COMP 480/580 — Probabilistic Algorithms and Data Structure **Oct 24, 2024**

Lecture 16

Lecturer: Anshumali Shrivastava *Scribe By: Vaibav Malpeddi (vm56), Prashithaa Balaji (pb55)*

1 What is Sampling?

Sampling is a statistical technique where a subset of data is selected from a larger population or dataset. Sampling enables the estimation of population characteristics or parameters when it's impractical to measure or analyze the entire population. By studying a sample, we can draw inferences about the larger group, reducing the cost, time, and effort required for data analysis.

There are two methods which are:

- Transformation-Based Sampling
- Rejection Sampling

1.1 Transformation-Based Sampling

This method uses mathematical transformations to convert samples from a simpler distribution (e.g., uniform) into samples from a target distribution. This approach includes techniques like the inverse transform method, which generates samples by applying the inverse of a cumulative distribution function (CDF) to uniform samples.

1.2 Rejection Sampling

This technique is used when we have a target distribution that is hard to sample from directly. It relies on a proposal distribution that "envelopes" the target distribution, using random samples from the proposal and accepting or rejecting them based on a condition. This method is especially useful in Monte Carlo methods and situations requiring approximate sampling.

2 Anomaly Detection

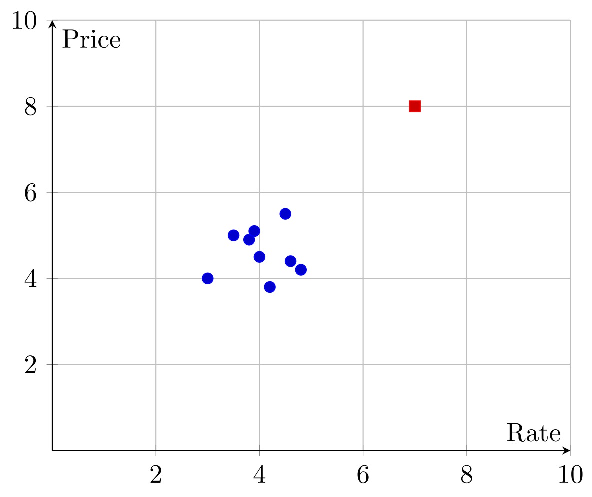
Anomaly detection is the process of examining specific data points and detecting rare occurrences that seem suspicious because they're different from the established pattern of behaviors. Anomaly detection identifies rare events or outliers by measuring deviation from typical patterns. Kernel Density Estimation (KDE) often aids in computing anomaly scores.

2.1 Kernel Density Estimation (KDE)

Kernel Density Estimation (KDE) is used to estimate the probability density function of a random variable. KDE helps identify anomalies by modeling the underlying distribution of data and determining how likely a new observation is, given this estimated distribution.

2.2 Example: Two-Dimensional Phases

Let's consider an example in a two-dimensional space in a game format, where data is continuously visualized. Imagine this space represents, for instance, the price of a house and the number of rooms it has. Typically, values fall within a certain range in this area, but then you notice an outlier—a point with a high number of rooms that doesn't align with the typical price patterns.



Red - represents the anomaly

In anomaly detection, we are observing something surprising enough—something abnormal that may indicate an attack.

Idea We have data in an x-y plane with two features, f_1 and f_2, in a two-dimensional space. For example, this could represent the relationship between the price of a house and the number of rooms. If we observe a point outside the typical range on this plane, we need to double-check it. Can we detect this point? In many cases, we don't get to see the geometric picture directly but still aim to detect whether there is an anomaly or not.

$$\underbrace{\text{Anomaly}(q) = \sum_{i=1}^{n} \text{Sim}(x_i, q)}_{\text{Similarity of query q with every element}} \tag{1}$$

If q is less than discriminator, then flag is raised.

2.3 Anomaly Detection for Networks

Anomaly detection is essential for identifying unusual patterns, such as Distributed Denial of Service (DDoS) attacks, which can disrupt network services. This method aims to improve the efficiency of detecting network anomalies through random sampling, with the condition that as long as links remain within a threshold, the method should be effective.

To increase efficiency, we use a random sampling method. As long as it reaches the threshold, the detection is considered efficient or effective. Locality Sensitive Hashing (LSH) can also be viewed as an efficient approach.

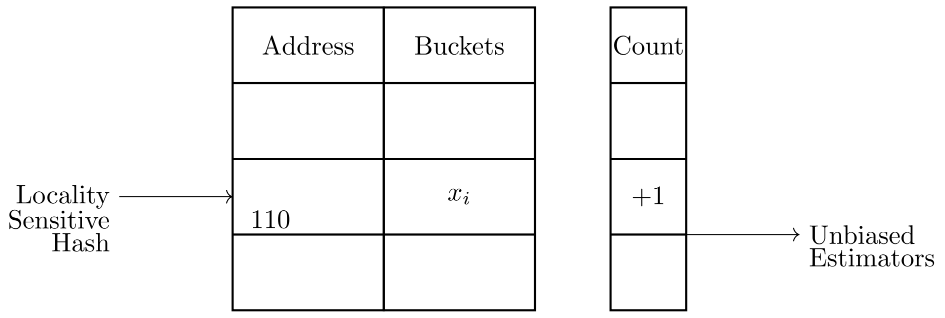


Figure 1: Diagram representing Locality Sensitive Hashing and unbiased estimators.

2.3.1 Locality Sensitive Hashing (LSH) in Anomaly Detection

LSH can be applied by ensuring that the probability of similar items hashing to the same bucket is proportional to their similarity. Specifically, using Jaccard similarity:

$$\Pr(h(x) = h(y)) \approx \mathrm{sim}(x, y)$$

where $\mathrm{sim}(x, y)$ represents the Jaccard similarity between x and y.

This indicates a similarity function, possibly using cosine or sine similarity to measure the resemblance between two data points.

$$\Pr(C_{x_i} \text{ is retrieved}|q) = \left(1 - C\left(1 - \mathrm{sim}(x_i, q)\right)^k\right)^2$$

2.4 Monatomic Query and Likelihood of Retrieval in Anomaly Detection

This approach involves a monatomic query where K is the parameter in the hash table. The likelihood of retrieval is computed by evaluating the similarity to every other algorithm, then taking a 50% probability based on an address in a hash table. This process incorporates parameters like similarity and the number of elements.

Let $w_i = \mathrm{sim}(x_i, q)$ represent the similarity between x_i and the query q.

The sample weights w_i are defined using a similarity measure, which may help in determining the likelihood of an item being an anomaly.

2.5 Weighted Sampling and Proportion Calculation

Let $X_i = \{x_1, x_2, \ldots, x_{10}\}$ represent a set of data points, and let $\omega_{10} = \{\omega_1^2, \omega_2^2, \ldots, \omega_{10}^2\}$ denote the query weights for these points, where $\sum \omega_i = 1$.

$$\text{The weight we sampled} \propto \left(1 - \left(1 - \mathrm{sim}(x_i, q)\right)^k\right)^2$$

Here, $\propto$ represents the proportional relationship in sampling weights.

The below represents the cumulative sum of weights up to 1:

$$0 \quad \omega_1^2 \quad \omega_1^2 + \omega_2^2 \quad \omega_1^2 + \omega_2^2 + \omega_3^2 \quad \ldots \quad 1$$

2.6 Locality Sensitive Hashing (LSH) as an Unbiased Estimator

Locality Sensitive Hashing (LSH) is claimed to be an unbiased estimator for detecting anomalies. By storing elements in a hash table, LSH helps to quickly identify high-risk items based on similarity and frequent patterns.

2.7 LSH as a Sampling Algorithm

The LSH sampling algorithm involves incrementing values in a data structure upon certain conditions, such as collisions, which might help in tracking anomalies when specific network patterns repeat. This approach includes a formula for calculating the expectation.

$$E[\text{retrieval count}] = f(\text{similarity}, K, \text{collisions})$$

This formula represents the expectation of the retrieval count, factoring in similarity, the hash table parameter K, and observed collisions in the data structure.

$$E\left(A\left[h(q)\right]\right) = \sum_{i=1}^{n} \mathrm{sim}(x_i, q)$$

$$A\left[h(q)\right] = \sum_{i=1}^{n} 1_{\{h(q)=h(x_i)\}}$$

1 - indicator variable

2.8 Similarity and Anomaly Detection in High-Dimensional Data

This analysis relates to the similarity between vectors or points, such as $\mathrm{Sim}(x_i, q)$, which indicates the expected similarity score when collisions occur. This approach is based on a 2018 research paper, suggesting a well-established method for detecting similarities and anomalies.

The lecture emphasized **random sampling**, where the algorithm does not differentiate between certain indices. This suggests that each data point has an equal probability of being sampled, regardless of its characteristics.

2.9 Anomaly Detection Through Similarity Summation

Anomaly detection is represented by a summation formula involving a function $f(x_i, q)$, where specific conditions help determine the presence of an anomaly. This method may consider the degree of similarity or divergence as an indicator, evaluating whether certain data points deviate significantly from others.

To achieve **optimal error**, sampling weights might be adjusted to prioritize more relevant or anomalous data points. **Locality-Sensitive Hashing (LSH)** aids in sampling values with higher weights, which optimizes the focus on significant anomalies.

2.10 Random Sampling and Clumping Detection Subroutine

A subroutine dedicated to handling random sampling and clumping detection can be implemented, potentially to manage clustered anomalies in the data. This subroutine would focus on detecting and managing clusters of anomalous data points, thereby improving the detection of significant patterns in the dataset.

3 Markov Chain Sampling

Markov Chain Sampling, specifically Markov Chain Monte Carlo (MCMC) methods, is a powerful technique used to sample from complex probability distributions when direct sampling is challenging. MCMC builds a sequence (or chain) of samples in a way that eventually follows the target distribution, allowing us to approximate integrals and expectations for large-scale or high-dimensional problems.

3.1 What is a Markov Chain?

A Markov Chain is a sequence of random states where each state depends only on the previous one (this is the Markov property). Mathematically, for a sequence of random variables $X_1, X_2, \ldots, X_n$, the probability of X_{n+1} depends only on X_n and not on prior values:

$$P(X_{n+1} = x | X_n = x_n, X_{n-1} = x_{n-1}, \ldots, X_1 = x_1) = P(X_{n+1} = x | X_n = x_n)$$

3.2 MCMC for Sampling

Markov Chain Monte Carlo (MCMC) methods use Markov chains to generate samples from a desired probability distribution by creating a sequence of states that eventually converge to the target distribution. Over time, the sequence of samples approximates the distribution closely, allowing us to estimate probabilities and expected values.

3.3 Key MCMC Algorithms for Sampling

Metropolis-Hastings Algorithm: One of the most common MCMC methods, where we:

1. Propose a new state Y based on the current state X.
2. Calculate the acceptance ratio r, which is the probability of accepting the proposed state:

$$r = \frac{\pi(Y) \cdot q(X|Y)}{\pi(X) \cdot q(Y|X)}$$

   where π is the target distribution and q is the proposal distribution.

3. Accept Y with probability $\min(1, r)$; otherwise, stay at X.

The algorithm generates a chain of states, with more time spent in areas of higher probability density.

Gibbs Sampling: A special case of MCMC, useful when sampling from high-dimensional distributions by breaking down the sampling into conditional steps:

- For each variable X_i in a multidimensional space, sample X_i conditioned on the other variables, X_{-i}.
- Iterate through all variables in turn to generate a sample for each one, conditioned on the latest values of the others.

Formula for Gibbs Sampling:

$$X_i^{(t+1)} \sim P(X_i | X_1^{(t+1)}, \ldots, X_{i-1}^{(t+1)}, X_{i+1}^{(t)}, \ldots, X_n^{(t)})$$

This approach is efficient for models with known conditional distributions.

3.4 Steps in MCMC Sampling

1. Initialize at an arbitrary starting point in the target distribution.
2. Generate Proposals using a transition rule (like the proposal distribution in Metropolis-Hastings).
3. **Acceptance Step:** Accept or reject the proposed state based on the acceptance ratio.
4. Repeat to create a chain of samples, and discard the initial "burn-in" period to ensure convergence.

3.5 Convergence and Burn-In Period

MCMC methods rely on the idea that, after a certain number of iterations, the Markov chain will converge to the target distribution. Early samples may not accurately represent this distribution, so a burn-in period discards these initial samples. Monitoring convergence can involve:

- **Trace plots:** Checking the stability of generated samples.
- **Autocorrelation:** Ensuring successive samples are not highly correlated.

Markov Chain sampling is essential for applications requiring complex distribution modeling, especially when traditional methods are impractical due to high dimensionality or computational cost.

4 Locality Sensitive Hashing (LSH)

Locality Sensitive Hashing (LSH) is a technique used to approximate nearest neighbor searches in high-dimensional data. It works by hashing input data points so that similar items (according to a chosen distance metric) are more likely to be hashed to the same "bucket" or hash value. This approach is highly efficient for applications involving large datasets, where exact nearest neighbor searches are computationally expensive.

4.1 How Locality Sensitive Hashing Works

LSH uses hash functions specifically designed to maximize the likelihood that similar items map to the same hash value while keeping different items in separate buckets. Unlike traditional hash functions that distribute values uniformly, LSH functions cluster similar items based on a predefined similarity measure, such as cosine similarity or Euclidean distance.

4.2 Key Concepts in LSH

Similarity Measure: The choice of distance metric (e.g., cosine similarity, Euclidean distance) determines the hash function type. Different LSH families are used based on the desired measure:

- **Cosine Similarity:** Uses random hyperplanes for hashing.
- **Euclidean Distance:** Uses hash functions based on random projections.

Hash Function Families: For LSH, a family of hash functions $H = \{h_1, h_2, \ldots, h_k\}$ is chosen so that:

- **Similar Items:** The probability $P(h(x) = h(y))$ is high for similar points x and y.
- **Dissimilar Items:** The probability $P(h(x) = h(y))$ is low for dissimilar points.

Multiple Hash Tables: To improve accuracy, multiple hash tables are often used, each with a different hash function. This approach reduces the chance of false negatives by increasing the likelihood that similar items end up in at least one common bucket.

4.3 LSH Algorithm Steps

Hashing Data Points:

- Each data point is hashed using a set of hash functions. Similar points are more likely to hash to the same bucket.
- For example, if using cosine similarity, the hash function may map points based on their orientation relative to random hyperplanes.

Querying for Nearest Neighbors:

- Given a query point, LSH retrieves all points that fall into the same bucket across all hash tables.
- The retrieved candidates are then checked to find the actual nearest neighbors, reducing the computation needed compared to searching through all data points.

Adjusting Accuracy and Efficiency:

- Parameters like the number of hash functions per table (k) and the number of hash tables (L) balance between computation time and retrieval accuracy.

4.4 Example of LSH with Cosine Similarity

For cosine similarity, LSH hashes points based on random hyperplanes:

- A random hyperplane is chosen by sampling a vector r from a standard Gaussian distribution.
- For a point x, the hash is determined by the sign of the dot product $r \cdot x$:

$$h(x) = \text{sign}(r \cdot x)$$

- Multiple random vectors create a series of hash bits, creating a unique binary hash for each point based on its location relative to each hyperplane.

Locality Sensitive Hashing simplifies similarity search in high-dimensional spaces, making it invaluable in data-intensive fields like information retrieval, computer vision, and data mining. Its ability to approximate similarity-based grouping while minimizing computational load is ideal for handling large-scale datasets.

5 Adaptive Sampling Techniques

Adaptive Sampling Techniques are methods that adjust sampling strategies dynamically based on the importance or characteristics of data points, aiming to improve sampling efficiency by focusing more resources on areas of interest. Unlike traditional fixed sampling methods, adaptive sampling identifies regions or data points that are likely to yield the most useful information for the analysis, allowing better estimates with fewer samples.

5.1 Why Use Adaptive Sampling?

Adaptive sampling is particularly valuable in situations where:

- Data is imbalanced or concentrated in certain regions (e.g., rare events or anomalies).
- Computational resources are limited, requiring selective sampling to minimize cost.
- Certain areas of the data space are more informative, such as areas with higher variability or areas relevant to a specific hypothesis.

5.2 Key Adaptive Sampling Techniques

Importance Sampling

- **Concept:** Samples are drawn from a distribution that emphasizes areas where the target function is larger or more variable.
- **Method:** Weight each sample according to the ratio of the target and proposal distributions:

  $$E_f[h(X)] \approx \frac{1}{N}\sum_{i=1}^{N} h(X_i)w(X_i), \quad \text{where } w(X_i) = \frac{f(X_i)}{g(X_i)}$$

  Here, $f(X)$ is the target distribution, and $g(X)$ is the proposal distribution. The weights $w(X_i)$ adjust for the discrepancy between distributions.
- **Applications:** Common in scenarios with rare events or tail estimations (e.g., financial risk modeling, reliability engineering).

Adaptive sampling techniques are crucial for applications that demand efficient and accurate data analysis, especially in high-dimensional spaces, real-time systems, and resource-limited environments. They provide significant computational savings and improved estimation accuracy by selectively focusing on the most informative areas of the data space.

6 Estimating Similarity Scores in High-Dimensional Data

Estimating similarity scores in high-dimensional data involves calculating a measure of how similar or close data points are in complex, high-dimensional spaces. High-dimensional data is common in fields like image recognition, natural language processing, and genomics, where each data point can have hundreds or thousands of attributes or features.

6.1 Challenges in High-Dimensional Similarity Estimation

- **Curse of Dimensionality:** In high-dimensional spaces, distances between points become less meaningful, as all points tend to appear equally distant from one another.
- **Computational Complexity:** Calculating pairwise similarity for large datasets with high-dimensional vectors is computationally expensive.
- **Data Sparsity:** High-dimensional data often contains many zeroes or irrelevant features, making it harder to find meaningful similarities.

6.2 Common Similarity Measures

Cosine Similarity

- **Description:** Measures the angle between two vectors, ignoring their magnitude. It is widely used in text and image data.
- **Formula:**

$$\text{Cosine Similarity}(A, B) = \frac{A \cdot B}{\|A\| \|B\|}$$

  where $A \cdot B$ is the dot product, and $\|A\| \|B\|$ are the magnitudes of A and B.

- **Range:** $[-1, 1]$, where 1 indicates identical orientation, 0 is orthogonal, and -1 is completely opposite.

Euclidean Distance

- **Description:** Measures the straight-line (or Euclidean) distance between points.
- **Formula:**

$$\text{Euclidean Distance}(A, B) = \sqrt{\sum_{i=1}^{n} (A_i - B_i)^2}$$

- **Applications:** Used in clustering and nearest neighbor searches; however, it can become less effective in high-dimensional spaces.

Jaccard Similarity

- **Description:** Measures similarity between sets by comparing their intersection and union.
- **Formula:**

$$\text{Jaccard Similarity}(A, B) = \frac{|A \cap B|}{|A \cup B|}$$

- **Applications:** Useful for sparse, binary data like word occurrences in text documents.

6.3 Techniques for Efficient Similarity Estimation in High Dimensions

Locality Sensitive Hashing (LSH)

- **Overview:** A hashing technique that groups similar items into the same "bucket" by mapping high-dimensional vectors to a lower-dimensional space. LSH uses hash functions tailored to the similarity metric (e.g., cosine or Euclidean) to ensure similar items are more likely to share a bucket.
- **Applications:** Useful in approximate nearest neighbor search, reducing the need to compute exact distances for all pairs.

Dimensionality Reduction

- Techniques like Principal Component Analysis (PCA) and t-SNE reduce the number of dimensions while retaining most of the data's variance or structure, making similarity computations more manageable.
- **Applications:** Often used as a preprocessing step in clustering, classification, or nearest neighbor tasks.

Random Projection

- **Overview:** Projects high-dimensional data into a lower-dimensional subspace using a random matrix. This method preserves distances approximately according to the Johnson-Lindenstrauss Lemma, which states that the distances between points in high dimensions can be preserved within a small error margin.

- **Formula:** Given a random matrix R, data vector X in high-dimensional space is projected to X' in a lower-dimensional space as:

$$X' = X \cdot R$$

Sparse Vector Techniques

- High-dimensional data can often be sparse (i.e., many features have zero or negligible values). By focusing on non-zero features or reducing dimensions to sparse components, similarity calculations become more efficient.
- Techniques like Sparse PCA and Truncated SVD help reduce computation without significant information loss.

Estimating similarity scores in high-dimensional data is critical for understanding relationships in complex datasets, enabling efficient clustering, recommendation, and anomaly detection across many fields.