Stanford CS 246 Notes

Yikun Chi

September 26, 2022

Most of the contents in this notes are from Stanford CS246 class slides and the book Mining of Massive Datasets

Contents

Chapter 1

Introduction, Spark and MapReduce

1.1 MapReduce: Early Distributed Computing Programming Model

In general, MapReduce is a style of programming designed for

- Easy parallel programming
- Invisible management of hardware and software failures
- Easy management of very-large-scale data

Some common implementations of MapReduce includes Hadoop, Spark, Flink, and the original Google implementation "MapReduce".

1.1.1 Component of MapReduce

The MapReduce framework has the following steps (note that user only needs to write map function and reduce function)

1. Map: Apply a user-written Map function to each input element.
    - Mapper applies the Map function to a single element
    - Many mappes grouped in a Map task (the unit of parallelism)
    - The output of the Map function is a set of 0, 1, or more key-valued pairs
    - The keys do not have to be unique. A Map task can produce several key-value pairs with the same key even from the same element.
2. Group by Key: Sort and shuffle

- System sorts all the key-value pairs by key, and output key - (list of values) pairs

3. Reduce : User-written Reduce function is applied to each key - (list of values). The output is another sequence of zero or more key-value pairs.

The MapReduce environment will automatically takes care of

- Partitioning the input data
- Scheduling the program's execution across a set of machines
- Performing the group by key step
- Handling machine failures
- Managing required inter-machine communication

Quick Example: Word Count
Task: We have a huge text document, and we need to count the number of times each distinct word appears in the file. The map function reads the input and produces a set of key-value pairs where the key is a word, and the value is 1. The reduce function is the sum function. (Note that because the Reduce function is associative and communicative, we can push some of the reducers work, addition, to the Map task. So the key is a word, but the value from Map function could be more than 1).

1.1.2 MapReduce and Machine Failures

If map worker failure:

- Map tasks completed or in-progress at worker are reset to idle and rescheduled
- Reduce workers are notified when map task is rescheduled on another worker

If reduce worker failure:

- Only in-progress tasks are reset to idle and the reduce task is restarted.

1.1.3 Algorithms Using MapReduce

1.1.3.1 Matrix-Vector Multiplication

Setup:
A $n \times n$ matrix M and a vector v of length n. We want to calculate $x = Mv$ where $x_i = \sum_{j=1}^{n} m_{ij} v_j$. We assuming that m_{ij} and v_j are discoverable by their coordinate i, j.We also assume v can be read into memory in its entirety.

Map:
For each matrix element m_{ij}, it produces the key-value pair $(i.m_{ij}v_j)$.

Reduce:
The Reduce function simply sums all the values with a given key i.

1.1.3.2 Matrix-Vector Multiplication with large vector

Setup:
If v is too large to fit into the memory, we slice M vertically and v horizontally in the same fashion. We can then divide M and v into different files where each files contains a set of stripes. See example in figure 1.1. We can then proceed normally with Map and Reduce.

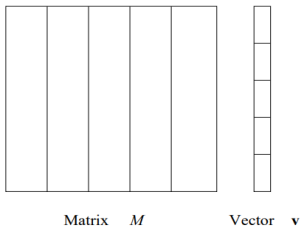

Figure 1.1: Division of a matrix and vector into five stripes

1.1.3.3 Relational-Algebra Operations: Selection

Task: Apply a condition C to each tuple in the relation(a table) and produce as output only those tuples that satisfy C. Result is labeled as $\sigma_C(R)$.

Map: For each tuple t in R, test if it satisfies C. If so, produce the key value pair (t, t), otherwise produce nothing.

Reduce: Identity function.

1.1.3.4 Relational-Algebra Operations: Projection

Task: For some subset S of the attributes of the relation, produce from each tuple only the components for the attributes in S. Result is labeled as $\pi_S(R)$.

Map: For each tuple t in R, construct a tuple t' by eliminating from t those components whose attributes are not in S. Output key-value pair (t', t')

Reduce: Identity function.

1.1.3.5 Relational-Algebra Operations: Union

Map: Turn each input tuple t into key-value pair (t, t) regardless which relation it comes from.

Reduce: Produce (t, t) regardless the input value is $(t, [t])$ or $(t, [t, t])$.

1.1.3.6 Relational-Algebra Operations: Intersection

Map: Turn each input tuple t into key-value pair (t, t) regardless which relation it comes from.

Reduce: Produce (t, t) only if the input value is $(t, [t, t])$.

1.1.3.7 Relational-Algebra Operation: Difference

Task: $R - S$.

Map: Produce key-value pair (t, X) where X can be either R or S based on where the input tuple is.

Reduce: Produce (t, t) only if the input value $(t, [R])$. e.g.: $(t, [R, S])$ means that the tuple exists in both relation.

1.1.3.8 Relational-Algebra Operation: Natural Join

Task: Inner join $R(A, B)$ with $S(B, C)$ on shared attributes B.

Map: For each tuple (a, b) of S, produce the key-value pair $(b, (R, a)$. Produce $(b, (S, c)$ for tuple in S.

Reduce: Input is $(b, (R, a), (S, c), (R, a'), (S, c'), ...$ Output: pairwise across R and S, and add b. So $(x, (b, a, c), (b, a', c), ...)$. Here key is x because it doesn't matter.

1.1.3.9 Relational-Algebra Operation: Grouping and Aggregation

Task: Group by attribute A, aggregating over attribute B and left over attribute C.

Map: For each tuple (a, b, c), produce key-value pair (a, b).

Reduce: The aggregation function.

1.1.3.10 Matrix Multiplication - Two Step

Task: calculate $P = MN$ where $p_{ik} = \sum_j m_{ij} n_{jk}$. We can think of MN as a natural join followed by grouping and aggregation. The natural join is

$M(I, J, V)$ with $N(J, K, W)$. This will gives us tuples $(i, j, k, v \times w)$. Subsequently we can group by I, K and aggregate over J.

1.1.3.11 Matrix Multiplication - One Step

Map: For each element m_{ij} of M, produce all the key-value pairs $((i, k), (M, i, m_{ij}))$ for $k = 1, 2, ...$ up to number of columns of N. For each element n_{jk} of N, produce all the key-value pairs $((i, k), (N, j, n_{jk}))$ for $i = 1, ...$ up to the number of rows of M.

Reduce: Each key (i, k) will have an associated list with all the values (M, j, m_{ij}) and (N, j, n_{jk}) for all possible values of j. The Reduce function connects two values on the list with the same j, and multiply m_{ij} with n_{jk} followed by summing at the key (i, k) level.

1.2 Spark: Extends MapReduce

1.2.1 Data-Flow Systems

MapReduce uses two ranks of tasks (Map, Reduce), and the data flows from the first rank to the second. Data-Flow Systems generalize this process by 1) Allowing any number of tasks / ranks, and 2) Allow functions other than Map and Reduce. As along as data flow is in one direction only, we can have the blocking property and allow recovery of tasks rather than whole jobs. So a dataflow / workflow system is a system with acyclic computation graph. Spark is a popular data-flow systems.

1.2.2 Spark Key Ideas

1.2.2.1 Resilient Distributed Dataset (RDD)

RDD is partitioned collection of records. It is stored across clusters, and read-only. Rdds can be created from Hadoop, or by transforming other RDDs (map, filter, join, union, intersection, distinct). The transformations are not computed until an action(count, collect, reduce, save) requires it

1.2.2.2 DataFrame

DataFrame are data organized into named columns. It is like a table in a relational database.

1.2.2.3 Dataset

Dataset extends DataFrame API provides type-safe, object-oriented programming interface.

Chapter 2

Frequent Itemset Mining & Association Rules

2.1 Intro

2.1.1 Setup

Given a large set of items and a large set of baskets. Each basket is a small subset of items. We want to discover association rules, which is people who bought $\{x, y, z\}$ tend to buy $\{v, w\}$. This is a general many-to-many mapping. It can be applied to many areas such as words/documents, drugs/patients, products/shopping carts and etc.

2.1.2 Frequent Itemsets

Support for itemset I is defined as number of baskets containing all items in I (sometime expressed as a fraction of the total number of baskets). The **support threshold** s is the threshold for the itemsets to be considered frequent. Note that itemset I could be a set with only one item.

2.1.3 Association Rules

An association rule is a if-then rules in the form of $\{i_1, ..., i_n\} \to j$, meaning if the basket containing item $i_1, ...$, it will likely to contain j.

The problem of association rule mining is to find all association rules with support $\geq c$ and confidence $\geq c$. The support of an association rule is the support of the set of items in the rule. The hard part of this problem is finding the frequent itemsets.

2.1.3.1 Confidence

The confidence of association rule is defined as

$$p(j|I) = \frac{support(I \cup j)}{support(I)}$$

2.1.3.2 Interest

The interest of an association rule is defined as

$$|Conf(I \to j) - P(j)|$$

It is used to filter out high confidence items that are due to the fact that they are in every basket. The absolute value capture both positive and negative associations between itemsets and items. In general, interest above 0.5 is good.

2.2 Mining Association Rules

The rules have two steps:

1. Find all frequent itemsets I
2. Rule generation
   - For every subset A of I, generate a rule $A \to I - A$
   - Intuition: since I is frequent, A is also frequent.
   - Variant 1: Single pass to compute the rule confidence
   - Variant 2: Notice that the confidence of rule $A, B, C \to D$ is always bigger than the confidence of rule $A, B \to C, D$. Because in the formula, the top is always the support of A, B, C, D. The support of A, B, C is definitely smaller or equal to the support of A, B. So if $A, B \to C, D$ is above confidence, then so is $A, B, C \to D$
3. Output the rules above the confidence threshold

To reduce the number of rules, we can post-process them and only output:

- Maximal frequent itemsets: No immediate superset is frequent
- Closed itemsets: No immediate superset has the same support

2.3 Finding Frequent Itemsets

Setup: Items are positive integers. Baskets configurations are just a long sequence of integers, and use -1 to indicate a basket break. We measure the cost of the algorithm by the number of passes it makes over the data (map to number

of disk I/Os).

For many frequent-itemset algorithms, main-memory is the critical resource. The number of different pairs we can count is limited by main memory.

Overall intuition: We want to generate all itemsets, but only keep count on the ones that will in the end turn out to be frequent (above certain support).

2.3.1 A - Priori Algorithm

We start with A-Priori algorithm for frequent pairs, and then extend to itemset.

Monotoncity If a set of items I appears at least s times, so does every subset J of I. The contrapositive statement for pairs is that if item i does not appear in s baskets, then no pair including i can appear in s baskets. In terms of support, this means that if i has support of s, then no combination including item i can have support larger than s.

Algorithm
The key insight from monotonciy is that a pair can only be frequent item if both items in the pair are frequent.

- Pass 1: Read baskets and count in main memory the number of occurrences of each individual item
- Identify frequent items with support threshold
- Pass 2: Read baskets again and keep track of the count of only those pairs where both elements are frequent. (Note this requires memory proportional to square of the number of frequent items, not the square of total items)
- Identify frequent pairs.

Extending to Triples and etc.
We can use an iterative process to obtain frequent k items. For each k, we start out by constructing candidate $k-$ tuples by using the information from pass for $k-1$, then we count all the candidate to obtain the set of truly frequent $k-$ tuples.

Using $k = 3$ as an example, we already have a list of frequent pairs. We now generate all possible $3-$ tuples candidate, and then do a pass on on all candidates and obtain the true pairs. Note that when we are generating possible $3-$ tuples, any candidate containing a pair that is not in the frequent pair list is not valid.

2.3.2 Park-Chen-Yu Algorithm

Modification to A-Priori During the first pass, maintain a hash table with as many buckets as we can fit into the idle memory. For each pair of items in the basket, increase the count of the bucket that the pair will hash to. During second pass, only count pairs that hash to frequent buckets.

Reasoning If a bucket count is less than s, then we know the sum of the occurrence of all pairs that would hash into the bucket is less than s. So none of its pairs can be frequent. Between passes, we can also replace the count of the bucket with a bit indicator.

2.4 Finding Frequent Itemsets (non-comprehensively)

If we have a lot of data, can we use 2 or fewer passes if we allow missing some frequent itemsets.

2.4.1 Random Sampling Algorithm

Take a random sample of the market baskets and run a-priori or one of its improvements. We can reduce the support threshold proportionally to match the sample size. We can then do a second pass of the entire data set to verify if the selections are truly frequent.

2.4.2 Savasere, Omiecinski, and Navathe

Two pass algorithm on the entire dataset

- Pass 1: Repeatedly read small subsets of the baskets into main memory and run an in-memory algorithm to find all frequent itemsets(With a proportional support threshold).
- Pass 2: Count all the candidate itemsets and determine which are frequent in the entire set (Essentially pigenon hole principle).

2.4.3 Toivonen

- Pass 1: Start with a random sample, but lower the threshold slightly for the sample. E.g.: if the sample is 1 %, use $s/125$ as the support threshold. Find frequent itemsets in the sample.
  - Find all frequent itemsets in the sample
  - Add the negative border to the itemsets that are frequent in the sample. (Negative border contains itemsets that are not frequent in the sample, but ALL of its immediate subsets are)

- Pass 2: Count the frequency of all candidate as well as the negative borders. If no itemset from the negative border turns out to be frequent, then we found all the frequent itemsets. Otherwise, start over again with another sample.

Chapter 3

Finding Similar Item: Locality-Sensitive Hashing

Corresponding textbook: Chapter 3 of Mining Massive Datasets

3.1 Intro

Given high dimensional large volume data points $x_1, ...$ and some distance function $d(x_1, x_2)$. We want to

1. Given q, find data points x_j that are within distance threshold $d(q, x_j) \leq s$ (How to do this less than $O(N)$
2. Find all pairs of data points (x_i, x_j) that are within distance threshold $d(x_i, x_j) \leq s$ (How to do this less than $O(N^2)$

LSH : is a family of related techniques. In generall it aims to hash similar items into the same bucket. LSH is not precise. Flase negatives exists, i.e.: similar items could be missed.

3.2 LSH Example: Similar Document

3.2.1 Overall Structure

1. Shingling: Converts a document into a set representation (Boolean vector). The output is the set of strings of length k that appear in the document.
2. Min-Hashing: Convert large sets of shingles to short signatures, while preserving similarity The signatures are short integer vectors that represent the sets, and reflect their similarity.

3. Locality-Sensitive Hashing: Focus on pairs of signatures likely to be from similar documents. Outputs are candidate pairs of signatures for which we need to test for similarity.

3.2.2 Shingling

k-shingle is the same as k-gram. k-gram can be generate at either characters, words, or some other level depends on the application.

In order to compress long shingles, we can hash shingle into integer, e.g.: 4 bytes. So the final output of a document D_i is the set of hashed values of its k-shingles $C_i = S(D_i)$.

Similarity of the output can be measured: by

$$\text{Jaccard similarity: } sim(D_1, D_2) = \frac{|C_1 \cap C_2|}{|C_1 \cup C_2|}$$
$$\text{Jaccard Distance: } d(C_1, C_2) = 1 - \frac{|C_1 \cap C_2|}{C_1 \cup C_2|}$$

The final output is a matrix representation of sets, with rows being set item (shingle) and columns are the documents. Column similarity is the Jaccard similarity of the corresponding sets (b/c rows with value 1).

3.2.3 Min-Hashing

Contex
In this step, we want to reduce the set representation to a smaller signature. The goal is to find a hash function h such that if $sim(C_1, C_2)$ is high, then with high probability $h(C_1) = h(C_2)$.

- Note here the hashing isn't going to be exact, hence the word probability.
- Not all similarity metrics have a suitable hash function. For Jaccard similarity, the hash function is called Min-Hashing

Min-Hashing Overview
Given the boolean matrix post Shingling, we apply several random permutation π to the matrix. The minhash function for column C and permutation π is the number of the first row post permutation in which column C has value 1, i.e.: $h_\pi(C) = \min_\pi \pi(C)$.

The output of min-hashing is the signature matrix, where the columns are the set (document shingling output) and the rows are the permutation. Cell value is the minhash value for the corresponding column and permutation. The signature of the column / document is the minhash values for all the permutation.

Min-Hashing Similarity
The similarity of the signatures are the percent of the signature that two columns agree on. We can show that min-hash similarity equals to the column Jaccard similarity in expectation. Proof see slides. Quick proof below:

Choose a random permutation π. Let X Let x be a doc (set of shingles), $z \in X$ is a shingle

It is equally likely that any $z \in X$ is mapped to the min element

$\implies Pr(\pi(z) = min(\pi(X))) = \dfrac{1}{|X|}$

Let y be s.t. $\pi(y) = min(\pi(C_1 \cup C_2)) = \dfrac{1}{|C_1 \cup C_2|}$

There are $|C_1 \cap C_2|$ number of options.

$\implies Pr(min(\pi(C_1)) = min(\pi(C_2))) = \dfrac{|C_1 \cap C_2|}{|C_1 \cup C_2|} = sim(C_1, C_2)$

Min-Hashing Implementation
In reality, we do not permute the rows. We pick a row hashing function. If we want to have 100 signature, we want to pick $K = 100$ hash function h_i, and the ordering under h_i gives a random permutation π of rows. The actual one-pass implementation would be

- For each column c and hash-function h_i, keep a slot $M(i, c)$ which is the min-hash value
- Initialize all $M(i, c)$ to be ∞
- Scan rows looking for 1s
  - If row j has 1 in column c
  - For each h_i: $M(i, c) = min(M(i, c), h_i(j))$

One universal hashing function is $h_{a,b}(x) = ((ax + b) \mod p) \mod N$ where a, b are random integers and p is a prime number that is greater than N

3.2.4 Locality-Sensitive Hashing

key idea
The goal is to find the documents with Jaccard similarity at least s. The idea is to use a hash function that tells whether x and y is a candidate pair for which the similarity must be evaluated. For Min-Hash matrices, we want to hash columns of signature matrix M to many buckets. Each pair of documents that hashes into the same bucket is a candidate pair.

Overview

- Divide matrix M (signature matrix from Min-Hashing) into b bands of r rows. (so $r \times b$ is the number of hash functions in the Min-hashing step.
- For each band, hash its portion of each column to a hash table with k buckets (make k as large as possible so no collision).
- Candidate column pairs are those that hash to the same bucket for ≥ 1 bands
- Tune b and r to catch most similar pairs but few non-similar pairs.

3.3 Theoretical Analysis of LSH

Ideally, we want LSH to work like figure 3.1. When the similarity is above the treshold, there is 100% chance that the band will be hashed into the same bucket.

If we only have one bucket and 1 hash function, we know that $p(h(c_1) = h(c_2)) = sim(D_1, D_2)$, which is shown in figure 3.2.

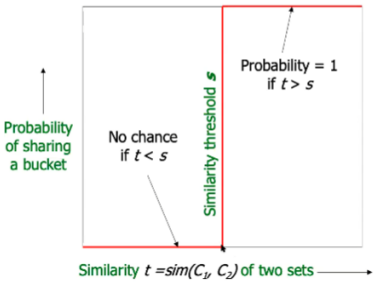

Figure 3.1: Ideal

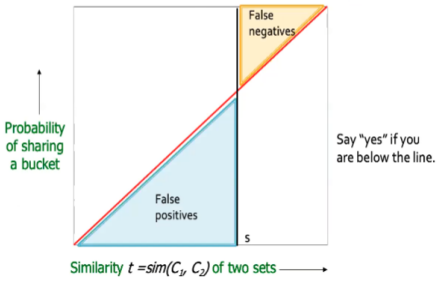

Figure 3.2: 1 signature 1 band

Now let's say we have b bands and r rows per band. Two columns have similarity s. The probability that at least 1 band is identical (becoming candidate pair) is $1 - (1 - s^r)^b$.

- The probability of a single row is the same is s
- The probability of of all rows the are the same in a band is s^r
- The probability of some rows in the band is not the same is $1 - s^r$
- The probability that all bands are not equal is $(1 - s^r)^b$

So given a fixed threshold t, we want to pick r and b to get the best S-curve (close to the step function). Notice different curve will have trade-off of false positive (blue region in figure 3.2 and false negative (yellow region). For more exploration, see demo.

3.4 Generalizing Min-hash to Locality Sensitive Hash Function

In the previous example, our input is a series of documents. We use shingling to transform the document, use Jaccard similarity / distance as the metrics, and use min-hash function to generate the key signature. Now, we want to generalize the distance + hash function to generate signatures.

3.4.1 Distance Measure

Let $d(\cdot)$ is a distance measure if it is a function from pairs of points x, y to a real number s.t.

- $d(x, y) \geq 0$
- $d(x, y) = 0 \qquad iff x = y$
- $d(x, y) = d(y, x)$
- $d(x, y) \leq d(x, z) + d(z, y)$ Triangle inequality.

Some common distance measurement includes

- Jaccard distance: $1 -$ Jaccard similarity
- Cosin distance: Angle between the vectors, $1 - \frac{A \cdot B}{\|A\|\|B\|}$
- Euclidean distance
  - L_2 norm (square root of the sum of the square of the difference in each dimension)
  - L_1 norm (sum of absolute value of the difference in each dimension)
- Edit distance: The minimum number of deletion / insertion to make two sequence equal. Alternative definition is $len(x)+len(y)-2*LCS$ (Longest common subsequence)
- Hamming Distance: The number of components that is different.

3.4.2 Locality-Sensitive Families for Hash Function

Suppose we have space S of points with a distance measure $d(x, y)$. A family H of hash functions (means we can effectively pick one at random and they do the same) is said to be $(d_1, d_2, p_1, p_2)-$sensitive if for any x, y in S:

- $d(x, y) \leq d_1 \Longrightarrow p(h(x) = h(y)) \geq p_1 \qquad \forall h \in H.$
- $d(x, y) \geq d_2 \Longrightarrow p(h(x) = h(y)) \leq p_2 \qquad \forall h \in H.$

3.4.2.1 Example: Jaccard similarity + Min-Hashing

Given Jaccard distance, let $d(x, y) \leq d_1$. So we know that $sim(x, y) = 1 - d(x, y) \geq 1 - d_1$. We know that the probability of min-hash function signatures agrees is same as Jaccard similarity.

3.4.3 Stacking LSH Families

3.4.3.1 AND Construction

Given a (d_1, d_2, p_1, p_2)-sensitive family F. We can construct a new family F' by the AND construction. $f' \in F'$ is constructed from taking the logical "and" of r set of functions from F: $\{f_1, ..., f_r\}$. So $f'(x) = f'(y)$ iff all f_i agrees.

The $f' \in F'$ is (d_1, d_2, p_1^r, p_2^r)

3.4.3.2 OR Construction

Similar to and construction.

The $f \in F'$ is $(d_1, d_2, 1 - (1 - p_1)^r, 1 - (1 - p_2)^r)$-sensitive.

3.4.3.3 Stacking And + OR

Notice "and" constructor push both probability lower, and "or" constructor push both probability higher. But they push to a different degree. So we can achieve a better outcome by stacking them, which would require more computation.

For example, if F is $(0.2, 0.6, 0.8, 0.4)$-sensitive. We can construct F_1 by using 4 and "and" constructor. We have F_1 is $(0.2, 0.6, 0.4096, 0.0256)$-sensitive. We then construct F_2 to be using 4 and "or" constructor, so F_2 is $(0.2, 0.6, 0.878, 0.0985)$. So comparing to the original F, we have both better false negative $(1 - 0.8) > (1 - 0.878)$ and better false positive $(0.4 > 0.0985)$

3.5 LSH Families for Hamming Distance

$f_i(x, y)$ is just the indicator function on whether the ith component of the vector is the same. The family F is $(d_1, d_2, 1 - \frac{d_1}{d}, 1 - \frac{d_2}{d})$-sensitive. The only limitations are 1) Hamming distance runs from 0 to d instead of 0 to 1, so we need to standardize. 2) There is only d number of functions in F.

3.6 LSH Families for Cosine Distance

Let vector x, y construct a plane A and θ be the angle between those two vectors on the plane. Consider a hyperplane B represented by the projection vector v. If the intersection of A and B is between vector x, y, then the dot product of $v \cdot x$ and $v\dot{y}$ will have different sign. Otherwise, the dot product will have the same sign. Since all angles are equally likely, so the probability that the dot product will have different sign is $\frac{\theta}{180}$.
So we can construction hash function $f \in F$ by picking a random vector v_f, and $f(x) = f(y)$ only if the dot product with v_f has the same sign. F is $(d_1, d_2, \frac{180-d_1}{180}, \frac{180-d_2}{180}$-sensitive.

Sketches Now instead of choosing random vector, we can restrict our choices to vector with component that is only 1 or -1. When picking a series of those vectors and compute the sign of the dot product with x, the result is called sketches of x. The percentage of sketches of x agree with sketches of y is the $\frac{\theta}{180}$ provided with enough vectors in sketches.

3.7 LSH Families for Euclidean Distance

Randomly pick a line, assuming the smaller angle between that line and (x, y) is θ. We divide the line into equally spaced bucket with α.
If $d(x, y)$ is smaller than $\frac{\alpha}{2}$, then there is at least 50% chance that x, y will be hashed into the same bucket. If $d(x, y) \geq 2\alpha$, then there can only be non-zero probability of hashing into the same bucket if $\cos(\theta)$ between 60 and 90 (1/3 chance) (otherwise cosine greater than 1/2).
So overall, we know F is $(\alpha/2, 2\alpha, 1/2, 1/3)$-sensitive.

Chapter 4

Clustering

Corresponding text: Chapter 7 of Mining Massive Datasets

4.1 Intro to Clustering

Two types of clustering techniques:

- Hierarchical / agglomerative algorithm: Start with each point in its own cluster and combine cluster step by step. Better when the cluster shapes are weird.
- Point Assignment: Points are considered in some order, and each one is assigned to the cluster into the best fit

Some other consideration:

- Euclidean space or not. Euclidean space cluster have a centroid, but non-Euclidean space might not necessarily have one.
- Can all the points fit into main memory

4.2 The Curse of Dimensionality

At high dimensional Euclidean space and many other non-Euclidean space, we have the following properties:

- Almost all pairs of points are equally far away from one another, and they are really far.
- Almost any two vectors are almost orthogonal

4.3 Hierarchical Clustering

4.3.1 Overall Structure

Begin with every points in its own cluster. At every step, pick two best cluster to merge.

4.3.2 Rules for Merging Cluster

Shortest distance between two centroids If we are in Euclidean space, we can define distance between two clusters as the Euclidean distance between their centroids. So we pick the two clusters at the shortest distance.

Minimum Minimum distance between any two points Let the distance of two clusters to be defined as the minimum distance between any two points from each cluster.

Minimum Average distance between all pairs of points Let the distance of two clusters to be defined as the average distance between all pairs of points in the cluster. Works better for weird structure such as rings.

Minimum radius of combined cluster Let radius of a cluster be the maximum distance between all the points and the centroid. Combine two clusters whose resulting cluster has the lowest radius.
A slight modification is to combine the clusters whose result has the lowest average distance between a point and the centroid.
Another modification could be to use the sum of the squares of the distances between the points and centroid

Minimum diameter of combined cluster Let the diameter of a cluster be the maximum distance between any two points of the cluster. Combine two cluster whose resulting cluster has the lowest diameter.

4.3.3 Rules for Stopping

Pre-defined number of clusters We stop when we reached the pre-defined number of clusters

Threshold: Average distance between centroid and points Take the average distance between the centroid and its points, if the distance across the threshold, stop.

Diameter / radius threshold Stop if the diameter / radius of the result cluster exceeds a threshold.

Density threshold Stop if the density of the result cluster is below some threshold. The density can be defined in many ways, but roughly it should be estimated by the number of points divided by some power of the diameter or radius of the cluster.

Evidence based Keep track of some metrics such as average diameter of all current cluster. If we combine two clusters that in reality should be separate, we expect a sudden jump of the metrics.

4.3.4 Efficiency Modification

Naive algorithm will take $O(n^2)$ time for the first step since each point is its own cluster and we need to compute the distance between each pair of cluster. Subsequent steps take $O((n-1)^2), O((n-2)^2)$.... So overall the algorithm is in $O(n^3)$. But we can run the below modification to get a better result;

1. Compute distance between all pairs of points $O(n^2)$ and insert the pairs and their distance into a priority queue

2. When we merge cluster C and D, remove all entries in the priority queue involving one of those two clusters $O(n \log n)$ since there are at most $2n$ deletions to be performed and priority queue deletion is $O(\log n)$

3. Compute all the distance between the new cluster and the remaining clusters $O(n \log n)$ time.

The last two step is executed at most n times. So overall we have $O(n^2 \log n)$

4.3.5 Hierarchical Clustering in Non-Euclidean Spaces

In non-Euclidean space, we no longer have cluster centroid. Instead, we can pick a point within the cluster to be the clustroid. Generally, points are picked to

- Minimize sum of the distances to the other points in the cluster

- Minimize the maximum idstance to another point in the cluster

- Minimize the sum of the squares of the distance to the other points in the cluster

All the rules rely on distance measure and they are valid as long as we swap it with the correct distance definition.
Alternatively, we can keep cluster just as a group of points. e.g.:

- Distance between cluster is the minimum / average distance between any two points in the cluster

4.4 Points Assignment: K-means Algorithm

4.4.1 Overall Structure

- Initially choose k points that are likely to be in different clustes and make them the centroid
- For each remaining point, find the centroid to which p is closest, add p to the cluster, adjust the centroid. Repeat the process until convergence.
- OPTIONAL: fixed the centroid and re-cluster all points

4.4.2 Initialization Strategy

Distance Based Randomly pick the first point, and keep picking the next point as the one that maximzies the minimum distance between the new point and the existing pool of points. Another way is to set the probability of addinga new point p is proportional to $D(p)^2$ where the $D(p)$ is the distance between p and nearest point in the sample pool.

Hierarchical clustering based Cluster a sample of data, and pick the point closest to the centroid from each cluster.

4.4.3 Picking the Right Value of K

Try different value of k, and access the quality through metrics (e.g.: average distance to centroid for all centroid). We know that this metric is largest when $k = 1$ and smallest when $k = N$. As we increase k, the average distance to centroid will rapidly fall. This is where k should be.

4.4.4 Bradley, Fayyad, and Reina (BFR) Algorithm

BFR is used when centroids information can be stored in main memory, but the overall data is too large to be fit into memory

Assumptions

- Euclidean space
- Normally distributed around a centroid
- Independent means and standard deviation along each dimension

Key Objects and representation in BFR

- Discard Set: Summaries of the clusters themselves
- Compressed Set: Summaries of sets of points that have been found close to one another, but not close to any existing cluster
- Retained Set: Points neither be assigned to a cluster nor are they sufficiently close to any other points to be forming compressed set.

The Discard Set and Compressed Set are represented by $2d+1$ in a d dimensional dataset. The numbers are: 1) The number of points in the set N, 2) The sum of the components of all the point in each dimension, 3) The sum of the squares of the components of all the point.

- Given those stats, we can calculate each component of the centroid location as $fracsumN$, the variance as $\frac{sumsq}{N} - (\frac{sum}{N})^2$

The points in Retained Set are held in main memory exactly as they appear in the input file.

Algorithm Assuming there are multiple chunks of data, for each chunk that we read in

1. All points that are sufficiently close to the centroid of a cluster (stored in Discard Set) are added to that cluster. Adjust that cluster's N, SUM, and $SUMSQ$
2. Cluster remaining points, along with points in the retained set. Any main-memory clustering algorithm can used. Clusters of more than one points are summarized and added to Compressed set. Singleton clusters become the retained set
3. Check if we can merge any mini-cluster in the compressed set (consists of old compressed set + new cluster from previous step) using certain criteria
4. Points that are assigned to a cluster or a minicluster (in the compressed set) are written out, with their assignment to secondary memory.

After last chunk of data, either treat compressed and retain set as outliers, or assign each point to the nearest centroid

Deciding when to add point to a cluster In the previous algorithm, the first step is to add a point to the cluster if it is sufficiently close to the centroids. There are two approaches:

- Add p to a cluster if it has the centroid closest to p, and very unlikely that after all the points have been processed, some other cluster centroid will be found nearer to p (harder to calculate)

- Based on the probability that if p belongs to a cluster, it would be found as far as it is from the centroid of that cluster

For the second approach, we can use the Mahalanobis distance. Let c be the centroid, with standard deviation σ_i in each dimension. The Mahalanobis distance between c and p is

$$\sqrt{\sum_{i=1}^{d} \frac{p_i - c_i}{\sigma_i}^2}$$

So we want add p to cluster c if it has the lowest Mahalanobis distance, and the distance is less than a threshold, e.g.: 4 (which means the probability of false negative is 1 in a million assuming normal distribution).

4.5 Clustering Using REpresentatives (CURE) Algorithm

CURE is also to be used in Euclidean space, but assumes no distribution form of the cluster

Process

1. Take a small sample of the data and cluster in main memory, ideally with hierarchical method.
2. Select a small set of points from each cluster to be representative points. Those points should be chosen to be as far from one another as possible.
3. Move each of the representative points a fixed fraction of the distance between its location and the centroid of its cluster (e.g.: 20%).
4. Merge two clusters if they have a pair of representative points that are sufficiently close. Repeat this step until convergence.
5. Bring in each point from the secondary storage, and compared with the representative points. Assign p to the cluster of the representative point that is closest to p.

Chapter 5

Dimension Reduction

Corresponding Text: Chapter 11 of MMDS

5.1 Eigenvalue and Eigenvector For Symmetric Matrices

5.1.1 Through determinant

The roots of the determinant of $M - \lambda I$ as an nth-degree polynomial in λ are the eigenvalues of M. We can then solve for $Me = ce$ where c is a known eigenvalue. This is a system of n equations and n unknowns, and then normalize e at the end.

5.1.2 Power Iteration

1. Initialize a non-zero vector x_k
2. Iterate $x_{k+1} = \frac{Mx_k}{\|Mx_k\|}$ (Frobenius norm) until converge
3. Get eigenvalue by $\lambda_1 = x^T Mx$
4. Create new matrix $M^* = M - \lambda_1 xx^T$, repeat previous step with the new matrix.

Side note: eigenvectors are orthogonal to each other, so a matrix of eigenvector E has the property of $EE^T = E^T E = I$

5.2 Principal-Component Analysis

Eigenvector matrix E (column ordered by decreasing eigenvalue) of $M^T M$ can be seen as a rotation and/or reflection of the original space. So ME is the

original data transformed into a new coordinate space and the axis are in decreasingly important fashion. Let E_k be the first k columns of E, then ME_k is a $k-$ dimensional principal component representation of M.

Sidenote: Connection between MM^T and M^TM : If e is an eigenvector of M^TM and $Me \neq 0$, then Me is an eigenvector of MM^T with λ as eigenvalue. Conversely, if e is an eigenvector of MM^T and $M^Te \neq 0$, then M^Te is an eigenvector of M^TM.
The eigenvalues of MM^T are the eigenvalues of M^TM plus additional 0's. If the dimension of MM^T is less than the dimension of M^TM, then the reverse is true.

5.3 Singular-Value Decomposition

5.3.1 Definition

Let M be an $m \times n$ matrix with rank r, then we can have the following decomposition $M = U\Sigma V^T$ where U, V are $m \times r, n \times r$ column-orthonormal matrix, and Σ is a diagonal matrix.

5.3.2 Interpretation

If we assumes that there are underlying concepts within the data, then SVD has the following interpretation:

- U matrix connects observation to concept. The value of M_{ij} is how much person i is tied to concept j
- Σ gives the strength of each concept
- V matrix connects features to concepts. The value of V_{ij} is how much feature i is tied to concept j

5.3.3 Dimensionality Reduction with SVD

Set s to be the smallest singular value and make it 0. Eliminate the corresponding s column of U and V. The result is equivalent to minimize the rmse / Frobenius norm of the difference between original matrix M and the approximation matrix M'

5.3.4 Querying with SVD

Example:

- Given a partial observation vector $q = [0, 0, ..., \lambda, 0, 0...]$, we can get the person's interest mapping through qV, and further recommendation through qVV^T

- We can obtain how similiar person a to person b by comparing their concept mapping aV and bV

5.3.5 Computing SVD

The SVD of a matrix M can be computed using the eigenvalues of M^TM or MM^T. Specifically:

- V is the matrix of eigenvectors of M^TM
- Σ^2 is the diagonal matrix whose entries are the corresponding eigenvalues.
- U is the matrix of eigenvectors of MM^T

5.4 CUR Decomposition

5.4.1 Definition

Let M be a matrix of m rows and n columns. Let r be the target number of concepts. A CUR-decomposition of M

- C, a $m \times r$ matrix from randomly chosen set of r columns of M,
- R, a $r \times n$ matrix from randomly chosen set of r rows of M
- U, a $r \times r$ matrix constructed the following way:
  - Let W be the $r \times r$ matrix from M by taking the corresponding columns and rows
  - Compute SVD of $W = X\Sigma Y^T$
  - Compute Σ^+, the Moore-Penrose psudoinverse of the diagional matrix (transpose and change the non-zero diagonal entry to inverse)
  - $U = Y(\Sigma^+)^2X^T$

The rows and columns are chosen independely (each row can be chosen multiple times) and proportionally to the Frobenius norm. So the probability of picking row i is $q_i = \frac{\sum_j m_{ij}^2}{\|M\|_f}$ and same goes for column. In addition, we normalize the chosen row/column by the $\sqrt{r * q_i}$ where r is the number of concept. In the event of duplicated rows / columns, we can merge them together. If we merge k rows of R into a single row, then the remaining row in R needs to be multiplied by $\sqrt{k}$. Finally, the consequence is that W is not a square matrix, but the decomposition still works.

Chapter 6

Recommendation Systems

Corresponding Text: Chapter 9 of MMDS

6.1 Utility Matrix

Utility matrix is a matrix with items as columns and users as rows. The cell value can be

- Binary indicating like / dislike
- 1 for user like the item, 0 for user did not choose the item
- Rating of the item

6.2 Content-Based Recommendations

Content based recommendations aims to generate a profile for item. Recommendations are based on similarity of the items.

6.2.1 Item Profiles

Item profile is a vector that describes the item.

6.2.1.1 Generating Profile

Categorical features can be encoded using one-hot encoding. For documents, we can use the top n / top n percent of the word measured by TF.IDF score as feature. Jaccard distance / cosine distance can be used to measure similarity. For images, we can rely on human tagging.

6.2.1.2 Scaling

Numerical features has to be scaled properly (tuning). Different scaling will impact similarity measures.

6.2.2 User Profile

User profile is a vector with the same component as item profile that describes user preference.

- For Boolean utility matrix, user profile is the average of the components of the item profile for the items in which the utility matrix has 1 for that user.
- For non-Boolean utility matrix, we can weight the utility matrix by subtracting the average value for a user. Then use this weight to take the average of the item profile.

6.2.3 Building content-based recommendations

We can produce recommendations either based on the cosine distance between the user's and item's vectors. Alternatively, we can build a classification algorithms for it using algorithms such as decision tree.

6.3 Collaborative Filtering

We want to focus on the similarity of the user ratings for two items. Use columns of utility matrix as item profile and rows of utility matrix as user profile. Recommendation for a user is then made by looking at the users that are most similar.

6.3.1 Measuring Similarity

Similarity can be measured using Jaccard Distance or Cosine distance depends on the utility matrix. Some additional pre-processing of the matrix can include

- Rounding: put low ratings as 0 and high ratings as 1
- Normalize: Subtract each rating from the average rating of that user.

6.3.2 The Duality of Similarity

Collaborative Filtering can be considered from either finding common user or finding common items.

- Given a user U and item I, we can find n similar users, and average their ratings for item I (counting only those who have rated I). We can get a recommendation directly this way. But intuitively, user to user similarity is less reliable since users can have complicated preferences.

- Alternatively, we can find m similar items, and take the average rating among the m items of the rating that U has given. But now we have to repeat this process for all items becore we can estimate the row for U.

6.3.3 Clustering Users and Items

We can choose to cluster the users and items in a iterative process.

1. Cluster the columns together
2. Clustering the rows together using post clustered columns
3. Repeat until desired extent, and then use the collaborative filtering technique for unknown cell values.

6.4 Dimensionality Reduction

Another approach for estimating the blank entries in the utility matrix can be seen as estimating the UV-decomposition of the matrix. I.e.: decomposing a $n \times m$ matrix into $n \times d$ and $d \times m$matrix.

6.4.1 UV Decomposition Evaluation

We use RMSE of the non-blank entries to evaluate the quality of UV decomposition. Mathematically, it is also the same as minimizing the sse.

6.4.2 Optimizing an arbitrary element

We can iteratively optimizing arbitrary elements of the U and V matrix. Suppose we want to vary u_{rs}, note that it only affects the row r of the product. We have

$$p_{rj} = \sum_{k=1}^{d} u_{rk} v_{kj}$$

Let $u_{rs} = x$, we have

$$p_{rj} = \sum_{k \neq s} u_{rk} v_{kj} + x v_{sj}$$

The rmse is

$$(m_{rj} - p_{rj})^2$$

After taking derivative, we have

$$x = \frac{\sum_j v_{sj}(m_{rj} - \sum_{k \neq s} u_{rk} v_{kj})}{\sum_j v_{sj}^2}$$

For element v_{rs}, the formula is

$$y = \frac{\sum_i u_{ir}(m_{is} - \sum_{k \neq r} u_{ik} v_{ks})}{\sum_i u_{ir}^2}$$

6.4.3 Pre-processing

In general, we want to normalize the matrix by

1. Subtract from each nonblank element m_{ij} the average rating of user i
2. Subtract the average rating of item j

6.4.4 Initialization

Normally we initialize U and V to be $\sqrt{a/d}$ where a is the average nonblank element of M and d with some pertubation.

Chapter 7

Link Analysis

Corresponding Text: Chapter 5 of MMDS

7.1 PageRank

7.1.1 Transition Matrix

Transition matrix is a column stochastic matrix. Entry M_{ij} indicates the probability of going to i from j. The long term limiting distribution is v such that $v = Mv$ as long as two conditions are met:

- The graph is strongly connected. It is possible to get from any node to any other node
- There are no dead end

7.1.2 General Structure of the Web

A web in general are composed of

- Strongly Connected Component (SCC)
- The in-component: pages that could reach SCC but not reachable from SCC
- The out-component: pages that are reachable from SCC but can't reach SCC
- Tendrils: 1) pages reachable from the in-component, but not able to reach the in-component. 2) pages that can reach out-component, but not reachable from the out-component.
- Tubes: pages reachable from the in-component and able to reach the out-component, but unable to reach the SCC or be reached from SCC
- Isolated component.

7.1.3 Avoding dead-end and Spider Traps

PageRank needs to avoid two situations:

- dead-end
- spider traps: a group of pages that all have outlinks but they never link to any other pages

They can be solved via

- Iteratively drop until no dead-end (after solving, nodes that got removed have their PageRank computed by summing over all predecessors PageRank divided by the number of successor of predecessor).
- Taxation (randomly teleport to another page). $v' = \beta Mv + (1-\beta)\frac{e}{n}$.

7.1.4 Efficient Computation of PageRank

7.1.4.1 Representing Transition Matrices

Transition matrix column can be represented by

- one integer: out degree
- one integer per nonzero entry in that column: the row numbers

7.1.4.2 Traditional Computation

One iteration of PageRank is $v' = \beta Mv + (1-\beta)\frac{e}{n}$. If v can fit into main memory, then this is just a Matrix vector multiplication in Map Reduce. Otherwise, we can use the stripping technique discussed in the intro section.

7.1.5 Use Combiners to Consolidate the Result Vector

In certain circumstances we wish to add terms for v'_i, the $i-th$ entry of the Mv. However, that term most likely is not available in main memory because:

- In the stripping technique, any vertical strip of M + horizontal strip of v will contribute to all components of the result vector v'
- M is stored column-by-column. But one column can affect any of the component of v'

Alternatively, we can partition the matrix into k^2 blocks while the vectors are still partitioned into k strips. E.g.: Let $M = 4 \times 4$ and let $k = 2$. The blocks of M are $M_{11}, M_{12}, M_{21}, M_{22}$

$$\begin{bmatrix} \begin{bmatrix} m_{11} & m_{12} \\ m_{21} & m_{22} \end{bmatrix} & \begin{bmatrix} m_{13} & m_{14} \\ m_{23} & m_{24} \end{bmatrix} \\ \begin{bmatrix} m_{31} & m_{32} \\ m_{41} & m_{42} \end{bmatrix} & \begin{bmatrix} m_{33} & m_{34} \\ m_{43} & m_{44} \end{bmatrix} \end{bmatrix} * \begin{bmatrix} \begin{bmatrix} v_1 \\ v_2 \end{bmatrix} \\ \begin{bmatrix} v_3 \\ v_4 \end{bmatrix} \end{bmatrix} = \begin{bmatrix} \begin{bmatrix} M_{11}v_1 + M_{12}v_2 \end{bmatrix} \\ \begin{bmatrix} M_{21}v_1 + M_{22}v_2 \end{bmatrix} \end{bmatrix}$$

Now, using k^2 Map tasks, each task gets one square of the matrix M, and one strip of the vector v. Notice that all the terms generated from M_{ij} and v_j contribute to v'_i and no other stripes of v'
For each block, we need to store

- columns that have at least one non-zero entry
- list those rows that have a nonzero entry
- outdegree for the column

7.1.6 Biased Random Walk

Only teleport to a subset of pages

$$v' = \beta M v + (1 - \beta)\frac{e_s}{|S|}$$

7.2 Link Spam

A spam farm can be constructed by pointing accessible pages to the target page and set up assisting pages for the target pages. The architecture is figure 7.1. Let there be n pages on the web in total. Let t be the target page, and have

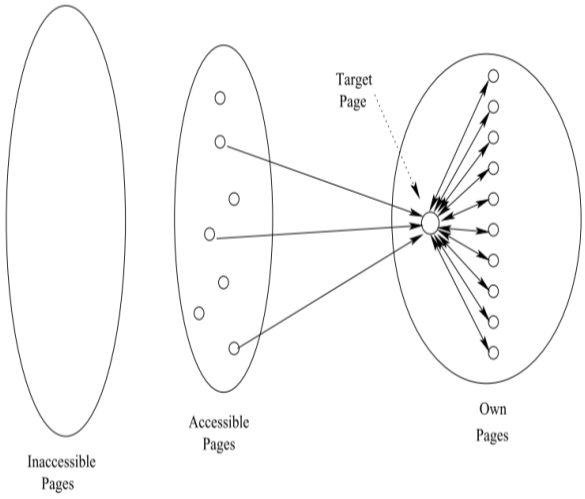

Figure 7.1: Spam Farm Architecture

m supporting pages. Let x be the amount of PageRank contributed by the accessible page. Let y be the PageRank of the target page. The PageRank of

each supporting page is

$$\beta \frac{y}{m} + \frac{1-\beta}{n}$$

The PageRank y of target t have 1) Contribution x from outside, 2) β times the PR of eary supporting page, 3) $(1-\beta)/n$ from taxation. 3 is normally so small that it doesn't matter. Combining 1 and 2 and solve for y we get

$$y = \frac{x}{1-\beta^2} + \frac{\beta}{1+\beta}\frac{m}{n}$$

If $\beta = 0.85$, we can see that y now gets $\frac{1}{1-0.85^2}$ times boost to the original true contribution x, in addition to bonus PageRank determined by the fraction $\frac{m}{n}$. Two ways to combat SpamFarm is

- TrustRank: Use a trust worthy (human examination / trust worthy domain) teleport set.
- Spam Mass: $p = \frac{r-t}{r}$ where t is the trust rank and r is the page rank. If spam mass is close to 1 or high, eliminate the pages.

7.3 Hubs and Authorities

Each web page has two scores:

- h: hubbiness of a page (how many good content this page links to).
- a: authority of a page (the quality of the content of this page)

The link matrix is an $n \times n$ matrix where $L_{ij} = 1$ if there is a link from page i to page j. The computation process is

1. Initiate $h = 1$
2. Compute $a = L^T h$, and scale so the largest component is 1
3. Compute $h = La$, scale so the largest component is 1
4. Repeat last two steps.