

TEL-AVIV UNIVERSITY
RAYMOND AND BEVERLY SACKLER
FACULTY OF EXACT SCIENCES
SCHOOL OF COMPUTER SCIENCE

Algorithms For Finding the Optimal k -Line-Means

Thesis submitted in partial fulfillment of the requirements for the
M.Sc. degree in the School of Computer Science, Tel-Aviv University

by

Dan Feldman

The research work for this thesis has been carried out at
Tel-Aviv University
under the supervision of Prof. Amos Fiat

December 2003

“Don’t accept any claim just because you heard it, or because it was passed on to you by tradition, or because it means the same as holy words, or because it is nice to hear and pronounce it, or because the person who says it has an appealing and charismatic personality, or because you honor the words of a particular teacher. Just when you yourself, as a consequence of your own experience, see for yourself that a certain saying or words of the teacher are worthy, or lead to good, are faultless, are accepted and admired by the wise, or following them has advantages and leads to happiness — then and only then, embrace it to yourself”

Buddha, “Visuddhi-Magga” (chap. ii.).

Acknowledgement

I would like to thank Prof. Amos Fiat for helpful discussions and patience, and for guiding me through the writing of this work. The observation and mathematical proof of theorem 4.2.1 is due to Prof. Arik Tamir, who also helped me with relevant and helpful remarks. Many of the references mentioned in section 1.4 are due to the help of Prof. Daniel Halperin. Thanks also for Prof. Nahum Kiryati who explained to me the motivation for using the LMS criterion (section 1.5), and for the discussion of the problem from computer vision prospective.

Contents

| | | |
|----------|---|-----------|
| 1 | Introduction | 1 |
| 1.1 | Abstract | 1 |
| 1.2 | Preface | 1 |
| 1.3 | Overview | 4 |
| 1.4 | Previous Work | 7 |
| 1.5 | Motivation for the LMS Criterion | 11 |
| 2 | 1-Line-Mean | 14 |
| 2.1 | The problem | 14 |
| 2.2 | Algebraic representation | 14 |
| 2.3 | The distance from a line | 15 |
| 2.4 | The center of gravity | 16 |
| 2.5 | Finding the 1-Line-Mean | 17 |
| 3 | Fast Updating the 1-Line-Mean | 19 |
| 3.1 | Adding new points to the set | 20 |
| 3.2 | Removing existing points from the set | 21 |
| 4 | 2-Line-Means | 22 |
| 4.1 | Decision Bounders | 22 |
| 4.2 | From lines to partitions | 22 |
| 4.3 | The Solution Paradigm | 24 |
| 4.4 | Naive Algorithms | 25 |
| 4.5 | $O(n^3)$ Possible Partitions | 26 |
| 4.6 | An $O(n^3)$ Solution | 29 |
| 5 | Data Structures | 31 |
| 5.1 | The Structure \mathcal{L} | 31 |
| 5.2 | The Structure \mathcal{F} | 33 |
| 5.3 | The Inner Structure $\Delta\mathcal{O}$ | 35 |
| 5.4 | The Structure \mathcal{O} | 38 |

| | | |
|----------|---------------------------------------|-----------|
| 6 | The Algorithm | 41 |
| 6.1 | Pseudo-Code | 41 |
| 6.2 | Description | 42 |
| 6.3 | Time Complexity | 44 |
| 7 | Generalizations | 45 |
| 7.1 | k -Line-Means | 45 |
| 7.2 | Weighted Axis | 45 |
| 7.3 | Different Norms | 47 |
| 7.4 | Weighted Points | 48 |
| 8 | Further Work and Open Problems | 51 |

Appendices

| | | |
|----------|---|-----------|
| A | Formal Proofs | 53 |
| A.1 | General Algebraic Review | 53 |
| A.2 | 1-Line-Mean | 54 |
| A.3 | Fast Updating the 1-Line-Mean | 61 |
| A.4 | 2-Line-Means | 65 |

Chapter 1

Introduction

1.1 Abstract

This work describes an algorithm for finding two straight lines in the plane such that the sum of squared Euclidean distances from every given point to its nearest line is minimized. The algorithm runs in time $O(n^3)$. We further generalize the algorithm for k -lines. We also show how to generalize the algorithm for use with weighted input points and different distance functions with no additional time complexity.

1.2 Preface

Although we don't know of any algorithm that solves the k -line-means problem as described in this section, there are many heuristic methods that try to solve it (see section 1.4, "Previous Work"). This fact is probably due to the importance of this problem in many areas. In the computer vision field, for example, it is one of the most fundamental problems [40]. This section tries to describe in simple words the problem we are trying to solve, and the motivation for solving it. Many of the claims will be explained or proved more formally in the next chapters.

The 1-line-mean is defined as follow:

Given a set of n points in the plane, the 1-line-mean is the line that will minimize the sum of squared distances from each point to that line.

Suppose that we made some experiments in the lab and wrote down the results as points on xy coordinates. Even if we believe that the phenomenon we are dealing with is linear, which means that all the points should be on the same line, it will probably will not be the case due to errors in the experiments. Our target is to find out what is the line that most likely represents the real phenomena. We will refer to this line as the "1-line-mean".

Figure 1.1 on page 2 shows a set of points, and a line that looks like a good guess. If the line represents the real phenomena, then the lengths d_1, d_2, \dots in the figure represent the errors of the experiments.

Another example relates to the OCR field: recognizing handwritten letters. If we will try to approximate the black pixels of an image with the letter ‘l’ on it, it is reasonable to assume that the 1-line-mean will match the points with much smaller error than pixels that form the letter ‘O’. This is because most of the pixels will be far from the line, regardless of its slope. In this case, the error of the approximated 1-line-mean can help us decide whether the given letter is ‘l’ or ‘O’.

It seems reasonable that every error in the experiment will increase the distance between the point and the original line, and therefore we will refer to that distance as the “error”. Why minimize the squared distances between the points and the line, and not, for example, just the sum of distances, or their median? this question is answered in section 1.5.

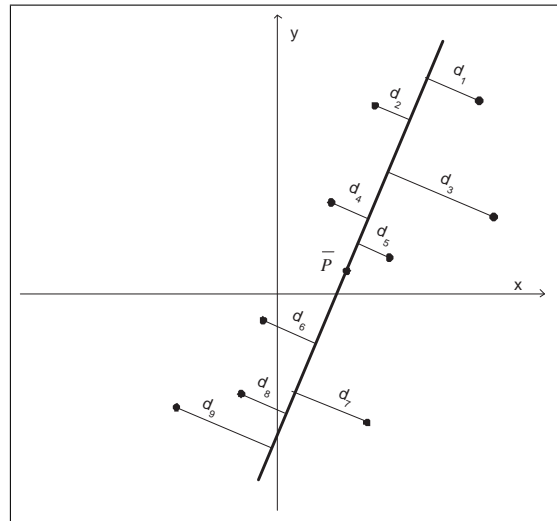


Figure 1.1: The 1-line-mean of a set of points.

The solution to this problem is about 100 years old and is used in an enormous number of fields: Image Processing, Data Mining, Text Mining, Graphics, Neuroscience, Statistics, Computer Vision, and many more. This is also the reason that the 1-line-mean has many different names: Linear Regression, Total Least Squares, Least Mean Squares, Principle Components Analysis (PCA), Latent Semantic Analysis (LSI), KL-Transform, etc. The reason for its vast popularity is probably that it solves quite efficiently a

basic problem in pattern recognition: Finding a linear phenomenon based on data with noise. One of the disadvantages of this algorithm is its global optimization. We are looking for **one** line that will approximate **all** the points. However, what if our inputs emerged from more than one phenomena?

If we believe, like in the last examples, that every point emerged from exactly one phenomena with a possible noise, and we want to find the two lines that are the most likely to create or approximate our points, we have the following changes to our problem:

*Given a set of n points in the plane, find **two** lines that will minimize the sum of squared distances from each point to its **nearest** line.*

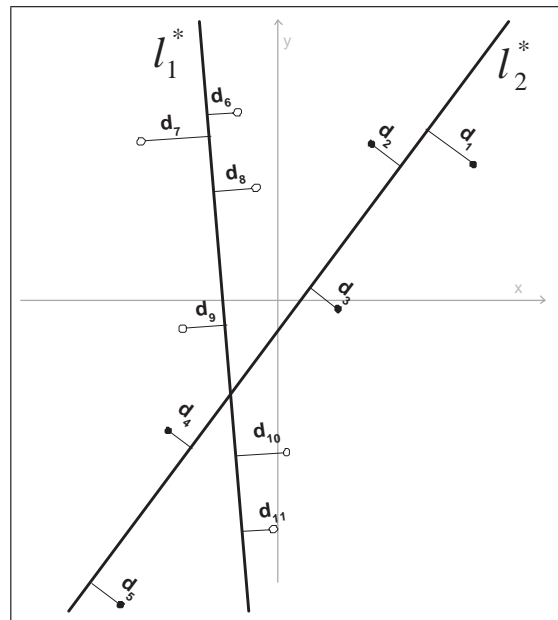


Figure 1.2: The 2-line-means of a set of points.

For example, if we know that the experiments (points) are mixed, and emerged from two independent linear patterns, and we have no idea which sample (point) belongs to which pattern. What is the pair of lines that is most likely to represent these two patterns? Back to our OCR example, let's examine an image with the letter 'x'. The sum of errors from the 1-line-mean computed from the letter 'x' will resemble the sum of errors for the 1-line-mean of the letter 'o'. This is because most pixels in both of the letters does not located on a single line. Clearly distinguishing between

those letters using the 1-line-mean will be difficult. However, we can use the fact that the pixels of the letter ‘x’ form two lines. Using the 2-line-means on the letter ‘x’ will therefore result in a small sum of errors. This sum of errors will be distinctively smaller than the sum of errors for the letter ‘o’.

Notice that although the 1-line-mean is the same as the first principle component returned by the PCA method, the other Principle Components usually have no connection with the k -line-means, and methods like simple PCA cannot solve the above problem. On the plane, for example, the PCA method will always return two orthogonal vectors: the first is the 1-line-mean and the second is its orthogonal. This is because, unlike the k -line-means algorithm, the PCA method does not cluster the points into groups, but tries to find a single orthogonal base that will be used for approximating all of them.

For the 1-line-mean we defined the error of a point to be the distance to that line. For the 2-line-means we assume that every point belongs to one of the line patterns and we define the error as the distance to the closer line. If we believe that the point was supposed to be on one of the lines, but moved due to noise, it is most likely (see section 1.5) to assume the point belongs to the closer line and define the error as the distance from it. This is why the term “nearer” is used in the new problem definition.

A solution to this problem is presented in this work. We generalize the case to k lines (see section 7.1) and call them the k -lines-means. The applications of this solution are relevant whenever there is more than one pattern in the observed data. It can also be used rather than the heuristics that are used to solve this problem and mentioned in section 1.4. Finally, the suggested clustering to lines can be more suitable for certain tasks than the known methods of clustering to k -centers. Such tasks are “features selection” and partitioning samples to multiple classifiers.

We called this problem the “2-line-means”, because, like the famous “ k -means” problem (see section 1.4), we want to cluster the data around two “means”. The difference is that our requested means are not points, but lines. The importance of this problem effected huge number of heuristics in many areas. However, the solution provided in this work is hereby proved to be optimal.

1.3 Overview

The rest of this work is self-contained in the sense that for every theorem (except 2.5.1), we give a formal proof in appendix A. For completeness

we added sections 1.5-3. Although most theorems described in those sections are well known, their explanations in text books often contain statistical terms (like covariances), geometric interpretation, or assume previous knowledge. The theorems in those sections were rewritten to contain only pure algebraical terms, using special definitions and symbols that will be needed for the following sections.

Section 1.5 explains the motivation for using the sum of squared distances (Least Mean Squares) as the criterion for the approximation. This is done using the “Maximum Likelihood” statistical criterion.

Section 2 presents the algorithm for finding the 1-line-mean, which is the line that minimizes the sum of squared distances from every point to it, as shown in Figure 1.1 (2). This line is also known as the first “Principle Component” (PC) or the Regression Line. In this section we also introduce some definitions that will be used in following sections. This is because the algorithm for solving the 2-line-means is based upon the 1-line case.

Section 3 explains how to update the 1-line-mean after we add or remove a set of points to the original set. This on-line update takes $O(|S|)$ time, where S is the group of added/removed points. The special case of adding or removing a single point from the set will be used in the final 2-line-means algorithm.

Section 4 describes the observation and preprocessing needed for solving the 2-line-means. As far as we know, these observations and proofs appear here for the first time.

Section 4.1 explains the basic observation that will be used for the theorems and descriptions in the following sections.

In Section 4.2 we define the 2-line-means as ℓ_1^* and ℓ_2^* and the subset of points that are closer to each by S_1^* (white) and S_2^* (black), respectively, as shown on Figure 4.1 (page 23). We show that ℓ_1^* is actually the 1-line-mean of the white points and the same for ℓ_2^* and the black points. It follows that to compute the 2-line-means it is enough to find the “right” partition of the points into two subsets. After calculating the 1-line-mean for each subset of this partition, we would get the desired 2-line-means.

In Section 4.3 we give an overview of the algorithm. This is done by iterating through all the possible partitions of the points into two subsets, knowing that one of them is the “right” partition of the 2-line-means, as described in section 4.2. For each partition we calculate

the 1-line-mean of each of its two subsets. By definition of the 2-line-means, the partition with the minimum sum of squares on these two lines must be the “right” partition of the 2-line-means. Next, we explain how to form the set of possible partitions, and how to iterate them efficiently.

In Section 4.4 we present some naive solutions for the problem in $O(n2^n)$ and $O(n^5)$ time complexity. The first algorithm uses the set of all possible 2^n partitions of n points into two subsets, and the second shows that the requested partition must be one of only $O(n^4)$ partitions. The second algorithm is based on the observation that the requested partition can be defined by two orthogonal lines — the bisector of the angles between the 2-line-means. Such lines will be called decision boundaries, and appear in boldface in Figure 4.1. This partition will remain the same after shifting each boundary right/down and rotating it until it meets two given points (see Figure 4.2 on page 25 and Figure 4.3 on page 26). As a result, the requested partition can be defined by two (no longer orthogonal) lines, each going through two given points. Because there are only $\binom{n}{2}\binom{n}{2}$ such pair of lines, we get a set of $O(n^4)$ partitions.

Both of these solutions calculate the 1-line-mean for each partition and its two subsets in $O(n)$ time.

In Section 4.5 we show that the requested partition of the points for the 2-line-means must be one of only nN partitions ($N = \binom{n}{2}$). This is done by showing that the requested partition can always be defined by 2 orthogonal lines (called decision boundaries), where the first is going through two input points, and its orthogonal is going through one point. The orthogonal decision boundaries in Figure 4.1, that separate the black and white points of the 2-line-means are not going through any given point. However, after shifting and then rotating them together (see Figures 4.2 and 4.4 on page 25), we can see that the same partition can be defined by two **orthogonal** lines, that are going through the input points.

In Section 4.6 the geometrical observation from the last section is used to form a scheme for an $O(n^3)$ algorithm. Although directly calculating the 1-line takes $O(n)$ for each partition, using the update algorithm of section 3 and the data structures of section 5 it takes only $O(1)$ for each partition.

In Section 5 we describe the data structures that will be used by the final algorithm. These data structures will not only be used to construct the nN possible partitions, but also to construct the partitions in an efficient order. The partitions will be generated in such an order that

most of the partitions differ from their successor by only one input point that moved from one subset to the other. This is the why we can use the results of section 3, and the number of possible partitions is the time complexity of the final algorithm.

In Section 6 we describe the pseudo-code for the final algorithm.

In Section 7 we generalize the results to k -lines, weighted points, and different distance functions.

In the Appendix all the theorems described on last sections are proven formally in their original order, together with formal definitions of the symbols and formulas in each section. The singular value decomposition is also explained shortly, together with related theorems and their proofs that are used for finding the 1-line-mean.

1.4 Previous Work

1.4.1 Projective Clustering

Although we didn't find in the literature any algorithms that solve the k -line-means problem as stated in this work, there are few articles about fitting lines/flats to data using the L_∞ distance, *i.e.* minimize the maximum distance from a point to a line/flat. This problem is called *projective clustering* [5]. From a geometric point of view, the *projective clustering* problem can be formulated as follows:

Given a set S of n points in \mathbb{R}^d and two integers, $k < n$ and $q \leq d$, find k q -dimensional flats h_1, \dots, h_k and partition S into k subsets S_1, \dots, S_k so that $\max_{1 \leq i \leq k} \max_{p \in S_i} d(p, h_i)$ is minimized.

That is, we partition S into k clusters and each cluster S_i is projected onto a q -dimensional linear subspace such that the maximum distance between a point, p , and its projection p' is minimized. The optimal value is denoted by w^* and is called the width of the set.

Projective clustering has recently received attention as a tool for creating more efficient nearest neighbor structures, as searching amid high dimensional point sets is becoming increasingly important; see [22] and references therein.

Meggido and Tamir [12] showed that it is NP-Hard to decide whether a set of n points in the plane can be covered by k lines. This implies not only that projective clustering is NP-Complete even in the planer case, but

also that approximating the minimum width within a constant factor is NP-Complete. Approximation algorithms for hitting compact sets by minimum number of lines are presented in [13].

For $q = d - 1$ and $k = 1$, the above problem is the classical *width problem*. The width of a point set can be computed in $\Theta(n \log n)$ time for $d = 2$ [14, 15], and in $O(n^{\frac{3}{2}+\epsilon})$ expected time for $d = 3$ [17]. Duncan et al. gave an algorithm for computing the width approximately in higher dimensions [16].

If $q = d - 1$, the above problem is equivalent to finding k hyper-strips that contain S so that the maximum width of a hyper-strip is minimized. if $q = 1$, then we want to cover S by k congruent hyper-cylinders of smallest radius.

The k -line-center problem is the projective clustering problem for $d = 2$ and $q = 1$. In the case where $d = k = 2$ and $q = 1$, the problem is called the 2-Line-Center [6]. This problem is usually described as follow:

Given a set S of n points in the plane, find two strips Δ_1 and Δ_2 that together cover the set S , such that the width of the larger strip is as small as possible.

Algorithms of running time $O(n^2 \text{polylog } n)$ for this problem can be found in [7, 10, 18, 3]. Agarwal and Sharir [7] presented for this problem an $O(n^2 \log^5 n)$ time algorithm, with an $O(n^2 \log^3 n)$ time decision algorithm and an extra factor of $O(\log^2 n)$ for the Megiddo [8] parametric optimization scheme. This result has been improved to $O(n^2 \log^4 n)$ by Katz and Sharir [9, 10] who applied an expander-based optimization approach with the decision algorithm of [7].

Later, Jaromczyk and Kowluk [11] improved this runtime to $O(n^2 \log^2 n)$ using special data structures and polar coordinates. Their data structure can be applied to a decision version algorithm that runs in time $O(n^2 \log n)$. Glozman et al. [6] used it in their algorithm as a subroutine in their optimization scheme and present another $O(n^2 \log^2 n)$ optimization algorithm. It is an open problem whether a sub-quadratic algorithm exists for this problem.

Approximations for the Two-Line-Center problem was suggested by Pankaj Agarwal [25]. The optimal width was denoted by w^* . They presented an algorithm that computes, for any $\epsilon > 0$, a cover of S by 2 strips of width at most $(1 + \epsilon)w^*$ in $O(n \log n + n/\epsilon^3 \log(1/\epsilon))$ time. Another algorithm pro-

posed an $O(n \log n)$ algorithm for this problem that covers S by two strips of width at most $3w^*$ [21].

On the planar case and $k > 2$ strips, w^* denotes the smallest value so that S can be covered by k strips, each of width at most w^* . As mentioned before, computing k strips of width of at most Cw^* , for any constant $C > 0$, that covers S , is known to be NP-Complete even when $d = 2$ [12].

Agarwal and Procopiuc [21] proposed a randomized algorithm that computes $O(k \log k)$ strips of width at most $6w^*$ that cover S , and whose expected running time is $O(nk^2 \log^4 n)$, if $k^2 \log k \leq n$. Their algorithm also works for larger values of k , but then the expected running time is $O(n^{2/3} k^{8/3} \log^4 n)$. Later, they proposed [23] an algorithm with near-linear expected running time that computes a cover by $O(k \log k)$ strips of width no larger than the width of the optimal cover by k strips.

For the case where $d > 2$, the strips are called hypercylinders and instead of ‘width’ the term ‘radius’ is used.

A greedy algorithm [19] is used to cover points by congruent q -dimensional hypercylinders. More precisely, if S can be covered by k hyper-cylinders of radius r , then the greedy algorithm cover S by $O(k \log n)$ hyper-cylinders of radius r in time $n^{O(d)}$.

The approximation factor can be improved to $O(k \log k)$ using the technique by Brönnimann and Godrich [20]. For example, when $q = 1$, $d = 2$, this approach computes a cover of S by $O(k \log k)$ strips of given width in time roughly $O(n^3 k \log k)$. The algorithm of Agarwal and Procopiuc [21] extends to covering points by hyper-cylinders in \mathbb{R}^d and to a few special cases of covering points by hyper-strips in \mathbb{R}^d . See also [24] for a recent improvement on the running time. Monte Carlo algorithms have also been developed for projecting S onto a single subspace [3].

Besides these results, very little is known about the projective clustering problem, even in the plane. However, because of its importance and vast application there are plenty of suggested heuristics from different areas. Heuristics for some variations of the projective clustering problem can be found in [30]. The most famous ones are probably the Independence Component Analysis (ICA) [29] and the Hough Transform [41].

1.4.2 Clustering by Point Centers

For a partition of an n -point set $S \subset \mathbb{R}^d$ into k subsets (clusters) S_1, S_2, \dots, S_k , we consider the cost function $\sum_{i=1}^k \sum_{x \in S_i} \|x - c(S_i)\|^2$, where $c(S_i)$ denotes the center of gravity of S_i . Clustering by the above function is frequently

used in the literature and in practical applications. In practice, mostly heuristic methods have been used, such as the *k-means algorithm* (local improvement of the current clustering by moving individual points among the clusters). Instead of minimizing the distance $d(p, p')$ from each point to its center, researchers have also tried to find clusters that preserve the inter-point distance within each cluster [3, 4], *i.e.* the distances between all pair of points in a cluster.

Algorithms with performance guarantees for k -clustering in \mathbb{R}^d have been considered by Inaba, Katoh, and Imai [28]. They observed that the number of distinct Voronoi partitions of a given n point set $S \subset \mathbb{R}^d$ induced by k points c_1, c_2, \dots, c_k is at most $O(n^{kd})$, and they can be enumerated in $O(n^{kd+1})$ time. Consequently, the optimum k -clustering under the variance-based cost defined above can be found in time polynomial in n , for any fixed d and k .

Hasegawa et al. [27] suggested an algorithm that in $O(n^{k+1})$ time can find a 2-approximately optimal k -clustering. They used the observation that choosing the input points as the centers of the clusters can always resolve in such approximation. For an arbitrary fixed k , and a fixed ϵ , Matousek [26] got an $O(n \log^k n)$ algorithm that finds partition with cost no worse than $(1 + \epsilon)$ -times the minimum cost.

For the case $k = 2$ Matousek [26] shows an algorithm that finds a 2-clustering with cost no worse than $(1 + \epsilon)$ -times the minimum cost in time $O(n \log n)$ with the constant of proportionality depends polynomially on ϵ .

In case there is a need to limit the minimum number of samples in each cluster Inaba, Katoh, and Imai [28] presented a randomized algorithm. More precisely, let $\epsilon > 0$ and $s \in [1, n]$ be parameters. Their algorithm finds, with probability at least $\frac{1}{2}$, a 2-clustering for which the cost is no worse than $(1 + \epsilon)$ -times the cost of any 2-clustering with cluster size at least s . The running time is $O(nm^d)$, where m is of the order $\frac{n}{\epsilon s} + \frac{n}{s} \log \frac{n}{s}$.

In the metric k -median problem, we are given n points in a metric space and select k of these to be cluster centers, and then assign each point to its closest selected center. If point j is assigned to the center i , the cost incurred is proportional to the distance between i and j . The goal is to select the k centers that minimize the sum of the assignment costs. These centers are called medians. The median is essentially the discrete analog of the centroid, and is also called the medroid [33]

Heuristics for the k -median problem with an approximate factor of 5 was suggested by Arya et al. [42]. The first $O(1)$ -approximate k -median algorithm was given by Charikar et al. [32] with a $6\frac{2}{3}$ approximation, and for general metric space. Subsequently, there have been several improvements

to this approximation. They also suggested a randomized constant-factor approximation. This constant depends on the ratio between the maximum and minimum interpoint distances.

Lin & Vitter [35] gave a polynomial time algorithm that finds, for any $\epsilon > 0$, a solution for which the objective function value is within a factor of $1 + \epsilon$ of the optimum, but it uses $(1 + 1/\epsilon)(\ln n + 1)k$ cluster centers. Later, they showed that by increasing the number of centers by a factor of $(1 + \epsilon)$, one could obtain a solution whose cost was at most $2(1 + 1/\epsilon)$ times the cost of the optimal solution (with at most k centers) [34]. For the k -median problem in 2-dimensional Euclidean metric, Arora, Ragahvan & Rao show a polynomial-time approximation scheme in [37]. Lin & Vitter also provided evidence that their result is best possible via a reduction from the set cover problem.

1.5 Motivation for the LMS Criterion

In this work we use the LMS (Least Mean Squares) distances as the criterion for the approximation error. That is, the error is considered the sum of squared Euclidean distances from each point to its nearest line. This criterion is used on many methods and algorithms, including the PCA. We could have used other criteria and distance functions. For example, the absolute sum of distances, the median distance, or consider the error as the largest distance from a point to its nearest line (see ‘Two-Line-Centers’ in section 1.4). This section we describe why this criterion might be more useful

The common reason people suggest for using the LMS criterion is that it is easy to work with. For example, it is easy to calculate integrals over x^2 but it is less trivial to do this on $|x|$, or that the standard norm in linear algebra is the sum of squares (L_2). The problem with this explanation is, first, that there are many solutions to problems that minimize the absolute distances (L_1) which are not necessarily harder to compute. In addition, we usually want to approximate our data using the criterion that will result in the most reasonable approximation, not the one that is easiest to calculate. However, the LMS criterion is useful, not only because of practical reasons, but also because of theoretical ones.

In the introduction we explained that for the 2-line-means problem we assume that our input points emerged from two independent lines. We also assumed that due to ‘errors’ (also called ‘noise’), the points were moved from their original lines. In many problems it is very natural to assume that the error for each point came from Gaussian distribution with zero mean. This means, for example, that the probability that a point will move

a distance right to its line, is the same as the probability it will move in the same distance to the left. It also means that it is very unlikely that the point will fall far from its line, but very likely that it will fall close to it. There are many reasons to believe that our error is Gaussian. Some of them:

- The Gaussian noise is considered by physicists as the distribution that represents many, if not most, of the noises in the world relative to their mathematical models, such as in electricity, communication, and optics.
- Practically, the Gaussian model is used in a vast number of fields such as: image processing, DSP, Data Mining, Biology, and has a large use on the industry.
- Usually, the best model that represents our intuition by asking for “approximation” is the Gaussian model. For example, if you try to draw a line that approximates the points in Figure 1.1 you will probably be drawing a line that is similar to the 1-line-mean. Criterion such as the minimum sum of absolute distances will result in several possible lines that far from this 1-line-mean and do not look at all like approximation in the usual sense.

After explaining the importance of the Gaussian model assumption, the rest of this section shows that the LMS criterion actually finds the most likely approximation under the assumption of Gaussian noise:

We define the distance of a given point, p_i , from the requested line, l , as $\text{dist}(p_i, l)$. By the assumption, this is a Gaussian random variable with an unknown standard deviation, σ , and a zero mean:

$$\text{Prob} \{ \text{dist}(p_i, l) = d_i \} = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-d_i^2/2\sigma^2} \quad (1.5.1)$$

For finding the line l which is the most likely to produce the given points, we should find:

$$\arg \max_{\ell} \text{Prob} \{ p_1, p_2, \dots, p_n \mid \ell \}$$

Using the last equation, this probability is equal to:

$$\arg \max_{\ell} \text{Prob} \{ \text{dist}(p_1, \ell) = d_1, \text{dist}(p_2, \ell) = d_2, \dots, \text{dist}(p_n, \ell) = d_n \mid \ell \}$$

Because we assume the points are independent, we can multiply their probabilities and search for:

$$\arg \max_{\ell} \prod_i \text{Prob} \{ \text{dist}(p_i, \ell) = d_i \mid \ell \}$$

And by (1.5.1), after removing the constant multiplier:

$$\arg \max_{\ell} \prod_i e^{-d_i^2/2\sigma^2}$$

\log is a monotonic function and we can use it without changing the result (argmax changed to argmin, instead of the minus sign):

$$\arg \min_{\ell} \sum_i d_i^2/2\sigma^2$$

Which gives us the requested result:

$$\arg \max_{\ell} \text{Prob} \{p_1, p_2, \dots, p_n \mid \ell\} = \arg \min_{\ell} \sum_i d_i^2$$

That is, the line that is most likely to represent the points under Gaussian error is the line that minimized the sum of squared distances (LMS) to the points. The proof is similar for k -lines, where $d_i = \min \{\text{dist}(p_i, \ell_1), \dots, \text{dist}(p_i, \ell_k)\}$.

Chapter 2

1-Line-Mean

This section explains how to find the 1-line-mean, which is the line that minimizes the sum of squared distances from all the input points to itself. We also define the terms that will be used in the next chapters.

2.1 The problem

Suppose we are given a set of points in the plane, p_1, p_2, \dots, p_n and we want to draw a new line. Assume also that we define “cost” or “error” as the squared distance from each point to that line. The total cost will be the sum of the squared distances from all the points to that line. In Figure 1.1 (page 2) this is $d_1^2 + d_2^2 + d_3^2$.

The 1-line-mean, l^* is the line that will give us the minimum error or cost we can get, given the set of points. Formally:

$$l^* = \arg \min_{\ell} \sum_{i=1}^n \text{dist}^2(\ell, p_i)$$

where $\text{dist}(\ell, p)$ is the distance between ℓ and the point p .

This total cost is called LMS, the “Least Mean Square”. The motivation for choosing that cost function, and not, for example, the sum of distances from all the points to the line (without taking the square root) is given in section 1.5. The solution to this problem can be found in [38], and will be explained in this section.

2.2 Algebraic representation

To solve this problem we will redefine it using linear algebra, to get a simple eigenvalues problem.

Every point, p_i , in the plane can be represented as a 2-d column vector whose entries are the coordinates of that point. That is, the point p_i will actually be a column vector $p_i = \begin{pmatrix} x_i \\ y_i \end{pmatrix} \in \mathbb{R}^2$, and this is the way we will refer to it from now on. Similarly, a set of points, S , will be treated as a $2 \times n$ matrix whose columns represent points.

For defining a line, ℓ , notice that we need only two properties: the direction of the line (its slope), that will be denoted by a unit vector $v \in \mathbb{R}^2$, and its distance from the origin, $c \in \mathbb{R}$ (see Figure 2.1 on page 16). We will denote by v_\perp the vector orthogonal to a unit vector, v :

$$v_\perp \stackrel{\text{def}}{=} \begin{pmatrix} v_y \\ -v_x \end{pmatrix} \quad \text{where} \quad v = \begin{pmatrix} v_x \\ v_y \end{pmatrix}. \quad (2.2.1)$$

Notice that v_\perp remains a unit vector, and represents v after a rotation of 90° clockwise.

Every point, p' , on the line can be represented as a combination of two orthogonal vectors, which form a base for \mathbb{R}^2 : v_\perp and v . It is easy to prove (appendix A) that the projection of any point, p , on any unit vector, v , is just their inner product, $v^t p$, as denoted in Figure 2.1:

Theorem 2.2.1. *The projection's length, $k \in \mathbb{R}$, of any vector, $p \in \mathbb{R}^2$, on a unit vector $v \in \mathbb{R}^2$ is $\langle v, p \rangle = v^t p$.*

In Figure 2.1 we can see that the projections for all the points on ℓ have the same length, c , on the vector v_\perp , and the difference between the points is only their projection on v (denoted by k). This gives us the next definition of a line, as the set of points we can get by “walking” c units in the direction of v_\perp and any additional number of units in the direction of v :

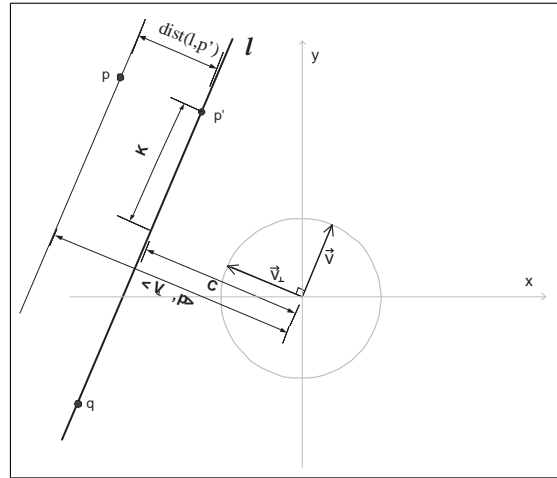
$$\ell = \text{line}(v, c) = \{cv_\perp + kv \mid k \in \mathbb{R}\}. \quad (2.2.2)$$

2.3 The distance from a line

We define the distance from a point, p , to a line, ℓ to be its shortest distance from any point on that line, and noted by $\text{dist}(\ell, p)$ as demonstrated in Figure 2.1 (page 16):

$$\text{dist}(\ell, p) \stackrel{\text{def}}{=} \min_{p_L \in \ell} \|p - p_L\| \quad (\text{A.2.4})$$

One of the reasons we picked the above definition of a line and not, for example, $y = ax + b$, is that giving the vector v and the number c , it is easy to find the distance from any point p to the line $\ell = \text{line}(v, c)$:

Figure 2.1: The line $\ell = \text{line}(v, c)$.

Theorem 2.3.1. *The shortest Euclidean distance between the line l and the point p is:*

$$\text{dist}(\ell, p) = |v_{\perp}^t p - c| \quad (2.3.1)$$

where $\ell = \text{line}(v, c)$.

For completeness, we show the proof in appendix A and shown with the point p in Figure 2.1. As expected, all the points on the line itself are with zero distance:

$$v_{\perp}^t p' = c. \quad (2.3.2)$$

2.4 The center of gravity

The average vector, \bar{p} , of n given points, which is also known as the “center of gravity” is defined as:

$$\bar{p} \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n p_i.$$

As demonstrated in Figure 1.1, this point has a useful property: the requested 1-line-mean must go through it. We actually prove something stronger than that. Suppose that we have a line, ℓ , with any given slope/direction (defined by a unit vector “ v ”), but we can put it as far as we want from the origin (“ c ”). The next theorem says that if we want to minimize the sum of squared distances from all the points to that line we should shift it until we reach point \bar{p} .

Recall from (2.3.2) that \bar{p} is on the line iff $c = v_{\perp}^t \bar{p}$.

Theorem 2.4.1. *for any fixed unit vector v :*

$$\arg \min_c \sum_{i=1}^n \text{dist}^2(\text{line}(v, c), p_i) = v_{\perp}^t \bar{p}.$$

In particular, given only the v value of the 1-line-mean, we can immediately calculate $c = v_{\perp}^t \bar{p}$. By the theorem, we cannot choose other c that will decrease the sum of squared distances.

In other words, if we set the point \bar{p} as our new origin $(0, 0)$, we know that the line we are looking for is a line that goes through the origin, *i.e.* after changing our coordinate system, its distance from the origin is zero:

$$\text{line}(v, c) = \text{line}(v, 0) = kv, \text{ for some } k \in \mathbb{R} \quad (2.4.1)$$

To accomplish this, all we have to do is to subtract the vector \bar{p} from each point. After finding the 1-line-mean in the new system, we can retrieve the original line just by shifting it back from the origin $((0, 0))$, that is, set $c = v_{\perp}^t \bar{p}$ instead of $c = 0$.

2.5 Finding the 1-Line-Mean

This section explains how to find the line that minimize the LMS, among all the lines that go through the origin $((0, 0))$. This means that if we shift all the points to the center of gravity, as described in last section, this line must be the 1-line-mean we are looking for.

Suppose the original points were p_1, p_2, \dots, p_n , then their shifted coordinates will be $p_1 - \bar{p}, p_2 - \bar{p}, \dots, p_n - \bar{p}$. Since we are looking for a line that goes through the origin ($c = 0$), the line can be written as $\text{line}(v, 0)$, and by (2.3.1) the squared distance from a point p_i to that line will be the square of:

$$d_i = \text{dist}(p_i, \ell) = v_{\perp}^t (p_i - \bar{p}) \quad (2.5.1)$$

Let us build a matrix with $p_i - \bar{p}$ as its i 'th row vector and call it \tilde{A} . If we multiply this matrix by the unit vector v_{\perp} , using last equation we'll get:

$$\tilde{A}v_{\perp} = \begin{pmatrix} d_1 \\ d_2 \\ \vdots \\ d_n \end{pmatrix}$$

Notice that taking the square root of the norm of this vector gives: $d_1^2 + d_2^2 + \dots + d_n^2$. This means that the squared sum of distances of a given line,

$\ell = \text{line}(v, 0)$, which is going through the origin in the direction of v is:

$$\sum_{i=1}^n \text{dist}^2(\ell, p_i) = \|\tilde{A}v_{\perp}\|^2$$

Now we have a pure algebraic problem.

Given the matrix \tilde{A} , find a unit vector, v_{\perp} , that minimizes $\|\tilde{A}v_{\perp}\|$:

$$v_{\perp} = \arg \min_{\|v\|=1} \|\tilde{A}v\| \quad (2.5.2)$$

This is an eigenvalue problem. To solve it we use the next general theorem, which is based on the singular value decomposition (see theorem 2.5.1 in the appendices):

Theorem 2.5.1. *Every real matrix $M_{m \times n}$ can be written in its unique SVD form which is:*

$$M = U\Sigma V$$

with $U_{m \times m}$ and $V_{n \times n}$ as orthogonal matrices, and $\Sigma_{p \times p}$ as a diagonal matrix such that $p = \min\{m, n\}$ and the values along the diagonal of Σ are in non-decreasing order.

In particular, the theorem says that we can describe our $2 \times n$ matrix, \tilde{A} , as the multiplication of 3 matrices: $\tilde{A} = U\Sigma V$, where U and Σ are of size 2×2 . We denote the two orthogonal row vectors of U by $\vec{\sigma}_{max}$ and $\vec{\sigma}_{min}$, and the first value of the diagonal matrix, Σ , by σ_{min} . Using this new notation, we have a solution to our problem, as proved in appendix A:

Theorem 2.5.2.

$$\arg \min_{\|v\|=1} \|\tilde{A}v\| = \vec{\sigma}_{min}$$

This means that the solution to (2.5.2) is the vector $v_{\perp} = \vec{\sigma}_{min}$, which means $v = \vec{\sigma}_{max}$. After finding v for the 1-line-mean, l^* , we can find c by theorem 2.4.1.

All together we have:

Theorem 2.5.3. *Let*

$$l^* = \arg \min_{\ell} \sum_{i=1}^n \text{dist}^2(\ell, p_i)$$

then $l^* = \text{line}(\vec{\sigma}_{max}, \vec{\sigma}_{min}^t \bar{p})$,
and

$$\min_{\ell} \sum_{i=1}^n \text{dist}^2(\ell, p_i) = \sigma_{min}^2$$

$$\sigma_{min}^2 = \sum_{i=1}^n \text{dist}^2(l^*, p_i)$$

Chapter 3

Fast Updating the 1-Line-Mean

Suppose we add or remove a point from the set of points, and want to know how good is the new 1-line-mean. This section shows a simple way to get the new LMS value immediately, *i.e.* in $O(1)$ time. This method will be used to find the 2-line-means efficiently. Notice that here we are interested only in the LMS value of the 1-line-mean, and not the line itself.

As was shown in the last section the LMS distance from all the points to the 1-line-mean is σ_{min}^2 , as defined above. This value is actually the smaller between the two eigenvalues of $\tilde{A}\tilde{A}^t$, denoted by $\lambda_{min}(\tilde{A}\tilde{A}^t)$, and the next theorem shows that it depends on only 6 values:

$$\begin{aligned}u_1 &= n \\u_2 &= x_1 + x_2 + \dots + x_n \\u_3 &= y_1 + y_2 + \dots + y_n \\u_4 &= (x_1^2 - y_1^2) + (x_2^2 - y_2^2) + \dots + (x_n^2 - y_n^2) \\u_5 &= x_1y_1 + x_2y_2 + \dots + x_ny_n \\u_6 &= (x_1^2 + y_1^2) + (x_2^2 + y_2^2) + \dots + (x_n^2 + y_n^2).\end{aligned}$$

Formally, we will define the vector $\text{vec}(A)$ as the vector containing the first 5 values, for reasons explained later:

$$\text{vec}(A) \stackrel{\text{def}}{=} \begin{pmatrix} u_1 \\ u_2 \\ u_3 \\ u_4 \\ u_5 \end{pmatrix} \tag{3.0.1}$$

Using only these values we can calculate σ_{min} (which is $\lambda_{min}(\tilde{A}\tilde{A}^t)$):

Theorem 3.0.4.

$$\lambda_{\min}(\tilde{A}\tilde{A}^t) = -\frac{1}{2} \left(\hat{\lambda}(\text{vec}(A)) - \sum_{i=1}^n x_i^2 + y_i^2 \right) \quad (3.0.2)$$

where $\hat{\lambda}$ is a function from \mathbb{R}^5 to \mathbb{R} as follows:

$$\hat{\lambda}(u) \stackrel{\text{def}}{=} \frac{1}{u_1} (u_2^2 + u_3^2) + \left\| \begin{pmatrix} u_4 - \frac{1}{u_1}(u_2^2 - u_3^2) \\ 2 \left(u_5 - \frac{1}{u_1} u_2 u_3 \right) \end{pmatrix} \right\| \quad (3.0.3)$$

So eventually, to calculate how good is our 1-line, *i.e.* the sum of distances from it, all we have to do is substitute $\text{vec}(A)$ and u_6 in 3.0.2 and calculate the result.

Notice that:

- To compute $\text{vec}(A)$ and u_6 require $O(n)$ time.
- Given $\text{vec}(A)$ and u_6 we can compute $\lambda_{\min}(\tilde{A}\tilde{A}^t)$ in $O(1)$ using the above equation.

Although we need to loop over all the points to find $\text{vec}(A)$ and u_6 (that is, $O(n)$ time), calculating the value of this function is immediately (time $O(1)$) if we already have these values.

3.1 Adding new points to the set

Suppose we calculate the 1-line-mean of our set of points as described above, and save the value $u_6 = \sum_{i=1}^n (x_i^2 + y_i^2)$ together with $\text{vec}(A)$. Now someone adds a new point, $p_{n+1} = \begin{pmatrix} x_{n+1} \\ y_{n+1} \end{pmatrix}$ to the set, which means that there is a new 1-line-mean with new distances. To find the new sum of distances we can repeat the process again, and find the new values u'_1, u'_2, \dots, u'_6 in time $O(n)$. A better approach will be to represent these values using the values we already have:

$$\begin{aligned} u'_1 &= n + 1 = u_1 + 1, \\ u'_2 &= x_1 + x_2 + \dots + x_n + x_{n+1} = u_2 + x_{n+1}, \\ u'_3 &= y_1 + y_2 + \dots + y_n + y_{n+1} = u_3 + y_{n+1}, \\ u'_4 &= (x_1^2 - y_1^2) + (x_2^2 - y_2^2) + \dots + (x_n^2 - y_n^2) + (x_{n+1}^2 - y_{n+1}^2), \\ &= u_4 + (x_{n+1}^2 - y_{n+1}^2), \\ u'_5 &= x_1 y_1 + x_2 y_2 + \dots + x_n y_n + x_{n+1} y_{n+1} = u_5 + x_{n+1} y_{n+1}, \\ u'_6 &= (x_1^2 + y_1^2) + (x_2^2 + y_2^2) + \dots + (x_n^2 + y_n^2) = u_6 + (x_{n+1}^2 + y_{n+1}^2). \end{aligned}$$

The above equations show that given the original values, the new values of $\text{vec}(A)$ and u_6 are calculated using only six “add” operations, meaning

$O(1)$ time. As was explained in the last section, given these values we can calculate the new sum of LMS distances using theorem 3.0.4 in $O(1)$ time. Although this is enough for the 2-line algorithm that will be shown later, theorem 3.1.1 shows a generalization of the above equations, where we add more than one point. This simple theorem shows that if we already have the original six values, and add a group of points as the columns matrix B , all we need to know to compute the new 1-line-mean is $\text{vec}(B)$:

Theorem 3.1.1.

$$\text{vec}([A|B]) = \text{vec}(A) + \text{vec}(B)$$

For the algorithm to the 2-line-means we will only add single point, $p = \begin{pmatrix} x \\ y \end{pmatrix}^t$, to the set in each step:

$$\text{vec}([A|B]) = \text{vec}(A) + \text{vec}\left(\begin{pmatrix} x \\ y \end{pmatrix}\right).$$

3.2 Removing existing points from the set

Given the values u_1, \dots, u_6 as before, we want to calculate the quality of the new 1-line-mean after removing one point, $p_j = (x_j, y_j)^t$ where $1 \leq j \leq n$. This is similar to adding a new point:

$$\begin{aligned} u'_1 &= u_1 - 1, \\ u'_2 &= u_2 - x_j, \\ u'_3 &= u_3 - y_j, \\ u'_4 &= u_4 - (x_j^2 - y_j^2), \\ u'_5 &= u_5 - x_j y_j, \\ u'_6 &= u_6 - (x_j^2 + y_j^2). \end{aligned}$$

After calculating these values (in $O(1)$ time) we can recalculate the sum of squared distances from the new 1-line-mean in $O(1)$ time, using theorem 3.0.4 as explained last section. Here, again, there is a generalization for removing a set of points, where the removed points are the columns of the matrix B , and $A \setminus B$ is the matrix after the changes:

Theorem 3.2.1.

$$\text{vec}(A \setminus B) = \text{vec}(A) - \text{vec}(B).$$

For the algorithm to the 2-line-means we will only remove a single point, $p = (x, y)^t$, from the set in each step:

$$\text{vec}([A|B]) = \text{vec}(A) - \text{vec}\left(\begin{pmatrix} x \\ y \end{pmatrix}\right).$$

Chapter 4

2-Line-Means

In this section we will present some geometrical observations that together with the data structures in the next section will be used to solve the 2-line-means problem in $O(n^3)$ time. The formal definitions of this section can be found in the appendix.

4.1 Decision Bounders

Figure 4.1 shows a set of points, S , and the 2-line-means marked by ℓ_1^* and ℓ_2^* . The points that are closer to ℓ_1 are white and the rest, who are closer to ℓ_2 , are black. We will call these groups S_1^* and S_2^* respectively.

Notice that we added two additional broken lines which cross exactly in the middle of the angles between ℓ_1^* and ℓ_2^* . We will call them decision bounders. It is easy to see that these decision bounders must be orthogonal to each other. Notice also that the distances to ℓ_1^* and to ℓ_2^* is the same for points that are exactly on the decision bounders, and that the decision bounders separate the points to four orthogonal groups: two white groups and two black groups of points. The upper-left and lower-right groups of white points will be called S_1^A and S_1^B respectively, as shown in Figure 4.1.

4.2 From lines to partitions

The next theorem tells us that if we have the right partition to white and black points (as described on previous section), finding the 2-line-means is straightforward: finding the 1-line-mean of the white points only will give us ℓ_1^* , and calculating the 1-line-mean of the rest of the points (the black ones) will give us the second optimal line, ℓ_2^* .

The geometrical explanation is that because any other line that will be used to approximate the white points cannot be better than their 1-line-mean, and the same for the black points, which means that this is the minimum sum

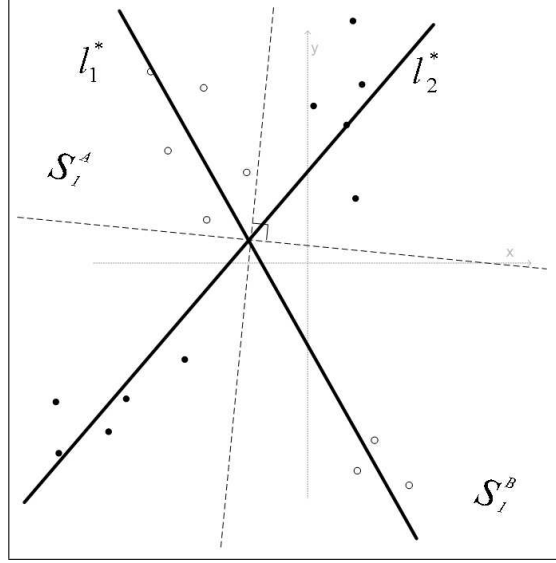


Figure 4.1: The 2-line-means and their bounders.

of squared distances for all the points. Any two other lines that partition the points into two other groups (not for white group and black group) cannot have better LMS distances because it contradicts our assumption that the given partitioning to white and black points is the partition of the 2-line-means. A more formal proof of this theorem is given in the appendix.

Theorem 4.2.1. *Pair of lines that solve the 2-line-means problem are the 1-line-mean of S_1^* , and the 1-line-mean of S_2^* . That is:*

$$(\ell_1^*, \ell_2^*) = (\ell'_1, \ell'_2)$$

where:

$$\ell'_1 = \arg \min_{\ell} \sum_{p \in S_1^*} \text{dist}^2(p, \ell) \quad \ell'_2 = \arg \min_{\ell} \sum_{p \in S_2^*} \text{dist}^2(p, \ell). \quad (4.2.1)$$

We can conclude that instead of searching for the 2-Lines, ℓ_1^*, ℓ_2^* , we can look for the partition of the set, S_1^* , and its complement. Because S_1^* is actually the union of S_1^A and S_1^B , the partition can also be defined by the two orthogonal decision bounders that separates them geometrically on the plane.

This means, that if we were only given the position of the decision bounders, we could find easily ℓ_1^* and ℓ_2^* and the sum of squared distances from them: first we would mark the groups S_1^A and S_1^B created by the bounders as S_1^* , then find the 1-line-mean of S_1^* with the sum of their squared distances,

as described in section 2.5. Applying the same for the complement group, S_2^* , will give us ℓ_2^* and the sum of squared distances from the second group to this line.

Adding the two sums will give us the total LMS distance from each point to its nearest line. If we have a candidate partition of A into two subsets, B and the rest of the points, we define G (Grade) to be this value:

$$G(A, B) \stackrel{\text{def}}{=} \sigma_{\min}^2(\widetilde{A \setminus B}) + \sigma_{\min}^2(\tilde{B}) \quad (4.2.2)$$

By its definition, the partition for the 2-line-means must have the minimum grade, which means, the minimum sum of two LMS distances.

4.3 The Solution Paradigm

In section 4.2 we saw that the 2-line-means are two lines, such that one of them is the 1-line-mean of subset of points, and the other line is the 1-line-mean of the rest. Since we already know (section 2.5) how to find the 1-line-mean of each subset and the sum of distances from this line, we have a simple algorithm to find the 2-line-means:

1. For every possible partition of the points into two subsets, S_1 and S_2 , do:
 2. Calculate the sum of distances from the 1-line-mean of S_1 , denote this value by s_1 .
 3. Calculate the sum of distances from the 1-line-mean of S_2 , denote this value by s_2 .
 4. Let $s := s_1 + s_2$.
 5. if $s < s^*$ then

$$\begin{aligned} s^* &:= s \\ S_1^* &= S_1 \\ S_2^* &= S_2 \end{aligned}$$
6. The 2-line-mean are the 1-line-mean of S_1^* and S_2^* .

The following algorithms will differ by the number of partitions they consider (step 1), and the time required to compute the values s_1 and s_2 (step 2 and 3).



Figure 4.2: After shifting the boundaries.

4.4 Naive Algorithms

We can use the fact that each of the boundaries can be shifted in a way that it will go through two points of the set. This can be done in two steps: first shifting the boundaries such that each line goes through one input point, and then rotate each one of them so as to touch a second point.

For the first step we move each of the boundaries to be a line that is parallel to the original boundary, but going through a point. This is easy to do by shifting one of the boundary down and the other to the right, until each one of them reach a point from the set (Figure 4.2). On the second step, we rotate each line around its point until it meets another point from the set, as shown in Figure 4.3. Notice, however that the resulting boundaries are no longer perpendicular, because of the last rotation.

We can conclude that there exist two boundaries such that each one of them goes through 2 points of the set. This means that we need to consider less than 2^n subsets, because the requested subsets are found between such 2 decision boundaries, and there are only $O(n^4)$ possible pair of such lines: $\binom{n}{2}$ for choosing the first line that will go through 2 input points, and then $\binom{n}{2}$ possibilities to draw the second line. If we calculate step 2 and 3 in $O(n)$ time we get polynomial running time $O(n^5)$ for finding the 2-line-means.

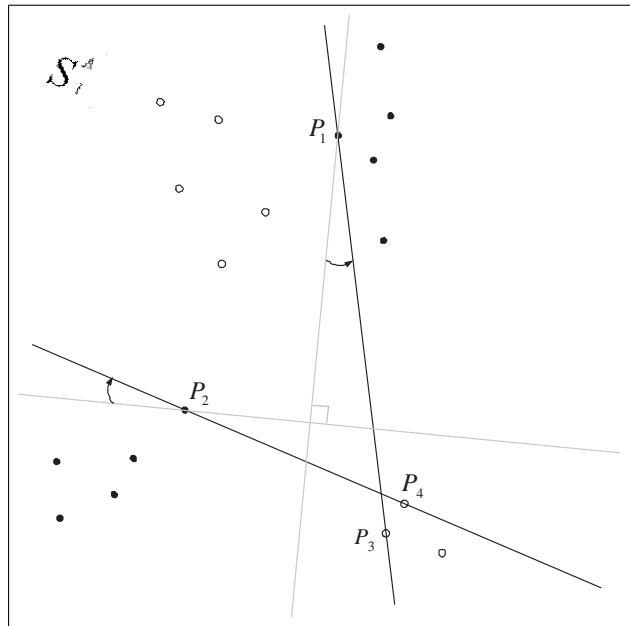


Figure 4.3: After Rotating each bounder to touch a second point.

4.5 $O(n^3)$ Possible Partitions

In the last section we showed that the subset S_1^* is the union of two groups, S_1^A and S_1^B , that are separated by the two orthogonal decision bounders.

In this section we will show that we can move the decision bounders to be not only orthogonal, but also going through points in S . Specifically, one of the lines, ℓ is connecting 2 points from the set, and its orthogonal is going through a third point, p_3 , as shown in Figure 4.4. We will do this by moving the decision bounders, while preserving their orthogonality and the four groups they are separating.

First we move each of the bounders to be a line that is parallel to the original bounder, but going through a point. This is easy to do by shifting one of the bounder down and the other to the right, until each one of them reach a point from the set (Figure 4.2). Notice that we didn't change the directions of the lines, so they are still orthogonal and also none of the points moved from one group to another.

For getting one of the lines to touch another point, notice that if we rotate these two lines simultaneously with the same angle — their orthogonality is preserved. So, we will rotate each line around the point it already

touches until the first one of them will touch another point.

On the special case where there are no points below the first bounder or right to the second one, we can move the bounder in the opposite direction. The above simulation shows that for any set of points we can conclude the next observation:

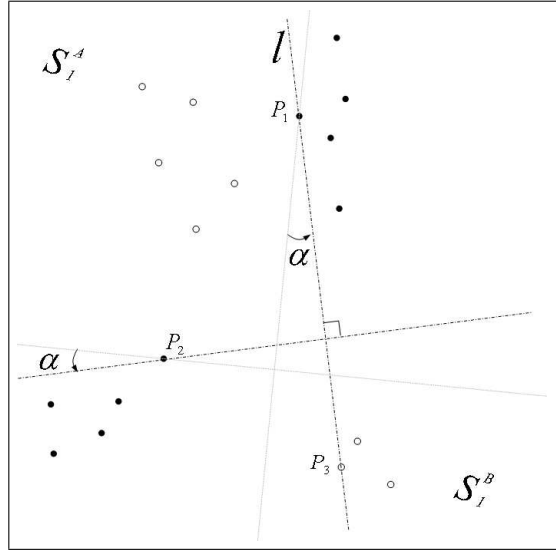


Figure 4.4: After Rotating each bounder while preserving their orthogonality.

Geometrical Observation. *For every set of points, there exists 2 decision bounders that separates the points into the two clusters of the 2-line-means, S_1^* and S_2^* , such that:*

1. *The two decision bounders are orthogonal.*
2. *One of the decision bounders goes through two input points.*
3. *The other decision bounders goes through an input point.*

This observation will be used later to reduce the number of possible partitions of the points, S_1^* and S_2^* . We can now represent the two subsets of each partition by a line that represents the first bounder, and an input point that goes through the second bounders. Formally, the white points in Figure 4.4 defined by:

$$S_1^A(\ell, p') \stackrel{\text{def}}{=} \{p \in S \mid v^t(p - p') \leq 0\} \cap \{p \in S \mid v_{\perp}^t p - c > 0\} \quad (4.5.1)$$

and the black points in the figures are:

$$S_1^B(\ell, p') \stackrel{\text{def}}{=} \{p \in S \mid v^t(p - p') > 0\} \cap \{p \in S \mid v_\perp^t p - c \leq 0\} \quad (4.5.2)$$

Notice that although the points of the two diagonal blocks (S_1^*) are always white, and the points on the other two blocks are always black, borderline points that are on the boundaries themselves (such as p_1 , p_2 and p_3 in the figure) can have any combination of colors. However, a borderline point that is above the line ℓ or on its left side must be black, and the opposite for a white one. This is due to the way we drag and rotate the boundaries from their original position. The “ $>$ ” and “ \leq ” on the formal definitions cause the union of the two groups to contain only white points (the group S_1^*), and all of them.

4.6 An $O(n^3)$ Solution

This section uses the main observations from last sections to suggest an algorithm that finds the 2-line-means in $O(n^3)$ time.

The observations that will be used on the next sections are:

1. We can find the LMS distances of n points from the 1-line-mean in $O(n)$ time (see section 2.5).
2. We can remove or add a point and find the new sum of LMS distances in $O(1)$ time (section 3).
3. The 2-line-means are two lines, such that one of them is the 1-line-mean of subset of points, and the other line is the 1-line-mean of the rest (section 4.2).
4. These two subsets are separated by two orthogonal lines: a line that going through two input points, and another that going through a third point (section 4.5).

Here we present a subset of only n^3 partitions that one of them is the right 2-lines partition. It is derived from the geometrical observation of last section and Figure 4.4. We have, again, $\binom{n}{2}$ possibilities for the first line, but because the second line is orthogonal to the first, all we have to find is the third point that it touches. There are n points, so the total number of partitions we should check is $O(n^3)$.

If we calculate step 2 and 3 in $O(n)$ we get a polynomial running time $O(n^4)$ for finding the 2-line-means. However, we can test the partitions on-line in time $O(1)$ and get an $O(n^3)$ time algorithm. Recall (section 4.2) that the requested partition must have the minimum $G(\cdot)$ value.

The following theorem tells us that to find the $G(\cdot)$ value of a partition we don't need to know exactly what are the points of all the set, and the points of one of the subset. The only information we need is the $\text{vec}(\cdot)$ value of this groups:

Theorem 4.6.1.

$$G(A, B) = \vec{G}(\text{vec}(A), \text{vec}(B)) + \frac{1}{2} \sum_{i=1}^n x_i^2 + y_i^2.$$

where $\vec{G}(u, w)$ is a function from two vectors, $u, w \in \mathbb{R}^5$ to \mathbb{R} :

$$\vec{G}(u, w) \stackrel{\text{def}}{=} -\frac{1}{2} \left[\hat{\lambda}(u - w) + \hat{\lambda}(w) \right]. \quad (4.6.1)$$

($\hat{\lambda}$ is a simple function on \mathbb{R}^5 that was defined in section 3 and definition (A.3.2)).

From the above theorem we learn that to find the desired partition (with the minimal grade), we can calculate \vec{G} for each candidate, B , of the $O(n^3)$ partitions, and choose the one with the smallest result:

Theorem 4.6.2. *If $\mathcal{S} \subseteq \{B|B \subseteq A\}$, which means that \mathcal{S} is any set of matrices that their columns are subset of A 's columns, then:*

$$\arg \min_{B \in \mathcal{S}} G(A, B) = \arg \min_{B \in \mathcal{S}} \vec{G}(\text{vec}(A), \text{vec}(B)).$$

Notice that we need to calculate $\text{vec}(A)$ only once for all the set of points, and \vec{G} can be calculated in $O(1)$ given $\text{vec}(A)$ and $\text{vec}(B)$. All we have to calculate are the possible values of $\text{vec}(B)$. The next theorem is based upon section 3 and explains how to calculate $\text{vec}(B)$ for each partition using the previous calculated value:

Theorem 4.6.3. *Assume that $B \subseteq A$, and suppose that we want to add the column vectors of B^+ to B and remove vectors that are included in B^- , that is:*

$$B' = ([B|B^+] \setminus B^-). \quad (4.6.2)$$

where B^+ and B^- are matrices with two rows each, and $B^- \subseteq B$.
Then:

$$\vec{G}(\text{vec}(A), \text{vec}(B')) = \vec{G}(\text{vec}(A), \text{vec}(B) + \text{vec}(B^+) - \text{vec}(B^-))$$

To calculate these values efficiently we will iterate through the $O(n^3)$ possible partitions, in an order such that on every iteration only one point will be added or removed. Section 5 describes how to construct such an order on the partitions. By theorems 3.1.1 and 3.2.1 we can track the changes of vec in such cases in $O(1)$. Doing so will result in an $O(n^3)$ time algorithm that finds the minimum LMS value of all these values. This algorithm is described in section 6.

Chapter 5

Data Structures

This chapter describes the data structures used by the final algorithm for finding the 2-line-means (section 6). Each data structure is constructed according to the specific set of points of our 2-line-means problem. Some of the data structures depend on the others, so the order of their construction and initialization should be as the order of their appearance on this section.

5.1 The Structure \mathcal{L}

5.1.1 Description

Let $\mathcal{L} = \{\ell \mid \ell \text{ goes through } p, q, \text{ where } p, q \in S\}$.

Let $\theta(\ell)$ be the slope of the line ℓ .

The data structure \mathcal{L} support two methods:

initialize(S) — prepares the set of lines L , using the given points set S .

Space Complexity: $\binom{n}{2} = O(n^2)$.

Time Complexity: $n^2 \log n$.

getLine(i) — returns ℓ such that $\theta(\ell)$ is the i 'th largest among all $\theta(\ell')$, $\ell' \in \mathcal{L}$.

Time Complexity: $O(1)$.

5.1.2 Implementation

\mathcal{L} represents the array of $N = \binom{n}{2}$ possible different lines, each goes through two input points. Each line $\ell_i \in \mathcal{L}, 1 \leq i \leq N$ is represented as a pair, (c_i, v_i) , of a real number and a unit vector such that $\ell_i = \text{line}(v_i, c_i)$, as defined in (A.2.3) and section 2.2.

However, we demand a specific order on these lines that will be used later in building $\Delta\mathcal{O}$ (section 5.3). The order of these lines will be ascending by their slopes. Formally, we define a function on $v = (v_x, v_y)$ that is proportional to the vector's slope, with no dependency on its sign (definition A.4.13):

$$\theta(v) \stackrel{\text{def}}{=} \begin{cases} \infty & v = (0, 1)^t \\ -\infty & v = (0, -1)^t \\ \frac{v_y}{v_x} & \text{otherwise.} \end{cases} \quad (5.1.1)$$

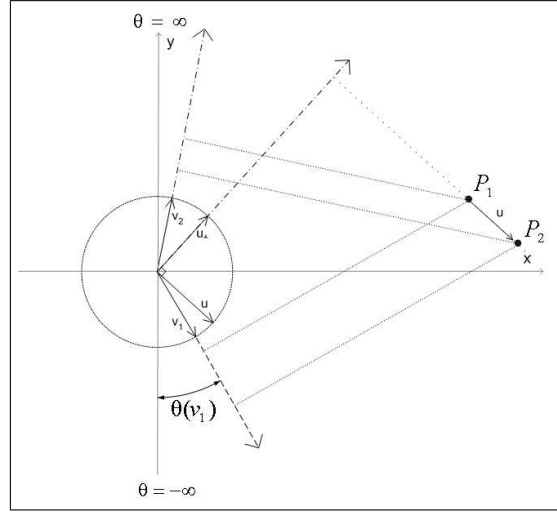


Figure 5.1: The order of two points on different lines.

In Figure 5.1 we can see, for example, that $\theta(v_1) < \theta(u_\perp) < \theta(v_2)$. Notice that the order of a line, ℓ_i in \mathcal{L} , depends only on its direction, v_i . If ℓ_i is the line that goes through the points p_1 and p_2 , v_i is the direction from one of the point to the other. Later we will reference the unit vector v_i as $\text{dir}(p_1, p_2)$ or $\text{dir}(p_2, p_1)$. These two vectors are different only by their sign, so for consistency we will take the one with the positive x-coordinate. Formally:

$$\text{dir}(p, q) \stackrel{\text{def}}{=} \begin{cases} \frac{p-q}{\|p-q\|} & p_x \geq q_x \\ \frac{q-p}{\|q-p\|} & p_x < q_x \end{cases} \quad (5.1.2)$$

The set of all vectors $v_i, 1 \leq i \leq N$ will be denoted by D :

$$D \stackrel{\text{def}}{=} \{\text{dir}(p, q) \mid p, q \in S\} \quad (5.1.3)$$

5.1.3 Time Complexity

A line $\ell_i = \text{line}(v_i, c_i)$ that goes through two input points p_1 and p_2 , can be evaluated from these points in a constant time: The direction v is the

direction from p_1 to p_2 (or vice versa), which is the vector $p_1 - p_2$ after normalization. c can be calculated easily using the projection of one of the points on the orthogonal to v_i (see Figure 2.1 and section 2.2). $\theta(v_i)$ can also be calculated in $O(1)$, given v_i .

We see that for each of the N pair of points, constant time is needed, and the total time to construct all the lines in \mathcal{L} is therefore $O(N)$. After getting ℓ_i for each $1 \leq i \leq N$, all we need to do is sort them by their θ values. This sorting takes $O(N \log N) = O(n^2 \log(n^2)) = O(n^2 \log n)$ time.

5.2 The Structure \mathcal{F}

5.2.1 Description

For any line $\ell \in \mathcal{L}$ and a point $p \in S$, let $IsLeft(p, \ell)$ be true iff the point p is left to the line ℓ . In the case where ℓ is horizontal, $IsLeft(\ell)$ will be true iff the point p is above it. This function is easy to be calculated in $O(1)$ (see theorem 6.3.1). Let $LeftSubset(\ell) = \{p \in S \mid IsLeft(p, \ell)\}$.

Let $\mathcal{F} = \{\text{vec}(LeftSubset(\ell)) \mid \ell \in \mathcal{L}\}$, where vec is the summary vector defined in A.3.1.

This data structure support two methods:

initialize(S) — constructs the set of summary vectors, \mathcal{F} , using the structure S .

Space Complexity: $O(n^2)$

Time Complexity: $O(n^2)$.

getFirstPartitionInfo(ℓ) — returns the vector $u \in \mathcal{F}$ such that $u = \text{vec}(LeftSubset(\ell))$.

Time Complexity: $O(1)$.

5.2.2 Implementation

To construct \mathcal{F} we will iterate through all the N subsets in such a way that from one iteration to the next, only a single point will be added or removed. This can be done if we will rotate a line counter clockwise, about a point $p \in S$. Notice that each time the rotating line comes across another point of the set, it is equals to one of the lines $\ell_i \in \mathcal{L}$, and the subset left to that line is therefore $LeftSubset_i$. When the rotated line meets the next point, the new subset is different by only one point from the previous one.

Notice that there are no more than n such iterations in the discrete sense. Every time a point is added or deleted in the process of rotating the line,

we calculate $LeftSubset\ell_i$ for another line, $\ell_i \in \mathcal{L}$, that is going through p .

To compute the summary vectors for every subset left to the rotating line, we don't have to know exactly what points are in the subset. Using the results of section 3, and the fact that the left subset in each iteration changes by only one point, we can update the summary vector in $O(1)$.

After completing this process we should have n summary vectors, which correspond to the n lines of \mathcal{L} that are going through p . If we repeat this process for all the points $p \in S$, we will eventually have the initial partition information for any first decision boundary ℓ . This partition

5.2.3 Time Complexity

The structure will be built incrementally with insert/delete determined by the angular order. We start with finding for each $p_k \in S$ the ordering of $S \setminus p_k$ with respect to the polar angle around p_k (from its right). It can be accomplished in $O(n^2)$ time and space; see [1].

The next process is repeated for each point $p \in S$:

For the first position of the rotating line we will create the summary vector explicitly, by checking for each point whether it is left to the line or not. This can be done in $O(n)$.

Next, we calculate the order that the rotating line will meet each point. For the points below p this order is determined by the angle of each point to the right of p . Similarly, the order that points above p will meet the rotating line is determined by their angle to the left of p . To fix the original polar order (that takes always the angle from right to p) to reflect the order that the line meets the points, we should subtract π from the angles larger than π . These angles are the second part of the polar ordered list, and after the subtraction we have two sorted parts of the list. We merged them to a one sorted list in $O(n)$ and the updated order is the order that the rotating line will meet the points around p .

After we have the initial summary vector and the order that the line meets the points, we can calculate each summary vector in this order. This can be done in $O(1)$ computed time, as described in section 3.1 for updating the 1-line-mean with a single point, or in section 3.2, if the point should be removed from the first set. It is easy to decide whether the point should be added or removed by noticing that if the new point is below p it must be added to the set as the line rotates clockwise, and it must be removed if it is located above p . We repeat updating the values of \mathcal{F} for all the points on the polar list of p recursively. On the end of this process we would have all the items of \mathcal{F} that correspond to a line in \mathcal{L} that goes through p . The total time for the process around p is $O(n)$:

Repeating the process for all the n points is therefore $O(n^2)$, which is the same as generating the original polar order for all the points, and we have total time $O(n^2)$ for the `initialize` method of this structure. Each calculated vector is attached to the corresponding line such that `getFirstPartitionInfo(ℓ)` can retrieve it in only $O(1)$ time.

5.3 The Inner Structure $\Delta\mathcal{O}$

5.3.1 Description

This structure is used by the structure \mathcal{O} to generate the list of points ordered by their projections on any line $\ell \in \mathcal{L}$. However, it will not contain a list of all the n points in order of the length of their projections on ℓ_i . Instead, for each line it will contain only the pair of points that need to be swapped relative to the order on the previous line, ℓ_{i-1} . Formally:

For any point, $p \in S$ we define $Index(p, i)$ to be the index of p in the array $\mathcal{O}(i)$. For any $1 \leq i \leq N - 1$ let $Changes(i) = \{(p, q) | Index(p, i) \geq Index(q, i) \text{ and } Index(p, i + 1) < Index(q, i + 1)\}$.

Let $\Delta\mathcal{O}$ be the set of pairs $\Delta\mathcal{O} = \{Changes(i) | 2 \leq i \leq N\}$.

The structure $\Delta\mathcal{O}$ support two methods:

`initialize(S, \mathcal{L})` — Constructs the pairs of $\Delta\mathcal{O}$. Time Complexity: $O(n^2)$.
Space Complexity: $O(n^2)$.

`getChangesForNextLine()` — On the i 'th call to this method it returns the pairs of $Changes(i)$. Amortized Time Complexity: $O(1)$.

The order of points by their projection on the line(v, c) depends on their inner product with v .

5.3.2 Implementation

Here we explain how to construct $Changes(i)$ for every $1 \leq i \leq n$ on the call to the `initialize` method. After the initialization the function `getChangesForNextLine` will return $Changes(i)$ on the i 'th call.

Notice that the order of the projections on $\ell = \text{line}(v, c)$ depends only on the unit vector v , so we can limit the discussion to projection order on different unit vectors. The key observation is that because all the lines in \mathcal{L} are sorted by their slopes, the projection order on each vector v_i is similar to the order on the previous vector, v_{i-1} . As will be shown, the order of the lines in \mathcal{L} promise us that every pair of points will exchange places only once during the process and we get a total of N changes ($|\Delta\mathcal{O}| = N$).

In Figure 5.1 we can see two points, p_1 and p_2 , and their projections on two vectors: v_1 and v_2 . We will assume that v_1 and v_2 are two directions of

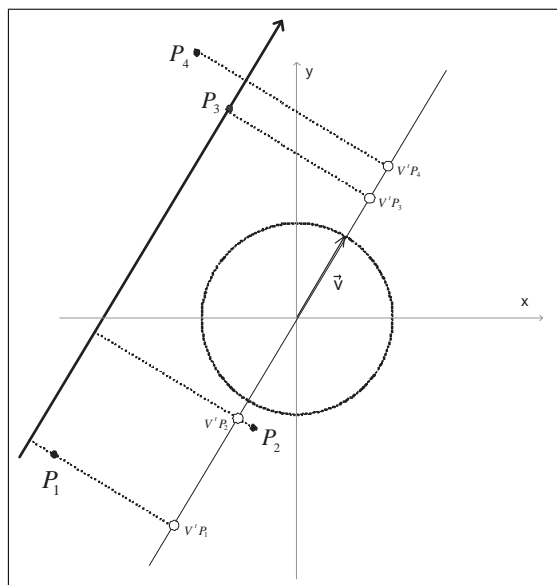


Figure 5.2:

lines in \mathcal{L} , which means $v_1, v_2 \in D$ (see definition A.4.12 and section 5.1). We also add another unit vector, u , which is the direction from p_1 to p_2 : $u = \text{dir}(p_1, p_2)$, as shown on the figure. We can see that the two points have the same projection on u_\perp , the orthogonal vector to u , and therefore have the same order on a line with a slope as u_\perp . The conclusion is that the projection order of the points was swapped when we moved from v_1 to v_2 , and their projection was equal on the vector u_\perp , which is between v_1 and v_2 .

The proof for the next theorem shows that it is not a coincidence. The orthogonal vector to $\text{dir}(p_1, p_2)$ is located between two unit vectors, v_1 and v_2 , iff the projected order of p_1 and p_2 is swap when changing the projected direction from v_1 to v_2 . To understand the role of u , notice that if u_\perp and v_1 were the same vector, it would mean that the direction between the two points, u , is exactly orthogonal to v_1 . In that case p_1 and p_2 would have the same projection on v_1 . A small movement of v_1 to each direction will change the order of their projection on it. The new order will depend on whether v_1 was rotated clockwise or not.

This observation is proven formally in appendix A, using the definition of θ (see section 5.1 and definition A.4.13):

Theorem 5.3.1. *Let $u, w \in D$ be two unit vectors such that $\theta(u) < \theta(w)$,*

and let $p, q \in S$ be two points in S such that $u^t p < u^t q$. Then:

$$w^t p \geq w^t q \Leftrightarrow \theta(u) < \theta(v_\perp) \leq \theta(w)$$

where $v = \frac{p-q}{\|p-q\|}$.

The conclusion from this theorem is that if we sort all the vectors in D together with their orthogonals, the points that should be swapped from one vector to the next, are the pairs of points that belong to the orthogonals between them. For example, if the order begins with:

$$\theta(v_1) < \theta(v_{8\perp}) < \theta(v_{9\perp}) < \theta(v_2) < \theta(v_3) \dots$$

where $v_8 = \text{dir}(p_1, p_2)$ and $v_9 = \text{dir}(p_3, p_4)$, then the projection order on v_2 is equal to that on v_1 except the points p_1 and p_2 , which will be switching places, and the same for p_3 and p_4 . The projection order of the points on v_2 and v_3 will be exactly the same.

Notice that each vector, v_i , corresponds to only one vector on the list, namely $v_{i\perp}$, and each such vector represents a swapping of exactly one pair of points. We can conclude that every pair is switched exactly one time, and we have a total of N switches during the iteration over the sorted list.

To construct $\Delta\mathcal{O}$ we sort all the vectors in D with their orthogonals as described above, and iterate through the ordered list. For every vector u_\perp between v_1 and v_2 we add a pair of points, (p_i, p_j) , to the list $\text{Changes}(1)$, where $u = \text{dir}(p_i, p_j)$. After the iteration gets to the vector v_2 , we begin to construct the list $\text{Changes}(2)$ that describes the changes between v_2 and v_3 . In the end of this process we get to v_n and finish constructing the list \mathcal{W}_N .

5.3.3 Time Complexity

The construction of $\Delta\mathcal{O}$ takes $O(n^2)$ time. For generating the sorted list of the vectors in D with their orthogonals we need to calculate the orthogonal for each vector in D . Using definition A.2.2 we can do this in $O(1)$ time.

Calculating the θ value for each vector is also $O(1)$ (using definition A.4.13), and the total time to construct the merged list before sorting is therefore $O(N)$. The list contains $2N$ values and we can sort them by their θ values in $O(2N \log(2N)) = O(n^2 \log n)$ time.

Actually we can get the list sorted in $O(N) = O(n^2)$ time, by merging the list \mathcal{L} , which is already sorted, with a sorted list of the orthogonal vectors. The order of the orthogonal vectors is very similar to the order of the original vectors: all we need to do is find the first vector, v_k , in \mathcal{L} such that

$$\theta(v_k) \geq 0.$$

As can be seen in Figure 5.2, the orthogonal for this vector on the positive x-axis is the closest to $\theta(-\infty)$ and must be the first on the sorted orthogonals list. For the same reasons, the rest of the orthogonal vectors to v_{k+1}, \dots, v_N , will follow in the same order. The rest of the orthogonals, $v_{1\perp} \dots v_{k-1\perp}$, are sorted and have positive θ values. They should be added after the orthogonal to v_N , which has the smallest negative θ value. It is easy to show that this list of orthogonals is sorted by their θ values. Merging these two sorted lists of N items to one sorted list takes $O(N) = O(n^2)$.

5.4 The Structure \mathcal{O}

5.4.1 Description

For any line $\ell = \text{line}(v, c)$, let $\text{ProjectionOrder}(\ell) = (p_1, p_2, \dots, p_n)$ be a permutation of the points in S as the order of projections among ℓ ($v^t p_1 \leq v^t p_2 \leq \dots \leq v^t p_n$) as the order of the points in Figure 5.3.

For any $1 \leq i \leq N$, let $\mathcal{O}(i) = \text{ProjectionOrder}(\mathcal{L}.\text{getLine}(i))$.

This data structure support three methods:

initialize(S, \mathcal{L}) — constructs $\mathcal{O}(1)$ and initialize the inner structure $\Delta\mathcal{O}$.

Space Complexity: $O(n^2)$. Time Complexity: $O(n^2 \log n)$.

prepareNextOrder— on the i 'th call to this method it constructs the projection order $\mathcal{O}(i)$.

Amortized Time Complexity: $O(1)$.

getNextPoint(ℓ) — returns the j 'th point on $\mathcal{O}(i)$, where $\mathcal{O}(i)$ is the permutation of points that were created on the last call to **prepareNextOrder**.

Time Complexity: $O(1)$.

The n possible partitions, where the first bounder is the line ℓ .

5.4.2 Implementation

Here we assume that the structure \mathcal{L} was already initialized.

initialize(S, \mathcal{L}) — On the call to this method the projection order among the first line of \mathcal{L} , which is $\mathcal{O}(1)$, will be created together with a call to $\Delta\mathcal{O}.\text{initialize}(\mathcal{L})$.

The order on the first line, $\mathcal{O}(1)$, can be retrieved by first calling $\mathcal{L}.\text{getLine}(1)$ which returns the direction of the first line, v , in $O(1)$ time. We get the projection order on this line by sorting the points of S such that: $v^t p_1 \leq v^t p_2 \leq \dots \leq v^t p_n$ (see Figure 5.2).

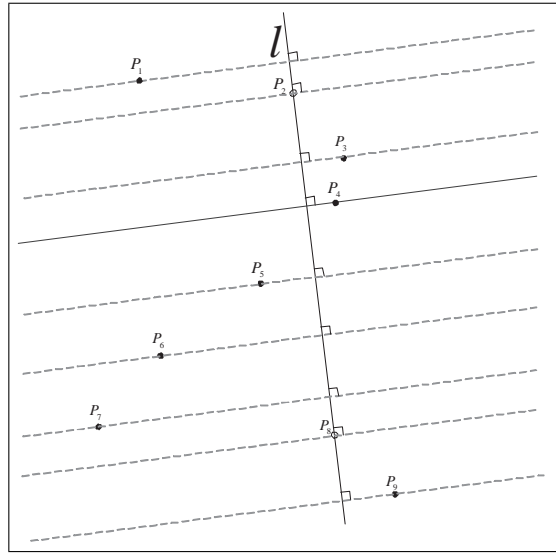


Figure 5.3:

prepareNextOrder— The first call to this function will return the projection order that was created on the call to **initialize**. For the i 'th call ($i > 2$) this function should return the projection order for the i 'th line in \mathcal{L} , $\mathcal{O}(i)$. Of course we can resort the points as described in **initialize** on each call, but this will cost us $O(Nn \log n) = O(n^3 \log n)$ for all the N calls to this method. Instead, we will use the structure $\Delta\mathcal{O}$ and update the order of the points for the next line incrementally.

On each call to this method, the returned order will be saved instead of the previous one. On the i 'th call, the new order will be retrieved by calling **getChangesForNextLine()** and swapping each returned pair in the current order.

5.4.3 Time Complexity

initialize(S, \mathcal{L}) — The sorting on the call to the **initialize** function takes $O(n \log n)$ time. In addition, this method call $\Delta\mathcal{O}.\text{initialize}(\mathcal{L})$ which takes $O(n^2)$ time and $O(n^2 \log n)$ space, which is the total time and space complexity for this method.

prepareNextOrder— For each line, $\ell_i \in \mathcal{L}$, we need to update the list of points ordered by their projections on the previous line using the result of $\Delta\mathcal{O}.\text{getChangesForNextLine}()$. Although the number of pairs of points that should be swap might be $O(n)$, the sum of swaps for all the lines in \mathcal{L} is the number of pairs in $\Delta\mathcal{O}$ which is $O(N)$ (see section 5.3).

Chapter 6

The Algorithm

Section 6.1 presents the pseudo-code for the 2-line-means algorithm, using the data structures of previous section. The next section explains each line of the algorithm.

The algorithm traverses all $\binom{n}{2}$ lines determined by 2 points in S , in an order that will be defined by the structure \mathcal{L} . In each such line it traverses all possible orthogonal lines that determine all arrangement of the rest of the points, in S , as shown in Figure 5.3.

As will be shown later and in theorem 6.3.1, from each one of the possible $O(n^3)$ partitions to the next only one point will be added or removed, and the grade for the new partition can be updated in $O(1)$ time resulted in an $O(n^3)$ time algorithm.

6.1 Pseudo-Code

The following algorithm computes the 2-line-means in $O(n^3)$ time, and is based on the observations of last sections. The input is a set of points as a columns matrix, $S = [p_1, p_2, \dots, p_n]$, and the output is the 2-line-means. The lines mentioned above and over the whole algorithm are assumed to be stored as a a pair of numbers, v , and a real, c , in the form $\text{line}(v, c)$ (see definition (A.2.3) in section 2.2). The algorithm also uses the function *IsLeft* which was defined in 5.2.

The algorithm is as follows:

```

 $(\ell_1^*, \ell_2^*) = \text{FIND-2-LINE-MEANS}(S)$ 
1.  $n := |S|$ 
2. initialize  $\mathcal{L}$ ,  $\mathcal{F}$ , and  $\mathcal{O}$  with the points of  $S$ .
3. for  $i:=1$  to  $\binom{n}{2}$ 
4.    $\ell := \mathcal{L}.\text{getLine}(i)$ 
5.    $\text{summaryVector} := \mathcal{F}.\text{getFirstPartitionInfo}(\ell)$ 
6.    $\mathcal{O}.\text{prepareNextOrder}()$ 
7.   for  $j:=1$  to  $n$ 
8.      $p := \mathcal{O}.\text{getNextPoint}()$ 
9.     if  $\text{IsLeft}(p, \ell)$  then
10.      remove the point  $p$  from  $\text{summaryVec}$ .
11.     else
12.      add  $p$  to  $\text{summaryVec}$ .
13.      $\text{currentGrade} :=$  the grade of  $\text{summaryVec}$ .
14.     if ( $\text{currentGrade}$  is better then the best grade) then
15.      update the best grade.
16.       $\ell^* := \ell$ 
17.       $p^* := p$ 
18.  $s_1^* :=$  find the partition of  $S$  that corresponds to the bounder  $\ell^*$ ,
      and its orthogonal that goes through  $p^*$ .
19.  $s_2^* := S \setminus s_1^*$ 
20.  $\ell_1^* :=$  the 1-line-mean of  $s_1^*$ .
21.  $\ell_2^* :=$  the 1-line-mean of  $s_2^*$ .

```

6.2 Description

- In line 2 the data structures described in section 5 are constructed for the set S . This is done by activating the initialization method of each one of them, with S as an argument.
- In lines 3-17 we iterate through all possible positions for the first decision bounder. These are actually the lines in \mathcal{L} .
- In line 4, ℓ represents the decision bounder (a line from \mathcal{L}) that is being checked for current iteration.
- In line 5 the summary vector for the first partition is retrieved from the data structure \mathcal{F} . This is the partition created by the bounder ℓ and the bounder that goes through p_1 , as shown in Figure 5.3. In this partition the summary vector has the information on the points left to ℓ .

- In line 6 the projection order of the points on the first decision boundary, ℓ , is created. This order will be used to iterate through the possible locations of the second decision boundary, as shown in Figure 5.3.
- In lines 7-17 all the possible partitions are being checked, where ℓ is assumed to be the first decision boundary. On each iteration the current partition is being evaluated and the second decision boundary move to the next point, until all the n points have been used, as shown in Figure 5.3.
- In line 8, p is the next point that the second boundary is assumed to go through. The order of the points is determined by the projection order on the first boundary, which was calculated on line 6.
- In lines 9-12 the summary vector for the new partition is updated, using the fact that only one point was added or removed in relation to the last partition (see theorem ??). The question whether the point should be added or removed from the summary vector is depend whether it was included in the first partition (line 5), *i.e.* whether it is left to ℓ or not. The *Left* function was defined on 5.2.
- In line 13 the sum of squared distances for current partition is calculated. This is done using the equation of theorem 4.6.1:

$$G(A, B) = \vec{G}(\text{vec}(A), \text{vec}(B)) + \frac{1}{2} \sum_{i=1}^n x_i^2 + y_i^2.$$

where A and B represents the whole set S and the subset for current partition, respectively. Notice that $\text{vec}(B) = \text{summaryvector}$, and $\text{vec}(A)$ together with $\sum_{i=1}^n x_i^2 + y_i^2$ are constant during the whole process and need to be calculated only once.

- In lines 14-17 the sum of squared distances for the current partition is compared to the minimum sum of the previous partitions. If the current partition is better – the best sum is updated, together with the partition boundaries. In the end of the process this partition will be the partition of the 2-line-means.
- Lines 18-19 are called once after all the candidate partitions have been checked. The two subsets of points for the best partition are being retrieved using its two boundaries: ℓ^* and its orthogonal through p^* . These are the subsets S_1^* and S_2^* that were defined in section 4.1 and Figure 4.1.
- In lines 20-21 the 2-line-means are being calculated as the two 1-line-mean of each subset, s_1^* and s_2^* . For explanation see section 4.2 and theorem 4.2.1.

6.3 Time Complexity

The fact that the algorithm time complexity is the same as the number of partitions (the two for loops) is due to the following theorem. This theorem shows that each new searched partition can be updated from the previous one, by $O(1)$ time.

Theorem 6.3.1 (). *Let $\ell = \text{line}(v, c)$. If $v^t p_1 < v^t p_2 < \dots < v^t p_n$ then:*

$$p_{i+1} \in S_1(\ell, p_i) \Leftrightarrow v_{\perp}^t p_{i+1} - c \leq 0$$

and:

$$S_1(\ell, p_{i+1}) = \begin{cases} S_1(\ell, p_i) \setminus p_{i+1} & p_{i+1} \in S_1(\ell, p_i) \\ S_1(\ell, p_i) \cup p_{i+1} & p_{i+1} \notin S_1(\ell, p_i) \end{cases}$$

The first part of the theorem is actually determines whether the new point is left or right to the line ℓ , and we can define:

$$IsLeft(\ell, p) \Leftrightarrow v_{\perp}^t p_{i+1} - c \leq 0$$

where $\ell = \text{line}(v, c)$.

Because every command in the inner loop takes $O(1)$, and the number of iterations is $Nn = O(n^3)$, this is the total time complexity for these lines. Lines 4-6 takes $O(1)$ and computed N times, and the rest of the lines are computed once and takes less than $O(n^3)$, so the total time complexity for the algorithm is $O(n^3)$.

Chapter 7

Generalizations

In this section we show some simple variations of the 2-line-means as described in this work. All the variations can be combined and does not require additional time complexity, except the generalization for k -lines and d -dimensions.

7.1 k -Line-Means

We can easily generalize the solution to k -line-means, using the observation that every two lines partition the data with two decision boundaries as described in this work. Every two lines that meet create such a partition and the combination of the whole $\binom{k}{2}$ pairs of decision boundaries define the partition of the points to the k groups (See Figure 7.1). As was described in previous chapters, every such pair defines $O(n^3)$ possible partitions, and the total possible partitions for the $\binom{k}{2}$ intersections is $O(n^3)^{\binom{k}{2}} = O(n^{k^2})$. Using the same data structures of section 5 we can implement an algorithm that calculate the k -line-means in this time.

4-line-means and their $\binom{4}{2}$ pairs of decision boundaries.

7.2 Weighted Axis

In the previous sections we defined the distance from a point to a line as the distance from this point to the closest point on the line:

$$\text{dist}(\ell, p) \stackrel{\text{def}}{=} \min_{p_L \in \ell} d(p - p_L) \quad (7.2.1)$$

where $d(v) = \sqrt{v_x^2 + v_y^2} = \|v\|$. In some cases, however, the error (distance) on the y -coordinate is more critical than the x -coordinate. For example, if the x -axis represents the time, and the y -coordinate represents the result of an experiment, we might assume that the measured time of the

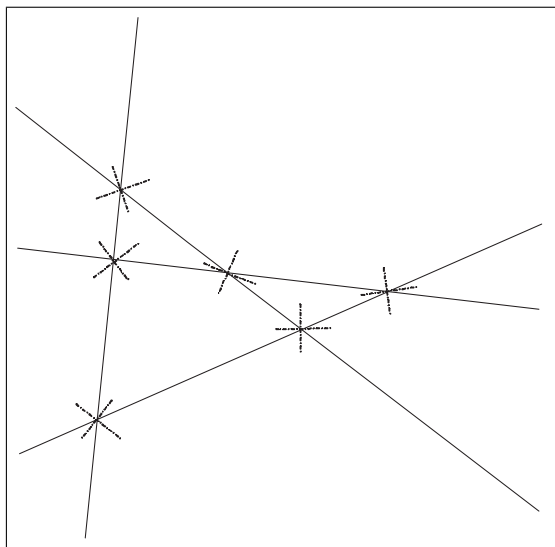


Figure 7.1:

experiment is much accurate than its result. Other case might be that both of the axis measure time, but are not using the same time unit.

In these cases we can give positive weights to the two dimensions, $w_x, w_y \in \mathbb{R}$ that represent their relative error and define the weighted distance as:

$$d_w(v) = \sqrt{(w_x v_x)^2 + (w_y v_y)^2} = \|Wv\| \quad (7.2.2)$$

where $W = \begin{pmatrix} w_x & 0 \\ 0 & w_y \end{pmatrix}$.

Notice that $w_x = w_y = 1$ is the original problem. In some cases, for example where the time measurement is accurate, we want the error to be assigned to only one of the axes, which means a vertical (or horizontal) distance from a point to the line (see Figure 7.2). In this case we choose w_x (or w_y) on definition (7.2.2) to be a very large number ($w_x \rightarrow \infty$), such that the point on a line, p_l , that is closest to the input point p will have to be with the same x -coordinate.

We can show a simple reduction from the k -line-means problem as described in this work. This reduction is actually true for any non-singular matrix W and not only a diagonal one. For every point, p , we define $p' = Wp$. Notice that because W is not singular, for any line ℓ we can find a line ℓ' such that $p \in \ell \iff p' \in \ell'$. The weighted distance from a point to a line is then:

$$\begin{aligned}
\text{dist}_w(\ell, p) &= \min_{p_L \in \ell} d_w(p - p_L) \\
&= \min_{p_L \in \ell} \|W(p - p_L)\| \\
&= \\
\min_{p_L \in \ell} \|Wp - Wp_L\| & \\
&= \min_{p_L \in \ell} \|p' - p'_L\| \\
&= \min_{p'_L \in \ell'} \|p' - p'_L\| \\
&= \text{dist}(\ell', p')
\end{aligned}$$

From this calculation we can conclude that for solving the k -line-means with a weighted distance, all we have to do is multiply each input point by W , and find the regular k -line-means. For each line of the result, ℓ' we should retrieve the original line, ℓ , using W . The conversion of the points and the lines takes no more than $O(n)$ time and done only once, hence the total space and time complexity are the same as the original problem.

In case where the error is only on one of the axes as described above ($w_x \rightarrow \infty$), there is a better solution, which gives an interesting reduction from the k -means problem for points (see 1.4).

It is easy to see that where the error is, for example, only on the y -axis as in Figure 7.2, the optimal k -line-means are horizontal lines $y_i = c_i$, where c_i is the average y -coordinate of the input points for the i 'th line mean.

The x -coordinate of the points has no effect on the result, and we can assume all the points are located on one vertical line which contains their projections on the y -axis. If we mark white points with height c_i on that line, as shown on the figure, the distances from the projected input points to that points, is the same as the distance from the original input points to their line-mean. When we look for the k -line-means with this distance function, the result should be k horizontal lines, and after projecting the points on the y -axis, we get the k -means problem on 1-dimension, which has many solutions and approximations as explained in section 1.4.

The 3-line-means where $w_x \rightarrow \infty$, and its 3-means representation.

7.3 Different Norms

The geometrical observations of section 4 can be separated from the algebra of the previous chapters. If we choose, for example, the L_1 norm instead of the Euclidean one, the new decision boundary of the solution will still be orthogonal and we can use the same data structure and the same algorithm as for the original problem. The only difference is that the function that finds the 1-line-mean of a set of point should be changed.

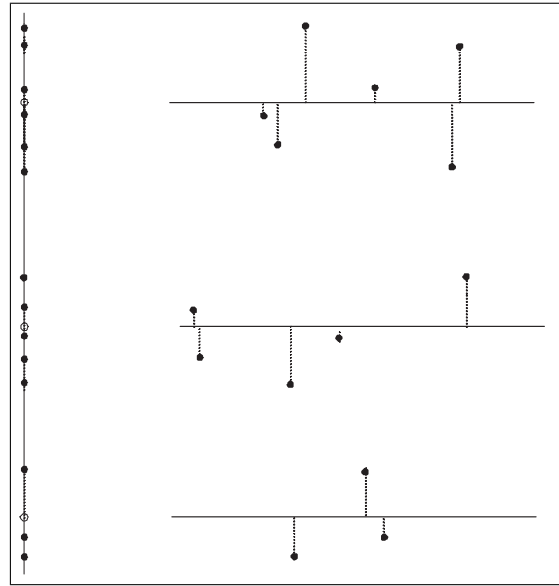


Figure 7.2:

In fact, it is easy to see that there are $O(n^3)$ possible partitions for every distance function that separates the points into two groups using two decision boundaries with a constant angle between them, as shown in section 4. If we know how to calculate the 1-line-mean for such a problem in $O(f)$ time, and update it for a change of a single point in $O(g)$ time, we can generalize the algorithm easily to an algorithm with time complexity of $O(n^2f + n^3g)$. This is because the total time for the construction of the data structures will remain the same except the structure \mathcal{F} which will take $O(n^2f)$ time to construct.

The algorithm for updating the partition will be called $O(n^3)$ times for each possible solution and this is how we got a total time of $O(n^2f + n^3g)$.

7.4 Weighted Points

In case that not all the points have the same importance, or we believe some of them has less error than the others, we might assign a weight w_i for each input point p_i , and minimize the sum of weighted squared distances from each point to its closer line. We show here, in a schematic form, how the algorithm for the 1-line-mean can be slightly changed to solve this problem. Using these changes, the algorithm for the 2-line-means can be achieved easily.

On the original problem, the value c of $\text{line}(v, c)$. The first change is of theorem 2.4.1. For any fixed unit vector v :

$$\arg \min_c \sum_{i=1}^n \text{dist}^2(\text{line}(v, c), p_i) = v_{\perp}^t \bar{p}.$$

In the weighted version, the theorem is the same except that \bar{p} is now the weighted average of the points:

$$\bar{p} \stackrel{\text{def}}{=} \frac{\sum_{i=1}^n w_i p_i}{\sum_{i=1}^n w_i}.$$

This result can be obtain using a similar proof for the original theorem, but using

$$\arg \min_c \sum_{i=1}^n w_i \text{dist}^2(\text{line}(v, c), p_i)$$

as the target c value.

After calculating the weighted average \bar{p} which the weighted 1-line-mean must go through, we can subtract it from each input point as done in the original case, and search for a line that is going through the origin $((0, 0))$. As in theorem 2.5.3 we will assume that the direction of the line is orthogonal to the unit vector v , that is, the weighted distance from a point to a line is $w_i(p_i - \bar{p})^t v$.

If we define W to be a diagonal matrix with w_i on the i 'th line, then the vector v that minimizes this expression is the vector that minimizes $\|W\tilde{B}v\|$, where \tilde{B} is the column matrix that represents the points after subtracting their average.

If we define \tilde{A} to be $W\tilde{B}$ then the requested vector is the vector that minimizes $\|\tilde{A}v\|$, and this vector will be the direction of the line. The vector v can be found using the *SVD* decomposition, as described in 2.5.3.

We summarize the result as theorem 2.5.3 with the changes in the definitions:

Theorem 2.5.3.

$$\ell^* = \arg \min_{\ell} \sum_{i=1}^n w_i \text{dist}^2(\ell, p_i)$$

where $\ell^* = \text{line}(\vec{\sigma}_{max}, \vec{\sigma}_{min}^t \bar{p})$, and this sum of squared distances equals to:

$$\sigma_{min}^2 = \sum_{i=1}^n w_i \text{dist}^2(\ell^*, p_i)$$

where \bar{p} is the weighted average as defined in this section, $\vec{\sigma}_{max}, \vec{\sigma}_{min}$ are the larger and smaller singular vectors of the matrix $W\tilde{B}$ as defined earlier, and σ_{min} is the smallest singular value of this matrix.

The on-line updating of the 1-line-means is very similar to the original one with changes that are similar to the ones that were shown here, and therefore we decided not to rewrite them again.

Chapter 8

Further Work and Open Problems

We can generalize the 2-line-means problem, as defined on this work, in different ways, or a combination of them.

- Assume that the given input points contain outliers. That is, the sum of errors should be calculated on only some of the points. For example, for the given set of n points and a number $k \leq n$, find a subset of $n - k$ points, such that its 2-line-means has the minimum sum of squared distances. This can be useful if we assume that some of the samples might be pure noise. It also might prevent the case when one or two extreme points can change significantly the mean lines positions. Some algorithms for one estimator can be found in [2]
- The input points can be taken from a dimension d , larger than two, *i.e.* the points will no longer be on the plane. In this case we can choose the means from several dimensions between 0 to $d - 1$. For example, 0-dimension will be mean points, which is the known k -means problem (see [28, 27]), and 1-dimension means will be lines in the space. The case of $d - 1$ dimensions for the means is the case of fitting hyperplanes to the data, and seems the most easy to generalize using this work. This is because in this case the geometrical observations will stay essentially the same, and the generalization of finding the 1-line-mean is straightforward.
- We believe that the results for the k -line-means can be much better, and it is an open problem what is the lower bound for the time complexity.
- In case where each sample (point) is labelled with category/class num-

ber, we might look at the means as classifiers, where each mean represents another class. Giving new sample, its class will be determined according to its closer mean. This kind of problems might suggest an alternative rigorous definitions to problems in the area of ensembles or multiple classifiers which usually uses heuristics and based on experiments that are often contradict others.

- It is not hard to show that many of the above generalizations are NP-Complete (see [12]), and the requested solution should only give an approximation to the optimal solution, or a randomized algorithm. Such approximations might be also used for solving the problem of this work with a better time complexity. This can be specially useful in areas such as data/text mining, or image processing when the number of samples are huge, but there expected to be many similar samples, and even a rough approximation might do fine.

As far as we know, most of the solutions to the above problems are heuristics (see section 1.4) and can be considered as open problems. Although some of these heuristics are very popular, a little work has been done to analyze them. An open problem is whether we can convert these heuristics or some of them into algorithms? For example, can we show on which subset they will do well, or show they are random algorithm that solve the problem with high probability or on special conditions? The open problem that is most related to this work is whether there is an optimal solution to the 2-line-means problem described here in less than cubic time complexity. We believe there is, and hope to show in the near future that the number of possible partitions might be smaller than $\Theta(n^3)$.

Appendix A

Formal Proofs

A.1 General Algebraic Review

Theorem A.1.1. For every matrix $M_{m \times n}$, with c_1, c_2, \dots, c_n as its columns:

$$MM^t = \sum_{k=1}^n c_k c_k^t$$

Writing a matrix in this form called its scatter form.

Proof. Lets mark the columns and rows of M as column vectors:

$$M = \left(\begin{array}{c|c|c|c} | & | & | & | \\ c_1 & c_2 & \cdots & c_n \\ | & | & | & | \end{array} \right) = \left(\begin{array}{c} - r_1^t - \\ - r_2^t - \\ \vdots \\ - r_m^t - \end{array} \right)$$

The scalars of each vector will be denoted by a second index, for example: $r_i = (r_{i1}, r_{i2}, \dots, r_{in})$. Notice that $r_{ij} = c_{ji}$.

We will also denote the i 'th row of the matrix MM^t as t_i and the i 'th row of the scatter matrix $\sum_{k=1}^n c_k c_k^t$ as a column vector, s_i .

Observe that for any k , the entry of the matrix $c_k c_k^t$ in row i , column j is $c_{ki} c_{kj}$. So for any $1 \leq i \leq n$:

$$s_{ij} = \sum_{k=1}^n c_{ki} c_{kj} = \sum_{k=1}^n r_{ik} r_{jk} = r_i^t r_j = t_{ij}$$

which means that all the entries of the corresponding matrices, MM^t and its scatter form, are the same. \square

Theorem A.1.2. if $Q_{m \times n}$ is an orthogonal matrix, and $v \in \mathbb{R}^n$ then:

$$\|Qv\| = \|v\|$$

Proof. We denote the rows of Q as columns vectors:

$$Q = \begin{pmatrix} -q_1^t - \\ -q_2^t - \\ \vdots \\ -q_m^t - \end{pmatrix}$$

and because $q_i^t v$ is a scalar, and $q_i^t v = v^t q_i$:

$$\begin{aligned} \|Qv\|^2 &= (q_1^t v)^2 + (q_2^t v)^2 + \dots + (q_m^t v)^2 \\ &= (v^t q_1)(q_1^t v) + (v^t q_2)(q_2^t v) + \dots + (v^t q_m)(q_m^t v) \\ &= v^t (q_1 q_1^t + q_2 q_2^t + \dots + q_m q_m^t) v \end{aligned}$$

using the scatter form (theorem A.1.1) with $A = Q^t$:

$$\|Qv\|^2 = v^t \left(\sum_{k=1}^m q_k q_k^t \right) v = v^t Q^t Q v = v^t I v = v^t v = \|v\|^2$$

and because $\|v\|$ is non-negative, taking the square root of each side gives:

$$\|Qv\| = \|v\|$$

□

A.2 1-Line-Mean

A.2.1 Definitions

p_i — a column vector that represents a point on the plane:

$$p_i \stackrel{\text{def}}{=} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad \text{where } (x_i, y_i) \in \mathbb{R}^2. \quad (\text{A.2.1})$$

A — a matrix of size $2 \times n$ with p_i as its i 'th column vector:

$$A = \{p_i\}$$

\bar{p} — the average column vector (centroid) of A :

$$\bar{p} \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n p_i$$

\tilde{A} — will be the matrix A after subtracting its centroid from each column:

$$\tilde{A} = \begin{pmatrix} | & | & & | \\ p_1 - \bar{p} & p_2 - \bar{p} & \cdots & p_n - \bar{p} \\ | & | & & | \end{pmatrix}$$

$\sigma_{min}, \sigma_{max}$ — the smallest and largest singular values of \tilde{A} , respectively. See theorem 2.5.1 for details.

$\vec{\sigma}_{min}, \vec{\sigma}_{max}$ — the left singular vectors corresponding to $\sigma_{min}, \sigma_{max}$ respectively.

v_{\perp} — where $v \in \mathbb{R}^2$ and $\|v\| = 1$, represents the unit vector that is orthogonal to v :

$$v_{\perp} \stackrel{\text{def}}{=} (v_y, -v_x) \quad \text{where} \quad v = (v_x, v_y). \quad (\text{A.2.2})$$

$\text{line}(v, c)$ — where $v \in \mathbb{R}^2$, $c \in \mathbb{R}$ and $\|v\| = 1$ represents a line in the plane with the unit vector v as its direction and c as its shortest distance from zero. More formally, this is the set of points:

$$\text{line}(v, c) \stackrel{\text{def}}{=} \{cv_{\perp} + kv | k \in \mathbb{R}\} \quad (\text{A.2.3})$$

$\text{dist}(\ell, p)$ — the shortest Euclidean distance between the point p and any point on the line ℓ . More formally

$$\text{dist}(\ell, p) \stackrel{\text{def}}{=} \min_{p_L \in \ell} \|p - p_L\| \quad (\text{A.2.4})$$

$$\text{dist}^2(\ell, p) - \{\text{dist}(\ell, p)\}^2$$

A.2.2 Theorems

Theorem 2.2.1. *The projection's length, $k \in \mathbb{R}$, of any vector, $p \in \mathbb{R}^2$, on a unit vector $v \in \mathbb{R}^2$, is $\langle v, p \rangle = v^t p$.*

Notice that if the angle between v and p is greater than 90° , we define the projection's length to be the projection's length on $-v$, with a minus sign added.

Proof. We define p' to be the projection of p on v , and k as the length of p' (see Figure 2.1). By definition, v and the projection on it, p' , has the same direction. Because v is a unit vector:

$$p' = \|p\|v = kv \quad (\text{A.2.5})$$

Because the line between p' and p is orthogonal to the vector v (again, by definition of projection), and the direction of this line is the vector $p' - p$, their inner product is zero:

$$\langle v, (p' - p) \rangle = v^t(p' - p) = v^t p' - v^t p = 0$$

which gives:

$$v^t p' = v^t p \quad (\text{A.2.6})$$

and after substituting p' (A.2.5):

$$v^t k v = v^t p.$$

Because k is scalar, we can separate it:

$$k = \frac{v^t p}{v^t v}$$

and because v is a unit vector, $v^t v = 1$:

$$k = v^t p.$$

□

Theorem 2.3.1. *The shortest Euclidean distance between the line l and the point p is:*

$$\text{dist}(\ell, p) = |v_{\perp}^t p - c|$$

where $\ell = \text{line}(v, c)$.

Proof. By definition:

$$\text{dist}(\ell, p) = \min_{p_L \in \ell} \|p - p_L\|$$

By the line definition (A.2.3), $p_L = cv_{\perp} + kv$ for some $k \in \mathbb{R}$:

$$\begin{aligned} \|p - p_L\|^2 &= (p - p_L)^t (p - p_L) = p^t p - 2p^t p_L + p_L^t p_L \\ &= p^t p - 2cv_{\perp}^t p + c^2 + k^2 - 2kv^t p \\ &= p^t p - 2cv_{\perp}^t p + c^2 + f(k) \end{aligned} \tag{A.2.7}$$

We'll find the k that minimize that expression:

$$f'(k) = 2(k - v^t p) \Rightarrow f''(k) = 2 > 0$$

$$f'(k) = 0 \Rightarrow k = v^t p$$

comparing with (A.2.7) gives:

$$\begin{aligned} \|p - p_L\|^2 &\geq p^t p - 2cv_{\perp}^t p + c^2 + (v^t p)^2 - 2(v^t p)^2 \\ &= p^t p - 2cv_{\perp}^t p + c^2 - (v^t p)^2 \end{aligned} \tag{A.2.8}$$

Notice that because v and v_{\perp} are orthogonal base for \mathbb{R}^2 :

$$p = v \langle v, p \rangle + v_{\perp} \langle v_{\perp}, p \rangle$$

and left-multiplying by p^t gives:

$$p^t p = (v^t p)^2 + (v_{\perp}^t p)^2$$

putting this in (A.2.8) gives:

$$\|p - p_L\|^2 \geq (v_{\perp}^t p)^2 - 2cv_{\perp}^t p + c^2 = (v_{\perp}^t p - c)^2$$

and after taking the square root of each side:

$$\text{dist}(\ell, p) = \min_{p_L \in \ell} \|p - p_L\| = |v_{\perp}^t p - c|$$

□

Theorem 2.4.1. *for any fixed unit vector v :*

$$\arg \min_c \sum_{i=1}^n \text{dist}^2(\text{line}(v, c), p_i) = v_{\perp}^t \bar{p}.$$

Proof. By theorem 2.3.1, for any $c \in \mathbb{R}$ and a unit vector v :

$$\begin{aligned} \sum_{i=1}^n \text{dist}^2(\text{line}(v, c), p_i) &= \sum_{i=1}^n (v_{\perp}^t p_i - c)^2 \\ &= \sum_{i=1}^n (v_{\perp}^t p_i)^2 + \sum_{i=1}^n c^2 - \sum_{i=1}^n 2cv_{\perp}^t p_i \\ &= \sum_{i=1}^n (v_{\perp}^t p_i)^2 + nc^2 - 2c \sum_{i=1}^n v_{\perp}^t p_i \\ &= \sum_{i=1}^n (v_{\perp}^t p_i)^2 + f(c) \end{aligned} \tag{A.2.9}$$

The value of c that minimize this expression, c^* , is:

$$f'(c) = 2(nc - \sum_{i=1}^n v_{\perp}^t p_i) \Rightarrow f''(c) = 2n > 0$$

$$f'(c^*) = 0 \Rightarrow c^* = \frac{1}{n} v_{\perp}^t \left(\sum_{i=1}^n p_i \right) = v_{\perp}^t \bar{p}$$

putting this in (A.2.9) gives:

$$\begin{aligned}
\sum_{i=1}^n \text{dist}^2(\text{line}(v, c), p_i) &\geq \sum_{i=1}^n (v_{\perp}^t p_i)^2 + f(c^*) \\
&= \sum_{i=1}^n (v_{\perp}^t p_i)^2 + nc^{*2} - 2c^* \sum_{i=1}^n v_{\perp}^t p_i \\
&= \sum_{i=1}^n (v_{\perp}^t p_i - c^*)^2 \\
&= \sum_{i=1}^n \text{dist}^2(\text{line}(v, c^*), p_i)
\end{aligned}$$

□

The following theorem is proven in [38] and is called The Singular Value Decomposition

Theorem 2.5.1. *Every real matrix $M_{m \times n}$ can be written in its unique SVD form which is:*

$$M = U\Sigma V$$

with $U_{m \times m}$ and $V_{n \times n}$ as orthogonal matrices, and $\Sigma_{p \times p}$ as a diagonal matrix such that $p = \min\{m, n\}$ and the values along the diagonal of Σ are in non-decreasing order.

The columns of matrix U are called the left singular vectors.

The diagonal entries of Σ are called their corresponding singular values.

Assume we have a matrix M whose SVD form is $U\Sigma V$, with non-decreasing order of singular values in the diagonal of Σ . For definition of the SVD form and singular values see theorem 2.5.1. we will denote:

$$U(M) \stackrel{\text{def}}{=} U \quad \Sigma(M) \stackrel{\text{def}}{=} \Sigma \quad V(M) \stackrel{\text{def}}{=} V$$

We also name the first and last column of U :

$$U(M) = \left(\begin{array}{c|c|c} \vec{\sigma}_{\min}(M) & \cdots & \vec{\sigma}_{\max}(M) \\ \hline & & \end{array} \right)$$

and the first and last diagonal entries of Σ (“the singular values”) by:

$$\Sigma(M) = \left(\begin{array}{ccc} \sigma_{\min}(M) & & \\ & \ddots & \\ & & \sigma_{\max}(M) \end{array} \right)$$

Theorem 2.5.2.

$$\min_{\|x\|=1} \|M^t x\| = \|M^t \vec{\sigma}_{min}(M)\| = \sigma_{min}(M).$$

That is, among all unit vectors the first left singular vector gets this expression to minimum. This minimum is equals to its corresponding singular value (which is also called $\|M\|$).

Proof. For readability, we will denote $V(M), \Sigma(M), U(M), \vec{\sigma}_{min}(M), \sigma_{min}(M)$ by $V, \Sigma, U, \vec{\sigma}_{min}, \sigma_{min}$, respectively.

First we show that:

$$\min_{\|x\|=1} \|M^t x\| \geq \sigma_{min}.$$

Lets denote by x^* the unit vector that minimize the expression, that is:

$$\min_{\|x\|=1} \|M^t x\| = \|M^t x^*\| \quad (\text{A.2.10})$$

By theorem 2.5.1:

$$\|M^t x^*\| = \|[U\Sigma V]^t x^*\| = \|V^t \Sigma U^t x^*\|$$

V is orthogonal, like its transpose, and by theorem A.1.2 :

$$\|M^t x^*\| = \|V^t \Sigma U^t x^*\| = \|V^t (\Sigma U^t x^*)\| = \|\Sigma U^t x^*\| \quad (\text{A.2.11})$$

By theorem A.1.2 the vector $x' = U^t x^*$ is also a unit vector. Together with (A.2.11) and (A.2.10):

$$\min_{\|x\|=1} \|\Sigma x\| \leq \|\Sigma x'\| = \|\Sigma U^t x^*\| = \|M^t x^*\| = \min_{\|x\|=1} \|M^t x\| \quad (\text{A.2.12})$$

But because Σ is a diagonal matrix, the lower bound for the left inequality side is:

$$\begin{aligned} \min_{\|x\|=1} \|\Sigma x\| &= \min_{\|x\|=1} \sqrt{(\sigma_{min} x_1)^2 + \dots + (\sigma_{max} x_p)^2} \\ &\geq \min_{\|x\|=1} \sqrt{(\sigma_{min} x_1)^2 + \dots + (\sigma_{min} x_p)^2} \\ &= \min_{\|x\|=1} \sqrt{\sigma_{min}^2 (x_1^2 + \dots + x_p^2)} \\ &= \sqrt{\sigma_{min}^2} = \sigma_{min} \end{aligned}$$

and this together with (A.2.12) gives:

$$\min_{\|x\|=1} \|M^t x\| \geq \min_{\|x\|=1} \|\Sigma x\| \geq \sigma_{min} \quad (\text{A.2.13})$$

Now all is left to show is that $x^* = \vec{\sigma}_{min}$ reaches the above lower bound. As we did in (A.2.11):

$$\|M^t \vec{\sigma}_{min}\| = \|V^t \Sigma U^t \vec{\sigma}_{min}\| = \|\Sigma U^t \vec{\sigma}_{min}\|$$

and while remembering that the columns of U are orthonormal and $\vec{\sigma}_{min}$ is one of them:

$$\|M^t \vec{\sigma}_{min}\| = \|\Sigma U^t \vec{\sigma}_{min}\| = \left\| \Sigma \begin{pmatrix} 1 \\ 0 \\ \vdots \\ 0 \end{pmatrix} \right\| = \sigma_{min}$$

this equation together with (A.2.13) gives the theorem:

$$\min_{\|x\|=1} \|M^t x\| = \|M^t \vec{\sigma}_{min}\| = \sigma_{min}.$$

□

Theorem 2.5.3.

$$\ell^* = \arg \min_{\ell} \sum_{i=1}^n \text{dist}^2(\ell, p_i)$$

where $\ell^* = \text{line}(\vec{\sigma}_{max}, \vec{\sigma}_{min}^t \bar{p})$, and this sum of squared distances equals to:

$$\sigma_{min}^2 = \sum_{i=1}^n \text{dist}^2(\ell^*, p_i)$$

Proof. By theorems 2.4.1 for every $c \in \mathbb{R}$ and a unit vector v :

$$\sum_{i=1}^n \text{dist}^2(\text{line}(v, c), p_i) \geq \sum_{i=1}^n \text{dist}^2(\text{line}(v, v_{\perp}^t \bar{p}), p_i) \quad (\text{A.2.14})$$

Using theorem 2.3.1:

$$\begin{aligned} \sum_{i=1}^n \text{dist}^2(\text{line}(v, v_{\perp}^t \bar{p}), p_i) &= \sum_{i=1}^n (v_{\perp}^t p_i - v_{\perp}^t \bar{p})^2 \\ &= \sum_{i=1}^n (v_{\perp}^t (p_i - \bar{p}))^2 \\ &= \sum_{i=1}^n ((p_i - \bar{p})^t v_{\perp})^2 \\ &= \|\tilde{A}^t v_{\perp}\|^2 \end{aligned} \quad (\text{A.2.15})$$

(A.2.14) together with (A.2.15) gives:

$$\sum_{i=1}^n \text{dist}^2(\text{line}(v, c), p_i) \geq \|\tilde{A}^t v_\perp\|^2$$

Using theorem 2.5.2:

$$\sum_{i=1}^n \text{dist}^2(\text{line}(v, c), p_i) \geq \|\tilde{A}^t v_\perp\|^2 \geq \|\tilde{A}^t \vec{\sigma}_{min}\|^2 = \sigma_{min}^2.$$

Because $\vec{\sigma}_{min}$ and $\vec{\sigma}_{max}$ are two orthogonal vectors in \mathbb{R}^2 , by last equation and (A.2.15) we've got:

$$\begin{aligned} \sum_{i=1}^n \text{dist}^2(\text{line}(v, c), p_i) &\geq \|\tilde{A}^t \vec{\sigma}_{min}\|^2 \\ &= \sum_{i=1}^n \text{dist}^2(\text{line}(\vec{\sigma}_{max}, \vec{\sigma}_{min}^t \bar{p}), p_i) \\ &= \sum_{i=1}^n \text{dist}^2(\ell^*, p_i) \\ &= \sigma_{min}^2 \end{aligned}$$

□

A.3 Fast Updating the 1-Line-Mean

A.3.1 Definitions

Let $A = \{a_i\}$ and $B = \{b_i\}$ be any two matrices of size $2 \times n$ and $2 \times n'$ with a_i and b_i as their i 'th column vector, respectively.

The first row vector of A will be denoted by x , and the second row will be denoted by y , that is:

$$x = (a_{11}, a_{12}, \dots, a_{1n}) = (x_1, x_2, \dots, x_n)$$

$$y = (a_{21}, a_{22}, \dots, a_{2n}) = (y_1, y_2, \dots, y_n)$$

and the same for the rows of B that will be denoted by x' and y' .

\hat{a} — will be the sum of A 's columns:

$$\hat{a} \stackrel{\text{def}}{=} \sum_{i=1}^n a_i$$

\bar{a} — the centroid of A :

$$\bar{a} \stackrel{\text{def}}{=} \frac{1}{n} \sum_{i=1}^n a_i = \frac{\hat{a}}{n}$$

\tilde{A} — will be the matrix A after subtracting its centroid from each column:

$$\tilde{A} = \left(\begin{array}{c|c|c|c} & & & \\ a_1 - \bar{a} & a_2 - \bar{a} & \cdots & a_n - \bar{a} \\ & & & \end{array} \right)$$

$A|B$ — the matrix A after adding the column vectors of B right after the existing columns.

$A \setminus B$ — the matrix A after removing its columns vectors that were included in B :

$$A \setminus B \stackrel{\text{def}}{=} \{a_i\} \setminus \{b_i\}$$

Remarks:

- The operator will be defined only when $B \subseteq A$, and $\{a_i\} \cap \{b_i\} = \emptyset$.
- The order of the remaining columns will not be changed.

$\text{vec}(A)$ — a function from a $2 \times n$ matrix to a vector $u \in \mathbb{R}^5$ that contains summary information about A :

$$\text{vec}(A) \stackrel{\text{def}}{=} \left(n, \sum_{i=1}^n x_i, \sum_{i=1}^n y_i, \sum_{i=1}^n x_i^2 - y_i^2, x^t y \right) \quad (\text{A.3.1})$$

$\sigma_{\min}(A)$ — the smallest singular value of A . See Theorem 2.5.1 for details.

$$\sigma_{\min}^2(A) = \sigma_{\min}(A)^2.$$

$\lambda_{\min}(A)$ — the smallest eigenvalue of A .

$\hat{\lambda}(u)$ — a function from \mathbb{R}^5 to \mathbb{R} :

$$\hat{\lambda}(u) \stackrel{\text{def}}{=} \frac{1}{u_1} (u_2^2 + u_3^2) + \left\| \begin{pmatrix} u_4 - \frac{1}{u_1}(u_2^2 - u_3^2) \\ 2 \left(u_5 - \frac{1}{u_1} u_2 u_3 \right) \end{pmatrix} \right\| \quad (\text{A.3.2})$$

A.3.2 Theorems

Theorem 3.0.4.

$$\lambda_{\min}(\tilde{A}\tilde{A}^t) = -\frac{1}{2} \left(\hat{\lambda}(\text{vec}(A)) - \sum_{i=1}^n x_i^2 + y_i^2 \right)$$

Proof. Notice that:

$$\sum_{i=1}^n \bar{a}a_i^t = \bar{a} \sum_{i=1}^n a_i^t = \frac{1}{n} \hat{a}\hat{a}^t \quad (\text{A.3.3})$$

and as a result:

$$\sum_{i=1}^n a_i\bar{a}^t = \left(\sum_{i=1}^n \bar{a}a_i^t \right)^t = \frac{1}{n} \hat{a}\hat{a}^t \quad (\text{A.3.4})$$

So we can write $\tilde{A}\tilde{A}^t$ as a function of AA^t , n and \hat{a} using its scatter form (see thorem A.1.1):

$$\begin{aligned} \tilde{A}\tilde{A}^t &= \sum_{i=1}^n (a_i - \bar{a})(a_i - \bar{a})^t = \sum_{i=1}^n a_i a_i^t - \sum_{i=1}^n a_i \bar{a}^t - \sum_{i=1}^n \bar{a} a_i^t + n\bar{a}\bar{a}^t \\ &\text{and using (A.3.3), (A.3.4) and theorem A.1.1:} \end{aligned} \quad (\text{A.3.5})$$

$$= AA^t - \frac{2}{n} \hat{a}\hat{a}^t + n\bar{a}\bar{a}^t = AA^t - \frac{2}{n} \hat{a}\hat{a}^t + \frac{1}{n} \hat{a}\hat{a}^t = AA^t - \frac{1}{n} \hat{a}\hat{a}^t$$

For simplicity, we'll name the entries of the symmetric matrix $\tilde{A}\tilde{A}^t$:

$$\begin{pmatrix} w_{11} & w_{12} \\ w_{12} & w_{22} \end{pmatrix} \stackrel{\text{def}}{=} \tilde{A}\tilde{A}^t$$

which means, by (A.3.5) and (A.3.1), that if we denote $u = \text{vec}(A)$:

$$\begin{aligned} w_{12} &= x^t y - \frac{1}{n} \left(\sum_{i=1}^n x_i \right) \left(\sum_{i=1}^n y_i \right) \\ &= u_5 - \frac{1}{u_1} u_2 u_3 \\ w_{11} - w_{22} &= \left(\sum_{i=1}^n x_i^2 \right) - \frac{1}{n} \left(\sum_{i=1}^n x_i \right)^2 - \left[\left(\sum_{i=1}^n y_i^2 \right) - \frac{1}{n} \left(\sum_{i=1}^n y_i \right)^2 \right] \\ &= \sum_{i=1}^n (x_i^2 - y_i^2) - \frac{1}{n} \left[\left(\sum_{i=1}^n x_i \right)^2 - \left(\sum_{i=1}^n y_i \right)^2 \right] \\ &= u_4 - \frac{1}{u_1} (u_2^2 - u_3^2) \\ w_{11} + w_{22} &= \sum_{i=1}^n (x_i^2 + y_i^2) - \frac{1}{u_1} (u_2^2 + u_3^2) \end{aligned} \quad (\text{A.3.6})$$

To compute the eigenvalue $\lambda_{\min}(\tilde{A}\tilde{A}^t)$, we'll calculate the character polynomial and then set it to zero:

$$\begin{aligned} |\tilde{A}\tilde{A}^t - \lambda I| &= \begin{vmatrix} w_{11} - \lambda & w_{12} \\ w_{12} & w_{22} - \lambda \end{vmatrix} = (w_{11} - \lambda)(w_{22} - \lambda) - w_{12}^2 \\ &= \lambda^2 - \lambda(w_{11} + w_{22}) + w_{11}w_{22} - w_{12}^2 = 0 \end{aligned} \quad (\text{A.3.7})$$

because $\tilde{A}\tilde{A}^t$ is a symmetric matrix $\Delta \geq 0$, and there must be two real eigenvalues. By applying (A.3.6) the smaller root of (A.3.7) is:

$$\begin{aligned} \lambda_{\min}(\tilde{A}\tilde{A}^t) &= \frac{1}{2} \left(w_{11} + w_{22} - \sqrt{(w_{11} + w_{22})^2 - 4(w_{11}w_{22} - w_{12}^2)} \right) \\ &= -\frac{1}{2} \left(\sqrt{(w_{11} - w_{22})^2 + 4w_{12}^2} - (w_{11} + w_{22}) \right) \\ &= -\frac{1}{2} \left(\left\| \begin{matrix} w_{11} - w_{22} \\ 2w_{12} \end{matrix} \right\| - (w_{11} + w_{22}) \right) \\ &= -\frac{1}{2} \left(\left\| \begin{matrix} u_4 - \frac{1}{u_1}(u_2^2 - u_3^2) \\ 2(u_5 - \frac{1}{u_1}u_2u_3) \end{matrix} \right\| + \frac{1}{u_1}(u_2^2 + u_3^2) - \sum_{i=1}^n x_i^2 + y_i^2 \right) \\ &= -\frac{1}{2} \left(\hat{\lambda}(\text{vec}(A)) - \sum_{i=1}^n x_i^2 + y_i^2 \right) \end{aligned}$$

□

Theorem 3.1.1.

$$\text{vec}([A|B]) = \text{vec}(A) + \text{vec}(B)$$

Proof.

$$\begin{aligned} \text{vec}([A|B]) &= \\ &= (n + n', \sum_{i=1}^n x_i + \sum_{i=1}^{n'} x'_i, \sum_{i=1}^n y_i + \sum_{i=1}^{n'} y'_i, \sum_{i=1}^n x_i^2 - y_i^2 + \sum_{i=1}^{n'} x_i'^2 - y_i'^2, x^t y + x'^t y') \\ &= (n, \sum_{i=1}^n x_i, \sum_{i=1}^n y_i, \sum_{i=1}^n x_i^2 - y_i^2, x^t y) + (n', \sum_{i=1}^{n'} x'_i, \sum_{i=1}^{n'} y'_i, \sum_{i=1}^{n'} x_i'^2 - y_i'^2, x'^t y') \\ &= \text{vec}(A) + \text{vec}(B) \end{aligned}$$

□

Theorem 3.2.1.

$$\text{vec}(A \setminus B) = \text{vec}(A) - \text{vec}(B)$$

Proof.

$$\begin{aligned}
\text{vec}(A \setminus B) &= \\
&= (n - n', \sum_{i=1}^n x_i - \sum_{i=1}^{n'} x'_i, \sum_{i=1}^n y_i - \sum_{i=1}^{n'} y'_i, \sum_{i=1}^n x_i^2 - y_i^2 - \sum_{i=1}^{n'} x_i'^2 - y_i'^2, x^t y + x'^t y') \\
&= (n, \sum_{i=1}^n x_i, \sum_{i=1}^n y_i, \sum_{i=1}^n x_i^2 - y_i^2, x^t y) - (n', \sum_{i=1}^{n'} x'_i, \sum_{i=1}^{n'} y'_i, \sum_{i=1}^{n'} x_i'^2 - y_i'^2, x'^t y') \\
&= \text{vec}(A) - \text{vec}(B)
\end{aligned}$$

□

A.4 2-Line-Means

A.4.1 Definitions

In the following definitions we will assume that $p' \in S$ and ℓ is a line such that $\ell = \text{line}(v, c)$:

p_i — a column vector that represents a point on the plane:

$$p_i \stackrel{\text{def}}{=} \begin{pmatrix} x_i \\ y_i \end{pmatrix} \quad \text{where } (x_i, y_i) \in \mathbb{R}^2. \quad (\text{A.4.1})$$

\tilde{A} — will be the matrix A after subtracting its centroid from each column:

$$\tilde{A} = \left(\begin{array}{c|c|c|c} & & & \\ a_1 - \bar{a} & a_2 - \bar{a} & \cdots & a_n - \bar{a} \\ & & & \end{array} \right)$$

$A|B$ — the matrix A after adding the column vectors of B right after the existing columns.

$A \setminus B$ — the matrix A after removing its columns vectors that were included in B :

$$A \setminus B \stackrel{\text{def}}{=} \{a_i\} \setminus \{b_i\}$$

Remarks:

- The operator will be defined only when $B \subseteq A$, and $\{a_i\} \cap \{b_i\} = \emptyset$.
- The order of the remaining columns will not be changed.

$B \subseteq A$ — will means that all the column vectors of B are in A . That is:
 $\{b_i\} \subseteq \{a_i\}$

$\text{vec}(A)$ — a function from a $2 \times n$ matrix to a vector $u \in \mathbb{R}^5$ that contains summary information about A :

$$\text{vec}(A) \stackrel{\text{def}}{=} (n, \sum_{i=1}^n x_i, \sum_{i=1}^n y_i, \sum_{i=1}^n x_i^2 - y_i^2, x^t y) \quad (\text{A.4.2})$$

$\sigma_{\min}(A)$ — the smallest singular value of A . See Theorem 2.5.1 for details.

$$\sigma_{\min}^2(A) = \sigma_{\min}(A)^2.$$

S — a set of n points in the plane:

$$S \stackrel{\text{def}}{=} \{p_i\} \quad \text{for every } 1 \leq i \leq n. \quad (\text{A.4.3})$$

$\text{line}(v, c)$ — where $v \in \mathbb{R}^2$, $c \in \mathbb{R}$ and $\|v\| = 1$ represents a line in the plane with the unit vector v as its direction and c as its shortest distance from zero. More formally, this is the set of points:

$$\{cv_{\perp} + kv \mid k \in \mathbb{R}\} \quad (\text{A.4.4})$$

$\text{dist}(\ell, p)$ — the shortest Euclidean distance between the point p and any point on the line ℓ . More formally

$$\text{dist}(\ell, p) \stackrel{\text{def}}{=} \min_{p_L \in \ell} \|p - p_L\| \quad (\text{A.4.5})$$

$$\text{dist}^2(\ell, p) - \{\text{dist}(\ell, p)\}^2$$

$S_1^A(\ell, p')$ — subset of points of S . Formally:

$$S_1^A(\ell, p') \stackrel{\text{def}}{=} \{p \in S \mid v^t(p - p') \leq 0\} \cap \{p \in S \mid v_{\perp}^t p - c > 0\} \quad (\text{A.4.6})$$

$S_1^B(\ell, p')$ — subset of points of S . Formally:

$$S_1^B(\ell, p') \stackrel{\text{def}}{=} \{p \in S \mid v^t(p - p') > 0\} \cap \{p \in S \mid v_{\perp}^t p - c \leq 0\} \quad (\text{A.4.7})$$

$S_1(\ell, p')$ — is the union of the points in the last two sets:

$$S_1(\ell, p') \stackrel{\text{def}}{=} S_1^A \cup S_1^B \quad (\text{A.4.8})$$

$S_2(\ell, p')$ — is the rest of the points that are not included in $S_1(\ell, p')$:

$$S_2(\ell, p') \stackrel{\text{def}}{=} S \setminus S_1(\ell, p')$$

ℓ_1^*, ℓ_2^* — Two lines that minimize the sum of squared distances from the points in S . Such two lines are the requested result for the 2-line-means problem. Formally:

$$(\ell_1^*, \ell_2^*) \stackrel{\text{def}}{=} \arg \min_{(\ell_1, \ell_2)} \sum_{i=1}^n \min\{\text{dist}(\ell_1, p_i), \text{dist}(\ell_2, p_i)\}^2 \quad (\text{A.4.9})$$

S_1^* — The points in S that are closer to ℓ_1^* than to ℓ_2^* :

$$S_1^* \stackrel{\text{def}}{=} \{p \in S \mid \text{dist}(\ell_1^*, p) \leq \text{dist}(\ell_2^*, p)\}$$

S_2^* — All the points that are not in S_1^* , that is, closer to ℓ_2^* than to ℓ_1^* :

$$S_2^* \stackrel{\text{def}}{=} S \setminus S_1^*$$

v_x, v_y — where $v \in \mathbb{R}^2$ means the first and second coordinate of v , respectively, that is:

$$v = (v_x, v_y)^t. \quad (\text{A.4.10})$$

Indexed points or vectors will be written as: $p_i = (p_{ix}, p_{iy})$ or $v_i = (v_{ix}, v_{iy})$, respectively.

$\text{dir}(p, q)$ — where $p, q \in S$, is a unit vector, u , that represents the direction from one point to another, as in Figure The sign of the vector is determined such that $u_x \geq 0$. That is:

$$\text{dir}(p, q) \stackrel{\text{def}}{=} \begin{cases} \frac{p-q}{\|p-q\|} & p_x \geq q_x \\ \frac{q-p}{\|q-p\|} & p_x < q_x \end{cases} \quad (\text{A.4.11})$$

D — will be the set:

$$D \stackrel{\text{def}}{=} \{\text{dir}(p, q) \mid p, q \in S\} \quad (\text{A.4.12})$$

$\theta(v)$ — where $v \in \mathbb{R}^2$ is a unit vector:

$$\theta(v) \stackrel{\text{def}}{=} \begin{cases} \infty & v = (0, 1)^t \\ -\infty & v = (0, -1)^t \\ \frac{v_y}{v_x} & \text{otherwise.} \end{cases} \quad (\text{A.4.13})$$

represents a value which is proportional to the angle of the vector from the positive x-axis. Specifically, \tan^{-1} of that angle.

$G(A, B)$ — where $B \subseteq A$, is a function from two matrices, each with two rows, to \mathbb{R} :

$$G(A, B) \stackrel{\text{def}}{=} \sigma_{\min}^2(\widetilde{A \setminus B}) + \sigma_{\min}^2(\tilde{B}) \quad (\text{A.4.14})$$

$\vec{G}(u, w)$ — is a function from two vectors, $u, w \in \mathbb{R}^5$ to \mathbb{R} :

$$\vec{G}(u, w) \stackrel{\text{def}}{=} -\frac{1}{2} \left[\hat{\lambda}(u - w) + \hat{\lambda}(w) \right] \quad (\text{A.4.15})$$

A.4.2 Theorems

Theorem 4.2.1. *Pair of lines that solves the 2-line-means problem are the 1-line-mean of S_1^* , and the 1-line-mean of S_2^* . That is:*

$$(\ell_1^*, \ell_2^*) = (\ell'_1, \ell'_2)$$

where:

$$\ell'_1 = \arg \min_{\ell} \sum_{p \in S_1^*} \text{dist}^2(p, \ell) \quad \ell'_2 = \arg \min_{\ell} \sum_{p \in S_2^*} \text{dist}^2(p, \ell). \quad (\text{A.4.16})$$

Proof. By definition of S_1^* and S_2^* , and the fact that S is their union:

$$\begin{aligned} & \sum_{i=1}^n \min\{\text{dist}(\ell_1^*, p_i), \text{dist}(\ell_2^*, p_i)\}^2 \\ &= \sum_{p \in S_1^*} \min\{\text{dist}(p, \ell_1^*), \text{dist}(p, \ell_2^*)\}^2 + \sum_{p \in S_2^*} \min\{\text{dist}(p, \ell_1^*), \text{dist}(p, \ell_2^*)\}^2 \\ &= \sum_{p \in S_1^*} \text{dist}^2(p, \ell_1^*) + \sum_{p \in S_2^*} \text{dist}^2(p, \ell_2^*) \\ &\geq \min_{\ell} \sum_{p \in S_1^*} \text{dist}^2(p, \ell) + \min_{\ell} \sum_{p \in S_2^*} \text{dist}^2(p, \ell) \\ &= \sum_{p \in S_1^*} \text{dist}^2(p, \ell'_1) + \sum_{p \in S_2^*} \text{dist}^2(p, \ell'_2) \\ &\geq \sum_{p \in S_1^*} \min\{\text{dist}(p, \ell'_1), \text{dist}(p, \ell'_2)\}^2 + \sum_{p \in S_2^*} \min\{\text{dist}(p, \ell'_1), \text{dist}(p, \ell'_2)\}^2 \\ &= \sum_{i=1}^n \min\{\text{dist}(\ell'_1, p_i), \text{dist}(\ell'_2, p_i)\}^2 \end{aligned} \quad (\text{A.4.17})$$

But by definition of ℓ_1^* and ℓ_2^* :

$$\begin{aligned} \sum_{i=1}^n \min\{\text{dist}(\ell_1^*, p_i), \text{dist}(\ell_2^*, p_i)\}^2 &= \min_{(\ell_1, \ell_2)} \sum_{i=1}^n \min\{\text{dist}(\ell_1, p_i), \text{dist}(\ell_2, p_i)\}^2 \\ &\leq \sum_{i=1}^n \min\{\text{dist}(\ell'_1, p_i), \text{dist}(\ell'_2, p_i)\}^2 \end{aligned} \quad (\text{A.4.18})$$

By (A.4.17) together with (A.4.18) we get:

$$\sum_{i=1}^n \min\{\text{dist}(\ell_1^*, p_i), \text{dist}(\ell_2^*, p_i)\}^2 = \sum_{i=1}^n \min\{\text{dist}(\ell'_1, p_i), \text{dist}(\ell'_2, p_i)\}^2$$

□

Theorem 4.6.1.

$$G(A, B) = \vec{G}(\text{vec}(A), \text{vec}(B)) + \frac{1}{2} \sum_{i=1}^n x_i^2 + y_i^2$$

Proof. by definition (A.4.14):

$$G(A, B) \stackrel{\text{def}}{=} \sigma_{\min}^2(\tilde{C}) + \sigma_{\min}^2(\tilde{B})$$

where $C = A \setminus B$.

It is known that for any matrix M :

$$\sigma_i^2(M) = \lambda_i(MM^t) \quad \text{for any } 1 \leq i \leq r \quad (\text{A.4.19})$$

where r is the rank of M and λ_i is its i 'th eigenvalue. From the definition we get:

$$G(A, B) = \lambda_{\min}(\tilde{C}\tilde{C}^t) + \lambda_{\min}(\tilde{B}\tilde{B}^t)$$

By Theorem 3.0.4 and the fact that the entries' sum of C is the sum of A minus the entries of B :

$$\begin{aligned} G(A, B) &= -\frac{1}{2} \left(\hat{\lambda}(\text{vec}(C)) - \sum_{i=1}^n x_i^2 + y_i^2 + \sum_{i=1}^{n'} x_i'^2 + y_i'^2 \right) \\ &\quad - \frac{1}{2} \left(\hat{\lambda}(\text{vec}(B)) - \sum_{i=1}^{n'} x_i'^2 + y_i'^2 \right) \quad (\text{A.4.20}) \\ &= -\frac{1}{2} \left[\hat{\lambda}(\text{vec}(C)) + \hat{\lambda}(\text{vec}(B)) \right] + \frac{1}{2} \sum_{i=1}^n x_i^2 + y_i^2 \end{aligned}$$

from theorem 3.2.1 we get :

$$\hat{\lambda}(\text{vec}(C)) = \hat{\lambda}(\text{vec}(A) - \text{vec}(B))$$

putting this in (A.4.20) gives:

$$G(A, B) = -\frac{1}{2} \left[\hat{\lambda}(\text{vec}(A) - \text{vec}(B)) + \hat{\lambda}(\text{vec}(B)) \right] + \frac{1}{2} \sum_{i=1}^n x_i^2 + y_i^2$$

and using definition (A.4.15):

$$G(A, B) = \vec{G}(\text{vec}(A), \text{vec}(B)) + \frac{1}{2} \sum_{i=1}^n x_i^2 + y_i^2$$

□

Theorem 4.6.2. *if $\mathcal{S} \subseteq \{B|B \subseteq A\}$, which means \mathcal{S} is any set of matrices that their columns are subset of A 's columns, then:*

$$\arg \min_{B \in \mathcal{S}} G(A, B) = \arg \min_{B \in \mathcal{S}} \vec{G}(\text{vec}(A), \text{vec}(B)).$$

Proof. By elimination we assume there exist two matrices, $B_1, B_2 \in \mathcal{S}$ such that:

$$\vec{G}(\text{vec}(A), \text{vec}(B_1)) \geq \vec{G}(\text{vec}(A), \text{vec}(B_2)), \quad (\text{A.4.21})$$

and

$$G(A, B_1) < G(A, B_2). \quad (\text{A.4.22})$$

By (A.4.21) and theorem 4.6.1:

$$\begin{aligned} G(A, B_1) &= \vec{G}(\text{vec}(A), \text{vec}(B_1)) + \frac{1}{2} \sum_{i=1}^n x_i^2 + y_i^2 \\ &\geq \vec{G}(\text{vec}(A), \text{vec}(B_2)) + \frac{1}{2} \sum_{i=1}^n x_i^2 + y_i^2 \\ &= G(A, B_2) \end{aligned}$$

which contradicts (A.4.22). \square

Theorem 4.6.3. *Assume that $B \subseteq A$, and suppose that we want to add the column vectors of B^+ to B and remove vectors that are included in B^- , that is:*

$$B' = ([B|B^+] \setminus B^-). \quad (\text{A.4.23})$$

where B^+ and B^- are matrices with two rows each, and $B^- \subseteq B$.

Then:

$$\vec{G}(\text{vec}(A), \text{vec}(B')) = \vec{G}(\text{vec}(A), \text{vec}(B) + \text{vec}(B^+) - \text{vec}(B^-))$$

Proof. Using theorem 3.1.1 and 3.2.1:

$$\text{vec}(B') = \text{vec}([B|B^+]) - \text{vec}(B^-) = \text{vec}(B) + \text{vec}(B^+) - \text{vec}(B^-) \quad (\text{A.4.24})$$

putting this into (A.4.24) gives:

$$\vec{G}(\text{vec}(A), \text{vec}(B')) = \vec{G}(\text{vec}(A), \text{vec}(B) + \text{vec}(B^+) - \text{vec}(B^-))$$

\square

Theorem 5.3.1. *Let $u, w \in D$ such that $\theta(u) < \theta(w)$, and let $p, q \in S$ be two points in S such that $u^t p < u^t q$. Then:*

$$w^t p \geq w^t q \Leftrightarrow \theta(u) < \theta(v_\perp) \leq \theta(w)$$

where $v = \frac{p-q}{\|p-q\|}$.

Proof. We denote the vectors as in (A.4.10), for example: $u = (u_x, u_y)$. First, observe the following:

$$\begin{aligned} u^t p < u^t q &\Leftrightarrow u^t p - u^t q < 0 \Leftrightarrow u^t(p - q) < 0 \\ &\Leftrightarrow u^t v < 0 \Leftrightarrow u_x v_x + u_y v_y < 0 \\ &\Leftrightarrow u_y v_y < -u_x v_x \end{aligned} \quad (\text{A.4.25})$$

Similarly,

$$w^t p \geq w^t q \Leftrightarrow w_y v_y \geq -w_x v_x \quad (\text{A.4.26})$$

Notice also that if $u_x = 0$ then $\theta(u) = \pm\infty$ by definition (A.4.13), but by the assumption $\theta(u) < \theta(w)$, which left us with $\theta(u) = -\infty$. This means:

$$u_x = 0 \Rightarrow \theta(u) = -\infty \leq \theta(v_\perp) \quad (\text{A.4.27})$$

and using this with definition (A.4.13) of θ :

$$u_x = 0 \Rightarrow \theta(u) = -\infty \Rightarrow u_y = -1. \quad (\text{A.4.28})$$

Similarly, we can conclude:

$$w_x = 0 \Rightarrow \theta(w) = \infty \geq \theta(v_\perp) \quad (\text{A.4.29})$$

and again by (A.4.13):

$$w_x = 0 \Rightarrow \theta(w) = \infty \Rightarrow w_y = 1. \quad (\text{A.4.30})$$

Now we can begin the proof:

(\Rightarrow) By (A.4.25), (A.4.26) and the assumptions we can write:

$$u_y v_y < -u_x v_x \quad (\text{A.4.31})$$

$$w_y v_y \geq -w_x v_x \quad (\text{A.4.32})$$

Notice also that by definition (A.4.11) of dir:

$$u_x \geq 0 \text{ and } w_x \geq 0 \quad (\text{A.4.33})$$

The proof will be separated into 3 cases:

I) $v_y > 0$, II) $v_y = 0$, III) $v_y < 0$.

I) First we prove that:

$$\theta(u) < \theta(v_{\perp}). \quad (\text{A.4.34})$$

Notice that $v_{\perp} = (v_y, -v_x)$, so:

$$v_y > 0 \Rightarrow v_{\perp x} = v_y > 0 \Rightarrow \theta(v_{\perp}) > -\infty$$

By this and (A.4.27):

$$u_x = 0 \Rightarrow \theta(u) < \theta(v_{\perp}).$$

If $u_x > 0$ then we can divide (A.4.31) by $u_x v_y > 0$:

$$\frac{u_y}{u_x} < -\frac{v_x}{v_y} \Rightarrow \theta(u) < \theta(v_{\perp}).$$

With similar proof:

$$\theta(w) \geq \theta(v_{\perp}) \quad (\text{A.4.35})$$

that's because by (A.4.29)

$$w_x = 0 \Rightarrow \theta(w) \geq \theta(v_{\perp}).$$

and when $w_x > 0$ we can divide (A.4.32) by $w_x v_y > 0$:

$$\frac{w_y}{w_x} \geq -\frac{v_x}{v_y} \Rightarrow \theta(w) \geq \theta(v_{\perp}).$$

By dir definition there are no other possibilities, and we can summarize (A.4.34) and (A.4.35) to the proof of case I:

$$\theta(u) < \theta(v_{\perp}) \leq \theta(w).$$

II) Here $v_y = 0$ and because v_y is a unit vector:

$$v_y = 0 \Rightarrow v_x = \pm 1 \quad (\text{A.4.36})$$

and by (A.4.31):

$$u_y v_y < -u_x v_x \Rightarrow 0 < -u_x v_x$$

But $u_x \geq 0$ by definition (A.4.11) of dir so:

$$u_y v_y < -u_x v_x \Rightarrow 0 < -v_x \Rightarrow v_x = -1 \quad (\text{A.4.37})$$

so:

$$v_x = -1 \Rightarrow v = (-1, 0) \Rightarrow v_{\perp} = (0, 1) \Rightarrow \theta(v_{\perp}) = \infty \quad (\text{A.4.38})$$

By the theorem's assumption:

$$\theta(u) < \theta(w) \Rightarrow \theta(u) < \infty$$

this together with (A.4.38) gives:

$$\theta(u) < \theta(v_\perp). \quad (\text{A.4.39})$$

For the second part, $v_x = -1$ by (A.4.37). Substituting in (A.4.32) gives:

$$w_y v_y \geq -w_x v_x \Rightarrow 0 \geq w_x \Rightarrow w_x = 0 \Rightarrow \theta(w) = \pm\infty$$

but because the theorem's assumption $\theta(u) < \theta(w)$ we can conclude:

$$\theta(w) > -\infty \Rightarrow \theta(w) = \infty = \theta(v_\perp)$$

This, together with (A.4.39) gives the proof for this case:

$$\theta(u) < \theta(v_\perp) \leq \theta(w)$$

III) Here $v_y < 0$, which also means $u_x > 0$. This is because from dir definition $u_x \geq 0$ and by (A.4.28):

$$u_x = 0 \Rightarrow u_y = -1 \Rightarrow u_y v_y > 0 \Rightarrow u_y v_y > -v_x u_x$$

which contradicts (A.4.31).

It is also true that $w_x > 0$: if $w_x = 0$ then from (A.4.30) we get $w_y = 1$, and from (A.4.32):

$$w_y v_y \geq -w_x v_x \Rightarrow v_y \geq 0$$

which contradicts this case.

We left with the option that $u_x > 0$ and $w_x > 0$, which is also impossible. This is because $u_x v_y < 0$, so by (A.4.31):

$$u_y v_y < -u_x v_x \Rightarrow \frac{u_y}{u_x} > -\frac{v_x}{v_y} \Rightarrow \theta(u) > \theta(v_\perp) \quad (\text{A.4.40})$$

Similarly, because $w_x v_y < 0$ and (A.4.32):

$$w_y v_y \geq -w_x v_x \Rightarrow \frac{w_y}{w_x} \leq -\frac{v_x}{v_y} \Rightarrow \theta(w) \leq \theta(v_\perp) \quad (\text{A.4.41})$$

But putting (A.4.40) and (A.4.41) together gives:

$$\theta(w) \leq \theta(v_\perp) < \theta(u) \Rightarrow \theta(w) < \theta(u)$$

which contradicts the theorem's assumption.

(\Leftarrow) Here we assume $\theta(u) < \theta(v_\perp) \leq \theta(w)$, $u^t p < w^t q$, and prove that $w^t p \geq w^t q$. This side of the proof will also divide to the same three cases.

I) For $v_y > 0$, if $w_x > 0$ we have $w_x v_y > 0$, so by the assumption:

$$\theta(w) \geq \theta(v_\perp) \Rightarrow \frac{w_y}{w_x} \geq -\frac{v_x}{v_y} \Rightarrow w_y v_y \geq -v_x w_x$$

by (A.4.26):

$$w_y v_y \geq -v_x w_x \Rightarrow w^t p \geq w^t q.$$

By definition (A.4.11) of dir , the only other case is $w_x = 0$. Using (A.4.30) we have:

$$w_x = 0 \Rightarrow w_y = 1 \Rightarrow w_y v_y > 0 \Rightarrow w_y v_y > -w_x w_x$$

and using (A.4.26) again:

$$w_y v_y > -w_x w_x \Rightarrow w^t p \geq w^t q$$

II) Here $v_y = 0$ and $\theta(v_\perp) = \pm\infty$. By the first assumption

$$\theta(u) < \theta(v_\perp) \Rightarrow \theta(v_\perp) > -\infty \Rightarrow \theta(v_\perp) = \infty.$$

and using the other part of the first assumption:

$$\theta(w) \geq \theta(v_\perp) \Rightarrow \theta(w) = \infty \Rightarrow w_x = 0 \Rightarrow -v_x w_x = 0.$$

And because in this case we assume $v_y = 0$:

$$-v_x w_x = 0 \Rightarrow w_y v_y = -v_x w_x$$

which using (A.4.26) gives:

$$w_y v_y = -v_x w_x \Rightarrow w^t p \geq w^t q$$

III) As expected by the other side of the proof, the case $v_y < 0$ can never happened. If $u_x > 0$ we have $u_x v_y < 0$, so by the first assumption:

$$\theta(u) < \theta(v_\perp) \Rightarrow \frac{u_y}{u_x} < -\frac{v_x}{v_y} \Rightarrow u_y v_y > -v_x u_x$$

But (A.4.25) is true by the second assumption so:

$$u^t p < u^t q \Rightarrow u_y v_y < -v_x u_x$$

and we have a contradiction.

By definition (A.4.11) of dir , the only other case is $u_x = 0$. Using (A.4.28):

$$u_x = 0 \Rightarrow u_y = -1 \Rightarrow u_y v_y > 0 \Rightarrow u_y v_y > -v_x u_x$$

But by (A.4.25):

$$u^t p < u^t q \Rightarrow u_y v_y < -v_x u_x$$

and we have a contradiction again.

□

Theorem 6.3.1. *Let $\ell = \text{line}(v, c)$. If $v^t p_1 < v^t p_2 < \dots < v^t p_n$ then:*

$$p_{i+1} \in S_1(\ell, p_i) \Leftrightarrow v_{\perp}^t p_{i+1} - c \leq 0$$

and:

$$S_1(\ell, p_{i+1}) = \begin{cases} S_1(\ell, p_i) \setminus p_{i+1} & p_{i+1} \in S_1(\ell, p_i) \\ S_1(\ell, p_i) \cup p_{i+1} & p_{i+1} \notin S_1(\ell, p_i) \end{cases}$$

Proof. For the first result, we use the premiss:

$$v^t p_{i+1} > v^t p_i \Rightarrow v^t(p_{i+1} - p_i) > 0$$

which we can deduce these two statements:

$$p_{i+1} \in S_1^B(\ell, p_i) \Leftrightarrow v_{\perp}^t p_{i+1} - c \leq 0 \quad (\text{A.4.42})$$

$$p_{i+1} \notin S_1^A(\ell, p_i) \quad (\text{A.4.43})$$

from (A.4.43) and definition (A.4.8) we get:

$$p_{i+1} \in S_1(\ell, p_i) \Leftrightarrow p_{i+1} \in S_1^B(\ell, p_i). \quad (\text{A.4.44})$$

and this together with (A.4.42) gives what we wanted to proof:

$$p_{i+1} \in S_1(\ell, p_i) \Leftrightarrow v_{\perp}^t p_{i+1} - c \leq 0. \quad (\text{A.4.45})$$

For the second result notice that:

$$\{p \in S \mid v^t(p - p_{i+1}) \leq 0\} = \{p \in S \mid v^t p \leq v^t p_{i+1}\} = \{p \in S \mid v^t p \leq p_i\} \cup p_{i+1}$$

so, using this and the distribution rule: :

$$\begin{aligned} S_1^A(\ell, p_{i+1}) &= \{p \in S \mid v^t(p - p_{i+1}) \leq 0\} \cap \{p \in S \mid v_{\perp}^t p - c > 0\} \\ &= (\{p \in S \mid v^t p \leq p_i\} \cup p_{i+1}) \cap \{p \in S \mid v_{\perp}^t p - c > 0\} \\ &= S_1^A(\ell, p_i) \cup (p_{i+1} \cap \{p \in S \mid v_{\perp}^t p - c > 0\}) \end{aligned}$$

which means, by (A.4.45):

$$S_1^A(\ell, p_{i+1}) = \begin{cases} S_1^A(\ell, p_i) & p_{i+1} \in S_1(\ell, p_i) \\ S_1^A(\ell, p_i) \cup p_{i+1} & p_{i+1} \notin S_1(\ell, p_i) \end{cases} \quad (\text{A.4.46})$$

For S_1^B we do the same with slight changes :

$$\begin{aligned} S_1^B(\ell, p_{i+1}) &= \{p \in S \mid v^t(p - p_{i+1}) > 0\} \cap \{p \in S \mid v_{\perp}^t p - c \leq 0\} \\ &= (\{p \in S \mid v^t p > p_i\} \setminus p_{i+1}) \cap \{p \in S \mid v_{\perp}^t p - c \leq 0\} \\ &= S_1^B(\ell, p_i) \setminus p_{i+1} \end{aligned}$$

last equation together with (A.4.6) and definition (??) gives:

$$\begin{aligned}
 S_1(\ell, p_{i+1}) = S_1^A(\ell, p_{i+1}) \cup S_1^B(\ell, p_{i+1}) &= \begin{cases} S_1^A(\ell, p_i) \cup S_1^B(\ell, p_i) \setminus p_{i+1} & p_{i+1} \in S_1(\ell, p_i) \\ S_1^A(\ell, p_i) \cup S_1^B(\ell, p_i) \cup p_{i+1} & p_{i+1} \notin S_1(\ell, p_i) \end{cases} \\
 &= \begin{cases} S_1 \setminus p_{i+1} & p_{i+1} \in S_1(\ell, p_i) \\ S_1 \cup p_{i+1} & p_{i+1} \notin S_1(\ell, p_i) \end{cases}
 \end{aligned}$$

□

Bibliography

- [1] D.T. Lee and Y.T Ching. The power of geometric duality revisited, *Inform. Process. Lett.*, 21 (1985), pp. 117-122,
- [2] J. Matousek, D. M. Mount, and N. S. Netanyahu (1993), Efficient randomized algorithms for the repeated median line estimator, in *Proceedings of the Fourth Annual ACM-SIAM Symposium on Discrete Algorithms*, Austin, Texas, January 25-27, 1993, 74-82.
- [3] P. Indyk, R. Motwani, P. Raghavan, and S. Vemplala, Locality-preserving hashing in multi-dimensional spaces. *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 618-625.
- [4] J. Kleinberg, Two algorithms for nearest-neighbour search in high dimension, *Proc. 29th Annu. ACM Sympos. Theory Comput.*, 1997, pp. 599-608.
- [5] P. K. Agarwal and C. M. Procopiuc. Covering points by strips in the plane, 1998. Unpublished manuscript.
- [6] A. Glozman, K. Kedem, and G. Shpitalnik, On some geometric selection and optimization problems via sorted matrices, *Proc 4th Workshop Algorithms Data Struct, Lecture Notes Comput. Sci*, Vol. 995, Springer-Verlag, 1995, pp. 26-37.
- [7] P. Agarwal and M. Sharir, “Planer geometric location problems”, *Algorithmica*, 11(1994), pp. 185-195.
- [8] N. Megiddo, “Applying parallel computation algorithms in the design of serial algorithms”, *J. ACM*, 30(1983), pp. 852-865

- [9] M.J. Katz, "Improved algorithms in geometric optimization via expanders", *Proc. 3rd Israel Symposium on Theory of Computing and Systems*, 1995, pp. 78-87.
- [10] M.J. Katz and M. Sharir, "An expander-based approach to geometric optimization", *Proc. 9th ACM Symp. on Computational Geometry*, 1993, pp. 198-207
- [11] J.W. Jaromczyk, M. Kowaluk, "The two-line center problem from a polar view: A new algorithm and data structure", *Proc. 4th Workshop Algorithms Data Struct.*, Lecture Notes in Computer Science, (955), 13-25, 1995
- [12] N. Megiddo and A. Tamir, On the complexity of locating linear facilities in the plane, *Oper. Res. Lett.*, 1(1982), 194-197
- [13] R. Hassin and N. Megiddo, Approximation algorithms for hitting objects by straight lines, *Discrete Appl. Math.*, 30(1991), 29-42.
- [14] M.E. Houle and G.T. Toussaint, Computing the width of a set, *IEEE Trans. Pattern Anal. Mach. Intell.*, PAMI-10 (1988), 761-765
- [15] D. T. Lee and Y. F. Wu, Geometric complexity of some location problems, *Algorithmica*, 1(1986), 193-211.
- [16] C.A. Duncan, M.T. Goodrich, and E.A. Ramos, Efficient approximation and optimization algorithms for computational metrology, *Proc. 8th ACM-SIAM Sympos. Discrete Algorithms*, 1997, pp. 121-139.
- [17] P.K. Agarwal and M. Sharir, Efficient randomized algorithms for some geometric optimization problems, *Discrete Comput. Geom.*, 16 (1996), 317-337.
- [18] M. J. Katz and M. Sharir, Optimal slope selection via expanders, *Inform. Process. Lett.*, 47 (1993), 115-122.

- [19] V. Chvatal, A greedy heuristic for the set-covering problem, *Math. Oper. Res.*, 4(1979), 233-235.
- [20] H. Brönnimann and M.T. Goodrich, Almost optimal set covers in finite VC-dimension, *Discrete Comput. Geom.*, 14(1995), 263-279.
- [21] P. K. Agarwal and C. M. Procopiuc. Covering points by strips in the plane, 1998. Unpublished manuscript.
- [22] P.K Agarwal and J. Erickson, Geometric range searching and its relatives, In B.Chazelle, J.E Goodman, and R. Pollack, editors, *Advances in Discrete and Computational Geometry, volume 223 of Contemporary Mathematics*, pages 1-56, 1999
- [23] P.K Agarwal and C.M Procopiuc, Approximation algorithms for projective clustering, In *Proc. 11th ACM-SIAM Sympos. Discrete Algorithms*, pages 538-547, 2000
- [24] S. Har-Peled, Clustering motion, In *Proc. 42nd Annu. IEEE Sympos. Found. Comput. Sci. 2001*
- [25] Pankaj K. Agarwal, Cecilia Magdalena Procopiuc, Kasturi R. Varadarajan, A $(1 + \epsilon)$ -approximation algorithm for 2-line-center, *Comput. Geom.* 26(2): 119-128 (2003)
- [26] Matousek, J. 1999. On Approximate Geometric k -Clustering. Manuscript, Department of Applied Mathematics, Charles University, Prague, Czech Republic.
- [27] S. Hasegawa, H. Imai, M. Inaba, N. Katoh, and J. Nakano. Efficient algorithms for variance-based k -clustering. In *Proc. First Pacific Conf. Computer Graphics Appl., Seoul, Korea, vol. 1*, pages 75-89. World Scientific Publishing Co., 1993.
- [28] M. Inaba, N. Katoh, and H.Imai. Applications of weighted Voronoi diagrams and randomization to variance-based k -clustering. In *Proc. 10th Annu. ACM Sympos. Comput. Geom.*, pages 332-339, 1994

- [29] A. Hyvärinen and E. Oja, and J. Karhunen, Independent Component Analysis, Wiley-Interscience (2001)
- [30] R. Cappelli, D. Maio, and D. Maltoni , Multispace KL for Pattern Representation and Classification, *IEEE Transactions on Pattern Analysis and Machine Intelligence Journal*, pp. 977-996.
- [31] M. Charikar and S. Guha. Improved combinatorial algorithms for facility location and k -median problems. In *Proceedings of the 40th Annual IEEE Symposium on Foundations of Computer Science*, pages 378-388, October 1999
- [32] M. Charikar, S. Guha., É. Tardos, and D.B. Shmoys. A Constant-factor approximation algorithm for the k -median problem. In *Proceedings of the 31st Annual ACM Symposium on Theory of Computing*, pages 1-10, May 1999.
- [33] C.D Manning and H. Schütze. *Foundation of Statistical Natural Language Processing*. MIT Press, Cambridge, 1999.
- [34] J. H. Lin and J.S Vitter. Approximation algorithms for geometric meidan problems. *Inform. Proc. Lett.*, 44:245-249, 1992.
- [35] M.R. Korupolu, C.G. Plaxton, and R. Rajaraman. Analysis of a local search heuristic for facility location problems. In *Proceedings of the 9th Annual ACM-SIAM Symposium on Discrete Algorithms*, pages 1-10, 1998.
- [36] D.B Shmoys and E. Tardos. An approximation Algorithm for the Generalized Assignment Problem. *Math. Programming*, 62:461-474, 1993.
- [37] S. Arora, P. Raghavan and S.Rao. Approximation schemes for Euclidean k -medians and related problems. *Proceedings of the 30th Annual ACM Symposium on Theory of Computing*, pp. 106-113, 1998
- [38] Gene H. Golub and Charles F. Van Loan, *Matrix Computation*, Johns Hopkins Univ Pr; 3rd edition, 1996
- [39] Herbert Edelsbrunner, Algorithms in Combinatorial Geometry, *Monographs on Tehoretical Computer Science*,1987
- [40] T. Breuel, Finding lines under bounded error, *In Pattern Recognition*, vol. 29, pp. 167-178, 1996
- [41] Jain, Ranesh, Machine Vision, McGraw-Hill, New York.

- [42] V. Arya, N. Garg, R. Khandekar, A. Meyerson, K. Munagala, and V. Pandit. Local search heuristics for k -median and facility location problems. In *Proceedings of 33rd ACM Symposium on Theory of Computing*, pages 21-29, 2001.