ideas that are often confused when discussing probabilities. First is the idea of a probability space, which allows us to quantify the idea of a probability. However, we mostly do not work directly with this basic probability space. Instead, we work with random variables (the second idea), which transfers the probability to a more convenient (often numerical) space. The third idea is the idea of a distribution or law associated with a random variable. We will introduce the first two ideas in this section and expand on the third idea in Section 6.2.

The probability space models a real-world process (referred to as an experiment) with random outcomes.

Probability space:

1. sample space Ω: set of all possible outcomes.
2. event space $\mathcal{A}(\mathcal{F})$: set of which elements (called events) are subsets of Ω in other word, element of $\mathcal{A}$ is* event, which is a collection of outcomes from Ω. Events of $\mathcal{A}$ are allowed to assign probabilities to.
   power set of Ω: $\mathcal{A} = \mathcal{P}(\Omega)$
3. probability $\mathcal{P}$: is a number measure the degree of belief that an event will occur. Given $A \in \mathcal{A}$, $P(A)$ is called the probability of A

Stochastic

coupling

Remark: probability space A probability space is a triple $(E, \mathcal{E}, \mathbb{P})$, that $(E, \mathcal{E})$ is a *measurable space* consisting of:

1. E: a sample space and is a set
2. $\mathcal{E}$: a σ-algebra and is a subsets of E
3. $\mathbb{P}$: a probability measure on $\mathcal{E}$

Typically, E os a Polish space (i.e., complete, separable, and metric) and $\mathcal{E}$ consists of its Borel sets.

Consider two probability measures $\mathbb{P}$ and $\mathbb{P}'$ on the same measueable space $(E, \mathcal{E})$. A product measurable space is defined as: $(E \times E, \mathcal{E} \otimes \mathcal{E})$. $\mathcal{E} \otimes \mathcal{E}$ is the smallest σ-algebra containing $\mathcal{E} \times \mathcal{E}$. If $E = \mathbb{R}$, then $\mathcal{E} = \mathcal{B}(\mathbb{R})$ the Borel σ-algebra.

Given two probability measures $\mathbb{P}$ and $\mathbb{Q}$ on corresponding measurable spaces $(P, \mathcal{P})$ and $(Q, \mathcal{Q})$, the *product measurable space* is defined as $(P \times Q, \mathcal{P} \otimes \mathcal{Q})$,where their the Cartesian product set is:

$$P \times Q := \{(p, q) : p \in P, q \in Q\}$$

where

- (p, q) is a pair of points.
- $P \times Q$ is the set of points, i.e., the *plane* of all coordinates (p, q).
- $\mathcal{P} \otimes \mathcal{Q}$ is the smallest σ-algebra on $P \times Q$, so that all "regions" in the plane are measurable.

The σ-algebra $\mathcal{P} \otimes \mathcal{Q}$ contains all measurable rectangles $A \times B$ for $A \in \mathcal{P}$ and $B \in \mathcal{Q}$. That is, if a set A and a set B are measurable, then $A \times B$ is measurable in the product space[1].

If $P = \mathbb{R}$ and $Q = \mathbb{R}$, then

$$P \times Q = \mathbb{R}^2, \quad \mathcal{P} \otimes \mathcal{Q} = \mathcal{B}(\mathbb{R}^2),$$

where $\mathcal{B}(\mathbb{R}^2)$ is the Borel σ-algebra on the plane.

Machine

Deep Learning

perceptron

Perception: the process through which a system interprets and understands sensory data

Perception prefers to the process by which an artificial system interprets, transforms and understands data—such as images, sounds, or text—by recognizing patterns and extracting meaningful information; similar to how humans perceive the world through their senses. This concept is foundational in computer vision, natural language processing, and speech recognition, where machines learn to recognize patterns and make sense of complex real-world data. The goal of perception in machine learning is to enable systems to identify and categorize objects, comprehend spoken words, or interpret written text, thereby bridging the gap between raw data and intelligent decision-making.

Perception relies heavily on algorithms that simulate aspects of human cognition, such as the ability to recognize patterns, infer meaning, and adapt to new information. One of the earliest and most influential models inspired by biological perception is the perceptron, a type of artificial neuron developed in the 1950s by Frank Rosenblatt. The perceptron takes multiple inputs, assigns weights to them, sums them up, and passes the result through an activation function to produce an output. This simple mechanism laid the foundation for more advanced neural network architectures used today. In modern machine learning, especially deep learning, layers of artificial neurons are stacked together to form deep neural networks capable of learning highly complex representations of data.

$$\hat{y} = g(w_0 + \boldsymbol{X}^T \boldsymbol{W})$$

$$\mathbf{X} = \begin{bmatrix} x_1 \\ \vdots \\ x_m \end{bmatrix} \quad \boldsymbol{W} = \begin{bmatrix} w_1 \\ \vdots \\ w_m \end{bmatrix}$$

loss function

Loss function: measures the error between a machine learning model's predictions and true values, guiding parameter optimization.

In machine learning, a **loss function** quantifies the difference between a model's predictions and the actual target values, serving as a critical component in training algorithms. During the training process, the model adjusts its parameters (e.g., weights in a neural network) to minimize this loss, thereby improving its predictive accuracy. The loss function provides a numerical measure of how far off the model's predictions are from the ground truth, guiding the optimization process, typically via techniques like gradient descent. For example, in regression tasks, a common loss function is the **Mean Squared Error (MSE)**, which calculates the average squared difference between predicted and actual values. In classification tasks, **Cross-Entropy Loss** is often used to measure the divergence between predicted probabilities and true class labels. The choice of loss function depends on the task, data type, and desired model behavior, making it a cornerstone of model performance.

Different loss functions are tailored to specific machine learning problems. For regression, MSE penalizes larger errors more heavily due to the squaring of differences, encouraging the model to make predictions close to the actual values. In contrast,Mean Absolute Error (MAE) treats all errors equally, which can be more robust to outliers. For classification, Binary Cross-Entropy or Categorical Cross-Entropy measures the dissimilarity between predicted probabilities and true labels, promoting confident and accurate classifications. In bioinformatics, where data like sequencing or omics datasets can be noisy or imbalanced, specialized loss functions like Focal Loss be used to focus on hard-to-classify examples or handle class imbalance. The loss function directly influences the model's learning dynamics, as it defines the objective that the optimization algorithm seeks to minimize, impacting convergence speed and model generalization.

Why the predicted is so different with actual values?

A **loss function** quantifies the error between a machine learning model's predictions and actual target values, guiding optimization. It measures model performance, enabling parameter adjustments to minimize errors during training, critical for tasks like regression or classification in computational biology.

- **Mean Squared Error (MSE)**: Averages squared differences between predictions and true values, emphasizing larger errors.
- **Mean Absolute Error (MAE)**: Averages absolute differences, robust to outliers in regression tasks.
- **Cross-Entropy Loss**: Measures divergence between predicted probabilities and true labels, common in classification.
- **Focal Loss**: Focuses on hard-to-classify examples, useful for imbalanced datasets like omics data.

`loss function`: measures the cost incurred from incorrect predictions. In other word, how far apart from the real value/ground true obversation with the predicted value.

$$\mathcal{L}(f(x^{(i)};\boldsymbol{W}), y^{(i)})$$

where $f(x^{(i)};\boldsymbol{W})$: predicted value
$y^{(i)}$: actual value
There are some common loss function:
`empirical loss`: measures the total loss over enture dataset. also called: objective / cost funtion / empirical loss

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n} f(x^{(i)};\boldsymbol{W}, y^{(i)})$$

`binary cross entropy loss`: specific for binary output (probability between 0 and 1)

$$J(\boldsymbol{W}) = -\frac{1}{n}\sum_{i=1}^{n} y^{(i)}\log(f(x^{(i)};\boldsymbol{W}) + (1-y^{(i)})\log(1-f(x^{(i)};\boldsymbol{W}))$$

`mean square error loss`: specific for continous real number (regression model)

$$J(\boldsymbol{W}) = \frac{1}{n}\sum_{i=1}^{n}(y^{(i)} - f(x^{(i)};\boldsymbol{W}))^2$$

Selecting an appropriate loss function is crucial for aligning the model's objectives with the problem's requirements. A poorly chosen loss function can lead to suboptimal performance, such as overfitting to outliers or failing to capture class imbalances in datasets like those encountered in computational biology. For instance, in genomic sequence classification, a loss function that accounts for imbalanced classes can improve the detection of rare variants. Beyond standard loss functions, custom loss functions can be designed in Python or R to incorporate domain-specific knowledge, such as penalizing biologically implausible predictions in omics data analysis. By carefully defining the loss, researchers can steer the model toward meaningful solutions, ensuring robust performance in real-world applications.

loss optimization

training neural networks: fine the network weights to achieve the lowest loss

$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} \frac{1}{n}\sum_{i=1}^{n} \mathcal{L}(f(x^{(i)};\boldsymbol{W}), y^{(i)})$$
$$\boldsymbol{W}^* = \arg\min_{\boldsymbol{W}} J(\boldsymbol{W})$$

Loss is the a function of network weights

gradient descent

1. initialize weights randomly $\sim \mathcal{N}(0,\mu^2)$
2. loop until convergence:
3. compute gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. update $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta\,\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. return weights

```
import tensorflow as tf
weights = tf.Variable([tf.random.normal()])

white True:
    with tf.GradientTape() as g:
        loss = compute_loss(weights)
        # computing gradient
        gradient = g.gradient(loss, weights)

    weights = weights - lr * gradient
```

computing gradients: backpropagation

optimization

optimization: $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta\,\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$

η : setting a $learningrate$

small η : converges slowly and gets stuck in false local minima

large η : overshoot. become ubstable and diverge

stable η coverge smoothly and avoid local minima

- mini-batches

$\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$: very computationally intensive

□ $\frac{\partial J_i(\boldsymbol{W})}{\partial \boldsymbol{W}}$: easy to compute but very noisy (stochastic)

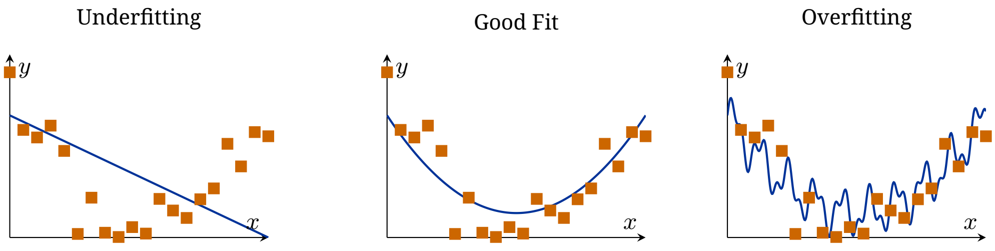
□ $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$: fast to compute and a much better estimate of the true gradient

stochastic gradient descent:

1. initialize weights randomly $\sim \mathcal{N}(0,\mu^2)$
2. loop until convergence:
3. compute gradient $\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}} = \frac{1}{B}\sum_{k=1}^{B} \frac{\partial J_k(\boldsymbol{W})}{\partial \boldsymbol{W}}$
4. update $\boldsymbol{W} \leftarrow \boldsymbol{W} - \eta\,\frac{\partial J(\boldsymbol{W})}{\partial \boldsymbol{W}}$
5. return weights

mini-batches while training:
- more accurate estimation of gradient
smoother convergence
allow for larger learning rates
- lead to fast training
can parallelize computation
achieve significant speed increases on GPU's

Overfitting



Bibliography

1. Brémaud, P. The Coupling Method 397–415. ISBN: 978-3-319-43476-6. https://doi.org/10.1007/978-3-319-43476-6_16 (Springer International Publishing, Cham, 2017).