UNIT 4

Nearest neighbor search:

Nearest neighbor search (**NNS**), as a form of proximity search, is the optimization problem of finding the point in a given set that is closest (or most similar) to a given point. Closeness is typically expressed in terms of a dissimilarity function: the less similar the objects, the larger the function values. Formally, the nearest-neighbor (NN) search problem is defined as follows: given a set S of points in a space M and a query point $q \in M$, find the closest point in S to q. Donald Knuth in vol. 3 of The Art of Computer Programming (1973) called it the **post-office problem**, referring to an application of assigning to a residence the nearest post office. A direct generalization of this problem is a k-NN search, where we need to find the k closest points.

Most commonly M is a metric space and dissimilarity is expressed as a distance metric, which is symmetric and satisfies the triangle inequality. Even more common, M is taken to be the d-dimensional vector space where dissimilarity is measured using the Euclidean distance, Manhattan distance or other distance metric. However, the dissimilarity function can be arbitrary. One example is asymmetric Bregman divergence, for which the triangle inequality does not hold.

Applications of Near-Neighbor Search:

The nearest neighbour search problem arises in numerous fields of application, including:

- Pattern recognition – in particular for optical character recognition
- Statistical classification – see k-nearest neighbor algorithm
- Computer vision
- Computational geometry – see Closest pair of points problem
- Databases – e.g. content-based image retrieval
- Coding theory – see maximum likelihood decoding
- Data compression – see MPEG-2 standard
- Robotic sensing[2]
- Recommendation systems, e.g. see Collaborative filtering
- Internet marketing – see contextual advertising and behavioral targeting
- DNA sequencing
- Spell checking – suggesting correct spelling
- Plagiarism detection
- Similarity scores for predicting career paths of professional athletes.
- Cluster analysis – assignment of a set of observations into subsets (called clusters) so that observations in the same cluster are similar in some sense, usually based on Euclidean distance
- Chemical similarity
- Sampling-based motion planning

Collaborative Filtering as a Similar-Sets Problem

Collaborative filtering (**CF**) is a technique used by recommender systems. Collaborative filtering has two senses, a narrow one and a more general one.

In the newer, narrower sense, collaborative filtering is a method of making automatic predictions (filtering) about the interests of a user by collecting preferences or taste information from many users (collaborating). The underlying assumption of the collaborative filtering approach is that if a person *A* has the same opinion as a person *B* on an issue, A is more likely to have B's opinion on a different issue than that of a randomly chosen person. For example, a collaborative filtering recommendation system for television tastes could make predictions about which television show a user should like given a partial list of that user's tastes (likes or dislikes). Note that these predictions are specific to the user, but use information gleaned from many users. This differs from the simpler approach of giving an average (non-specific) score for each item of interest, for example based on its number of votes.

In the more general sense, collaborative filtering is the process of filtering for information or patterns using techniques involving collaboration among multiple agents, viewpoints, data sources, etc.[2] Applications of collaborative filtering typically involve very large data sets. Collaborative filtering methods have been applied to many different kinds of data including: sensing and monitoring data, such as in mineral exploration, environmental sensing over large areas or multiple sensors; financial data, such as financial service institutions that integrate many financial sources; or in electronic commerce and web applications where the focus is on user data, etc. The remainder of this discussion focuses on collaborative filtering for user data, although some of the methods and approaches may apply to the other major applications as well.

Data Stream Mining is the process of extracting knowledge structures from continuous, rapid data records. A data stream is an ordered sequence of instances that in many applications of data stream mining can be read only once or a small number of times using limited computing and storage capabilities.

In many data stream mining applications, the goal is to predict the class or value of new instances in the data stream given some knowledge about the class membership or values of previous instances in the data stream. Machine learning techniques can be used to learn this prediction task from labeled examples in an automated fashion. Often, concepts from the field of incremental learning are applied to cope with structural changes, on-line learning and real-time demands. In many applications, especially operating within non-stationary environments, the distribution underlying the instances or the rules underlying their labeling may change over time, i.e. the goal of the prediction, the class to be predicted or the target value to be predicted, may change over time. This problem is referred to as concept drift. Detecting concept drift is a central issue to data stream mining. Other challenges that arise when applying machine learning to streaming data include: partially and delayed labeled data, recovery from concept drifts, and temporal dependencies.

Examples of data streams include computer network traffic, phone conversations, ATM transactions, web searches, and sensor data. Data stream mining can be considered a subfield of data mining, machine learning, and knowledge discovery.

A **data stream management system** (DSMS) is a computer software system to manage continuous data streams. It is similar to a database management system (DBMS), which is, however, designed for static data in conventional databases. A DSMS also offers a flexible query processing so that the information needed can be expressed using queries. However, in contrast to a DBMS, a DSMS executes a *continuous query* that is not only performed once, but is permanently installed. Therefore, the query is continuously executed until it is explicitly uninstalled. Since most DSMS are data-driven, a continuous query produces new results as long as new data arrive at the system. This basic concept is similar to Complex event processing so that both technologies are partially coalescing.

Count-distinct problem

In computer science, the **count-distinct problem** (also known in applied mathematics as the **cardinality estimation problem**) is the problem of finding the number of distinct elements in a data stream with repeated elements. This is a well-known problem with numerous applications. The elements might represent IP addresses of packets passing through a router, unique visitors to a web site, elements in a large database, motifs in a DNA sequence, or elements of RFID/sensor networks.

Formal definition

Instance: A stream of elements with repetitions, and an integer . Let be the number of distinct elements, namely , and let these elements be .

Objective: Find an estimate of using only storage units, where .

An example of an instance for the cardinality estimation problem is the stream: . For this instance, .

Naive solution

The naive solution to the problem is as follows:

```
 Initialize a counter, c, to zero,     .
 Initialize an efficient dictionary data structure, D, such as hash table or
search tree in which insertion and membership can be performed quickly.

 For each element     , a membership query is issued.

     If      is not a member of D (    )

         Add      to D

         Increase c by one,

     Otherwise (    ) do nothing.
```

```
Output      .
```

As long as the number of distinct elements is not too big, *D* fits in main memory and an exact answer can be retrieved. However, this approach does not scale for bounded storage, or if the

computation performed for each element should be minimized. In such a case, several streaming algorithms have been proposed that use a fixed number of storage units.