Chapter 21

The Perceptron algorithm and variants

By Sariel Har-Peled, March 28, 2023[1] Version: 0.2

21.1. The perceptron algorithm

Assume, that we are given examples, say a database of cars, and you would like to determine which cars are sport cars, and which are regular cars. Each car record, can be interpreted as a point in high dimensions. For example, a sport car with 4 doors, manufactured in 1997, by Quaky (with manufacturer ID 6) will be represented by the point $(4, 1997, 6)$, marked as a sport car. A tractor made by General Mess (manufacturer ID 3) in 1998, would be stored as $(0, 1997, 3)$ and would be labeled as not a sport car.

Naturally, in a real database there might be hundreds of attributes in each record, for engine size, weight, price, maximum speed, cruising speed, etc, etc, etc.

We would like to automate this ***classification*** process, so that tagging the records whether they correspond to race cars be done automatically without a specialist being involved. We would like to have a learning algorithm, such that given several classified examples, develop its own conjecture about what is the rule of the classification, and we can use it for classifying new data.

That is, there are two stages for ***learning***: ***training*** and ***classifying***. More formally, we are trying to learn a function

$$f : \mathbb{R}^d \to \{-1, 1\}.$$

The challenge is, of course, that f might have infinite complexity – informally, think about a label assigned to items where the label is completely random – there is nothing to learn except knowing the label for all possible items.

This situation is extremely rare is the real world, and we would limit ourselves to a set of functions that can be easily described. For example, consider a set of red and blue points that are linearly separable, as demonstrated in Figure 21.1.1. That is, we are trying to learn a line ℓ that separates the red points from the blue points.

The natural question is now, given the red and blue points, how to compute the line ℓ? Well, a line or more generally a plane (or even a hyperplane) is the zero set of a linear function, that has the form

$$\forall x \in \mathbb{R}^d \qquad f(x) = \langle a, x \rangle + b,$$

where $a = (a_1, \ldots, a_d), b = (b_1, \ldots, b_d) \in \mathbb{R}^2$, and $\langle a, x \rangle = \sum_i a_i x_i$ is the ***dot product*** of a and x. The classification itself, would be done by computing the sign of $f(x)$; that is $sign(f(x))$. Specifically, if $sign(f(x))$ is negative, it outside the class, if it is positive it is inside.

A set of training examples is a set of pairs

$$S = \{(x_1, y_1), \ldots, (x_n, y_n)\},$$

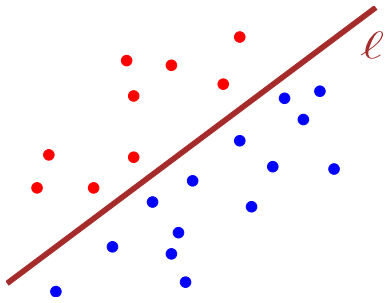


Figure 21.1.1: Linear separable red and blue point sets.

where $x_i \in \mathbb{R}^d$ and $y_i \in \{-1,1\}$, for $i = 1, \ldots, n$.

A ***linear classifier*** h is a pair (w, b) where $w \in \mathbb{R}^d$ and $b \in \mathbb{R}$. The classification of $x \in \mathbb{R}^d$ is $sign(\langle w, x\rangle + b)$. For a ***labeled*** example (x, y), h classifies (x, y) correctly if $\text{sign}(\langle w, x\rangle + b) = y$.

Assume that the underlying label we are trying to learn has a linear classifier (this is a problematic assumption – more on this later), and you are given "enough" examples (i.e., n). How to compute the linear classifier for these examples?

One natural option is to use linear programming. Indeed, we are looking for $(\mathbb{w}, b)$, such that for an $(\mathbb{x}_i, y_i)$ we have $\text{sign}(\langle \mathbb{w}, \mathbb{x}_i\rangle + b) = y_i$, which is

$$\begin{aligned} & \langle \mathbb{w}, \mathbb{x}_i\rangle + b \geq 0 && \text{if } y_i = 1, \\ \text{and} \qquad & \langle \mathbb{w}, \mathbb{x}_i\rangle + b \leq 0 && \text{if } y_i = -1. \end{aligned}$$

Or equivalently, let $\mathbb{x}_i = \left(x_i^1, \ldots, x_i^d\right) \in \mathbb{R}^d$, for $i = 1, \ldots, m$, and let $\mathbb{w} = \left(w^1, \ldots, w^d\right)$, then we get the linear constraint

$$\begin{aligned} & \sum_{k=1}^{d} w^k x_i^k + b \geq 0 && \text{if } y_i = 1, \\ \text{and} \qquad & \sum_{k=1}^{d} w^k x_i^k + b \leq 0 && \text{if } y_i = -1. \end{aligned}$$

Thus, we get a set of linear constraints, one for each training example, and we need to solve the resulting linear program.

The main stumbling block is that linear programming is very sensitive to noise. Namely, if we have points that are misclassified, we would not find a solution, because no solution satisfying all of the constraints exists. Instead, we are going to use an iterative algorithm that converges to the optimal solution if it exists, see Figure 21.1.2.

Why does the **perceptron** algorithm converges to the right solution? Well, assume that we made a mistake on a sample $(\mathbb{x}, y)$ and $y = 1$. Then, $\langle \mathbb{w}_k, \mathbb{x}\rangle < 0$, and

$$\langle \mathbb{w}_{k+1}, \mathbb{x}\rangle = \langle \mathbb{w}_k + y * \mathbb{x}, x\rangle = \langle \mathbb{w}_k, \mathbb{x}\rangle + y \langle \mathbb{x}, \mathbb{x}\rangle = \langle \mathbb{w}_k, \mathbb{x}\rangle + y \|\mathbb{x}\| > \langle \mathbb{w}_k, \mathbb{x}\rangle .$$

Namely, we are "walking" in the right direction, in the sense that the new value assigned to $\mathbb{x}$ by $\mathbb{w}_{k+1}$ is larger ("more positive") than the old value assigned to $\mathbb{x}$ by $\mathbb{w}_k$.

Theorem 21.1.1. *Let S be a training set of examples, and let $R = \max_{(x,y)\in S} \left\|x\right\|$. Suppose that there exists a vector w_{opt} such that $\left\|w_{opt}\right\| = 1$, and a number $\gamma > 0$, such that*

$$y \langle w_{opt},\ x\rangle \geq \gamma \qquad \forall (x, y) \in S.$$

```
Algorithm perceptron(S: a set of l examples)
  w_0 ← 0, k ← 0
  R = max_{(x,y)∈S} ||x||.
  repeat
    for (x, y) ∈ S do
      if sign(⟨w_k, x⟩) ≠ y then
        w_{k+1} ← w_k + y * x
        k ← k + 1
  until no mistakes are made in the classification
  return w_k and k
```

Figure 21.1.2: The **perceptron** algorithm.

Then, the number of mistakes made by the online **perceptron** *algorithm on S is at most*

$$\left(\frac{R}{\gamma}\right)^2.$$

Proof: Intuitively, the **perceptron** algorithm weight vector converges to w_{opt}, To see that, let us define the distance between w_{opt} and the weight vector in the kth update:

$$\alpha_k = \left\| w_k - \frac{R^2}{\gamma} w_{opt} \right\|^2.$$

We next quantify the change between α_k and α_{k+1} (the example being misclassified is (x, y)):

$$\begin{aligned}
\alpha_{k+1} &= \left\| w_{k+1} - \frac{R^2}{\gamma} w_{opt} \right\|^2 \\
&= \left\| w_k + y\mathtt{x} - \frac{R^2}{\gamma} w_{opt} \right\|^2 \\
&= \left\| \left(w_k - \frac{R^2}{\gamma} w_{opt} \right) + y\mathtt{x} \right\|^2 \\
&= \left\langle \left(w_k - \frac{R^2}{\gamma} w_{opt} \right) + y\mathtt{x},\ \left(w_k - \frac{R^2}{\gamma} w_{opt} \right) + y\mathtt{x} \right\rangle.
\end{aligned}$$

Expanding this we get:

$$\begin{aligned}
\alpha_{k+1} &= \left\langle \left(w_k - \frac{R^2}{\gamma} w_{opt} \right), \left(w_k - \frac{R^2}{\gamma} w_{opt} \right) \right\rangle + 2y \left\langle \left(w_k - \frac{R^2}{\gamma} w_{opt} \right), \mathtt{x} \right\rangle + \langle \mathtt{x}, \mathtt{x} \rangle \\
&= \alpha_k + 2y \left\langle \left(w_k - \frac{R^2}{\gamma} w_{opt} \right), x \right\rangle + \left\| x \right\|^2.
\end{aligned}$$

As $(\mathtt{x}, y)$ is misclassified, it must be that $\text{sign}(\langle w_k, \mathtt{x} \rangle) \neq y$, which implies that $\text{sign}(y \langle \mathtt{w}_k, \mathtt{x} \rangle) = -1$; that is $y \langle \mathtt{w}_k, \mathtt{x} \rangle < 0$. Now, since $\left\| x \right\| \leq R$, we have

$$\begin{aligned}
\alpha_{k+1} &\leq \alpha_k + R^2 + 2y \langle w_k, \mathtt{x} \rangle - 2y \left\langle \frac{R^2}{\gamma} w_{opt}, \mathtt{x} \right\rangle \\
&\leq \alpha_k + R^2 + \qquad\qquad -2\frac{R^2}{\gamma} y \langle w_{opt}, x \rangle.
\end{aligned}$$

Next, since $y \langle w_{opt} \ , \ x \rangle \geq \gamma$ for $\forall (x, y) \in S$, we have that

$$\alpha_{k+1} \leq \alpha_k + R^2 - 2\frac{R^2}{\gamma}\gamma \leq \alpha_k + R^2 - 2R^2 \leq \alpha_k - R^2.$$

We have: $\alpha_{k+1} \leq \alpha_k - R^2$, and

$$\alpha_0 = \left\| 0 - \frac{R^2}{\gamma} w_{opt} \right\|^2 = \frac{R^4}{\gamma^2} \left\| w_{opt} \right\|^2 = \frac{R^4}{\gamma^2} \quad .$$

Finally, observe that $\alpha_i \geq 0$ for all i. At each misclassification, α_i shrinks by R^2. It starts at R^4/γ^2. As such, he algorithm can perform at most

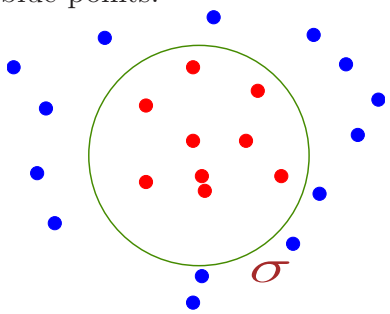
$$\frac{R^4/\gamma^2}{R^2} = \frac{R^2}{\gamma^2},$$

iterations before α_i becomes negative, which is of course impossible. ■

It is important to observe that any linear program can be written as the problem of separating red points from blue points. As such, the **perceptron** algorithm can be used to solve linear programs.

21.2. Learning A Circle

Given a set of red points, and blue points in the plane, we want to learn a circle that contains all the red points, and does not contain the blue points.



How to compute the circle σ ?

It turns out we need a simple but very clever trick. For every point $(x, y) \in P$ map it to the point $(x, \ y, \ x^2 + y^2)$. Let $z(P) = \left\{ (x, \ y, \ x^2 + y^2) \ \middle| \ (x, y) \in P \right\}$ be the resulting point set.

Theorem 21.2.1. *Two sets of points R and B are separable by a circle in two dimensions, if and only if $z(R)$ and $z(B)$ are separable by a plane in three dimensions.*

Proof: Let $\sigma \equiv (x - a)^2 + (y - b)^2 = r^2$ be the circle containing all the points of R and having all the points of B outside. Clearly, $(x - a)^2 + (y - b)^2 \leq r^2$ for all the points of R. Equivalently

$$-2ax - 2by + \left(x^2 + y^2 \right) \leq r^2 - a^2 - b^2.$$

Setting $z = x^2 + y^2$ we get that

$$h \equiv -2ax - 2by + z - r^2 + a^2 + b^2 \leq 0.$$

Namely, $p \in \sigma$ if and only if $h(z(p)) \leq 0$. We just proved that if the point set is separable by a circle, then the lifted point set $z(R)$ and $z(B)$ are separable by a plane.

As for the other direction, assume that $z(R)$ and $z(B)$ are separable in $3d$ and let

$$h \equiv ax + by + cz + d = 0$$

be the separating plane, such that all the point of $z(R)$ evaluate to a negative number by h. Namely, for $(x, y, x^2 + y^2) \in z(R)$ we have $ax + by + c(x^2 + y^2) + d \leq 0$

and similarly, for $(x, y, x^2 + y^2) \in B$ we have $ax + by + c(x^2 + y^2) + d \geq 0$.

Let $U(h) = \Big\{(x, y) \;\Big|\; h((x, y, x^2 + y^2)) \leq 0\Big\}$. Clearly, if $U(h)$ is a circle, then this implies that $R \subset U(h)$ and $B \cap U(h) = \emptyset$, as required.

So, $U(h)$ are all the points in the plane, such that

$$ax + by + c\Big(x^2 + y^2\Big) \leq -d.$$

Equivalently

$$\left(x^2 + \frac{a}{c}x\right) + \left(y^2 + \frac{b}{c}y\right) \leq -\frac{d}{c}$$

$$\left(x + \frac{a}{2c}\right)^2 + \left(y + \frac{b}{2c}\right)^2 \leq \frac{a^2 + b^2}{4c^2} - \frac{d}{c}$$

but this defines the interior of a circle in the plane, as claimed. ■

This example show that linear separability is a powerful technique that can be used to learn complicated concepts that are considerably more complicated than just hyperplane separation. This lifting technique showed above is the ***kernel technique*** or ***linearization***.

21.3. Active learning, sparsity and large margin

Let P be a point set of n points in $\mathbb{R}^d$. Every point has a label/color (say *black* or *white*), but we do not know the labels. In particular, let B and W be the set of black and white points in P. Furthermore, let $\nabla = \mathrm{diam}(\mathsf{P})$, and assume that there exist two parallel hyperplanes h, h' in distance γ from each other, such that the slab between h and h' does not contain an point of P, and the points of B are on one side of this slab, and the points of W are on the other side. The quantity γ is the ***margin*** of P.

A somewhat more convenient way to handle such slabs, is to consider two points b and w in $\mathbb{R}^d$. Let $\mathrm{slab}(\mathsf{b}, \mathsf{w})$ be the region of points in $\mathbb{R}^d$, such that their projection onto the line spanned by b and w is contained in the open segment bw. We use $(1 - \varepsilon)\mathrm{slab}(\mathsf{b}, \mathsf{w})$ to denote the slab formed from $\mathrm{slab}(\mathsf{b}, \mathsf{w})$ by shrinking it by a factor of $(1 - \varepsilon)$ around its middle hyperplane. Formally, it is defined as $(1 - \varepsilon)\mathrm{slab}(\mathsf{b}, \mathsf{w}) = \mathrm{slab}(\mathsf{b}', \mathsf{w}')$, where $\mathsf{b}' = (1 - \varepsilon/2)\mathsf{b} + (\varepsilon/2)\mathsf{w}$ and $\mathsf{w}' = (\varepsilon/2)\mathsf{b} + (1 - \varepsilon/2)\mathsf{w}$.

In the following, we assume have an access to a ***labeling oracle*** that can return the label of a specific query point. Similarly, we assume access to a ***counterexample oracle***, such that given a slab that does not contain any points of P in its interior, and supposedly separates the points of P into B and W, it returns a point that is mislabeled by this classifier (i.e., slab) if such a point exists.

Conceptually, asking queries from the oracles is quite expensive, and the algorithm tries to minimize the number of such queries.

The algorithm. Assume there are two points $\mathsf{b}_1 \in \mathsf{B}$ and $\mathsf{w}_1 \in \mathsf{W}$. For $i > 0$, in the ith iteration, the algorithm considers the slab $S_i = (1-\varepsilon)\mathrm{slab}(\mathsf{b}_i, \mathsf{w}_i)$. There are two possibilities:

(A) If the slab S_i contains no points of P, then the algorithm uses the counterexample oracle to check if it is done – that is, all the points are classified correctly. Otherwise, a badly classified point p_i was returned.

(B) The S_i contains some points of P, and let p_i be the closest point to the middle hyperplane of the slab S_i. The algorithm uses the labeling oracle to get the label of p_i.

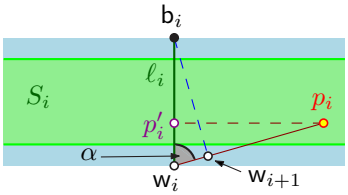
Assume that the label of p_i is white. Then, the algorithm set w_{i+1} be the projection of b_i to $\mathsf{w}_i p_i$, and $\mathsf{b}_{i+1} = \mathsf{b}_i$ (the case that p_i is black is handled in a symmetric fashion).

Lemma 21.3.1 ([HRZ07]). *Let P be a set of points in $\mathbb{R}^d$, with diameter ∇. Assume there is an unknown partition of P into two point sets W and B, of white and black points, respectively, and this partition has margin γ. Furthermore, we are given an access to a labeling and counterexample oracles. Finally, there are two given initial points $\mathsf{b}_1 \in \mathsf{B}$ and $\mathsf{w}_1 \in \mathsf{W}$.*

Then, for any $\varepsilon > 0$, one can compute using an iterative algorithm, in $I = O\Big((\nabla/\gamma)^2/\varepsilon^2\Big)$ iterations and in $O(Idn)$ time, a slab of width $\geq (1-\varepsilon)\gamma$ that separates B from W. This algorithm performs I calls to the labeling/counterexample oracles.

Proof: Our purpose is to analyze the number of iterations of this algorithm till it terminates. So, let $\ell_i = \|\mathsf{b}_i\mathsf{w}_i\|$. Clearly, $\nabla \geq \ell_0 \geq \ell_1 \geq \cdots \geq \gamma$, the last step follows as $\mathsf{b}_i \in \mathcal{CH}(\mathsf{B})$ and $\mathsf{w}_i \in \mathcal{CH}(\mathsf{W})$, and the distance $\mathsf{d}(\mathcal{CH}(\mathsf{B}), \mathcal{CH}(\mathsf{W})) \geq \gamma$, where $\mathsf{d}(X,Y) = \min_{x\in X}\min_{y\in Y}\|xy\|$.

Let p_i' be the projection of p_i to the line spanned by $\mathsf{w}_i\mathsf{b}_i$. Observe that if $p_i \in S_i$ then $\|p_i'\mathsf{w}_i\| \geq \varepsilon\ell_i/2$. Formally, the points w_i breaks the line spanned by w_i and b_i into two parts, and b_i and p_i' are on the same side, and p_i' is distance at least $\ell_i/2$ away from w_i along this ray. Observe that if case (B) above happened, then p_i is not inside S_i, and this distance is significantly larger.



Setting $\alpha = \angle p_i\mathsf{w}_i\mathsf{b}_i$, we have $\cos\alpha = \dfrac{\|p_i'\mathsf{w}_i\|}{\|\mathsf{w}_i p_i\|} \geq \dfrac{\varepsilon\ell_i/2}{\nabla}$. As such, we have

$$\ell_{i+1} = \ell_i \sin\alpha \leq \ell_i\sqrt{1-\left(\frac{\varepsilon\ell_i}{2\nabla}\right)^2} \leq \left(1-\left(\frac{\varepsilon\ell_i}{4\nabla}\right)^2\right)\ell_i. \tag{21.3.1}$$

We have that $\ell_{i+k} \leq \ell_i/2$, for $k = \left\lceil 64\nabla^2/(\varepsilon\ell_i)^2 \right\rceil$. Indeed, if $\ell_{i+k} > \ell_i/2$, then

$$\ell_{i+k} \leq \ell_i \prod_{j=0}^{k-1}\left(1-\left(\frac{\varepsilon\ell_{i+j}}{4\nabla}\right)^2\right) \leq \ell_i \prod_{j=0}^{k-1}\left(1-\left(\frac{\varepsilon\ell_{i+k}}{4\nabla}\right)^2\right) \leq \ell_i \exp\left(-k\left(\frac{\varepsilon\ell_{i+k}}{4\nabla}\right)^2\right) \tag{21.3.2}$$

$$\leq \ell_i \exp\left(-k\left(\frac{\varepsilon\ell_i}{8\nabla}\right)^2\right) \leq \ell_i \exp\left(-k\left(\frac{\varepsilon\ell_i}{8\nabla}\right)^2\right) \leq \frac{\ell_i}{e}, \tag{21.3.3}$$

which is a contradiction.

In particular, the jth epoch of the algorithm are the iterations where $\ell_i \in \left[\nabla/2^{j-1}, \nabla/2^j\right]$. Namely, during an epoch the width of the current slab shrinks by a factor of two. By Eq. (21.3.3), the jth epoch lasts $n_j = O\Big((2^j/\varepsilon)^2\Big)$ iterations. As such, the total number of iterations $\sum_j n_j$ is dominated by the last

epoch, that starts (roughly) when $\ell_i \leq 2\gamma$, and end when it hits γ. This last epoch takes $O\left(\nabla^2/(\varepsilon\gamma)^2\right)$ iterations, which also bounds the total number of iterations. ■

Remark. (A) if the data is already labeled, then the algorithm of Lemma 21.3.1 can be implemented directly resulting in the same running time as stated. This algorithm approximates the maximum margin classifier to the data. Specifically, the above algorithm $(1+\varepsilon)$-approximates the distance $\mathsf{d}(\mathsf{B},\mathsf{W})$, and it can be interpreted as an approximation algorithm for the associated quadratic program.

(B) One can implement the counterexample oracle, by sampling enough labels, and using the labeling oracle. This is introduces a certain level of error. See [HRZ07] for details.

21.3.1. Computing the approximate distance to the convex hull

The following is well known, and is included for the sake of completeness, see [HKMR15]. It also follows readily from the Preceptron algorithm (see Remark **??** below).

Lemma 21.3.2. *Let $\mathsf{P} \subseteq \mathbb{R}^d$ be a point set, $\varepsilon > 0$ be a parameter, and let $\mathsf{q} \in \mathbb{R}^d$ be a given query point. Then, one can compute, in $O(|\mathsf{P}|\, d/\varepsilon^2)$ time, a point $q \in \mathcal{CH}(\mathsf{P})$, such that $\|\mathsf{q}q\| \leq \mathsf{d}(\mathsf{q},\mathcal{CH}(\mathsf{P})) + \varepsilon\nabla$, where $\nabla = \operatorname{diam}(\mathsf{P})$. Furthermore, q is a convex combination of $O(1/\varepsilon^2)$ points of P.*

Proof: The algorithm is iterative, computing a sequence of points $q_0, \ldots, q_i$ inside $\mathcal{CH}(\mathsf{P})$ that approach q. Initially, $p_0 = q_0$ is the closest point of P to q. In the ith iteration, the algorithm computes the vector $v_i = \mathsf{q} - q_{i-1}$, and the point $p_i \in \mathsf{P}$ that is extremal in the direction of v_i. Now, the algorithm sets q_i to be the closest point to q on the segment $s_i = q_{i-1}p_i$, and continues to the next iteration, for $M = O(1/\varepsilon^2)$ iterations. The algorithm returns the point q_M as the desired answer.

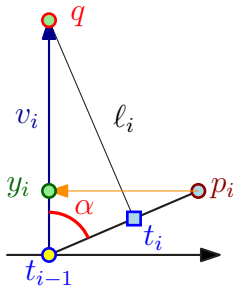


Figure 21.3.1

By induction, the point $q_i \in \mathcal{CH}(\{p_0, \ldots, p_i\})$. Furthermore, observe that the distance of the points $q_0, q_1, \ldots$ from q is monotonically decreasing. In particular, for all $i > 0$, q_i must fall in the middle of the segment s_i, as otherwise, p_i would be closer to q than p_0, a contradiction to the definition of p_0.

Project the point p_i to the segment $q_{i-1}\mathsf{q}$, and let s_i be the projected point. Observe that $\|\mathsf{q}\mathsf{s}_i\|$ is a lower bound on $\mathsf{d}(\mathsf{q},\mathcal{CH}(\mathsf{P}))$. Therefore, if $\|\mathsf{s}_i q_{i-1}\| \leq \varepsilon\nabla$ then we are done, as $\|\mathsf{q}q_{i-1}\| \leq \|q_{i-1}\mathsf{s}_i\| + \|\mathsf{s}_i\mathsf{q}\| \leq \varepsilon\nabla + \mathsf{d}(\mathsf{q},\mathcal{CH}(\mathsf{P}))$. (In particular, one can use this as alternative stopping condition for the algorithm, instead of counting iterations.)

So, let α be the angle $\angle p_i q_{i-1}\mathsf{q}$. Observe that as $q_{i-1}p_i \subseteq \mathcal{CH}(\mathsf{P})$, it follows that $\|q_{i-1}p_i\| \leq \operatorname{diam}(\mathsf{P}) = \nabla$. Furthermore, $\cos\alpha = \dfrac{\|\mathsf{s}_i q_{i-1}\|}{\|q_{i-1}p_i\|} > \dfrac{\varepsilon\nabla}{\nabla} = \varepsilon$, since $\|\mathsf{s}_i q_{i-1}\| > \varepsilon\nabla$. Hence, $\sin\alpha = \sqrt{1-\cos^2\alpha} \leq \sqrt{1-\varepsilon^2} \leq 1 - \varepsilon^2/2$. Let $\ell_{i-1} = \|\mathsf{q}q_{i-1}\|$. We have that

$$\ell_i = \|\mathsf{q}q_i\| = \|\mathsf{q}q_{i-1}\| \sin\alpha \leq (1-\varepsilon^2/2)\ell_{i-1}.$$

Analyzing the number of iterations required by the algorithm is somewhat tedious. If $\ell_0 = \|\mathsf{q}q_0\| \geq (4/\varepsilon^2)\nabla$ then the algorithm would be done in one iteration as otherwise $\ell_1 \leq \ell_0 - 2\nabla$, which is impossible. In particular, after $4/\varepsilon^2$ iterations the distance ℓ_i shrinks by a factor of two, and as such, after $O((1/\varepsilon^2)\log(1/\varepsilon))$ iterations the algorithm is done.

One can do somewhat better. By the above, we can assume that $\mathsf{d}(\mathsf{q},\mathsf{P}) = O(\nabla/\varepsilon^2)$. Now, set $\varepsilon_j = 1/2^{2+j}$. By the above, after $n_0 = O((1/\varepsilon_0^2)\log(1/\varepsilon_0)) = O(1)$ iterations, $\ell_{n_0} \leq \mathsf{d}(\mathsf{q},\mathcal{CH}(\mathsf{P})) + \mathrm{diam}(\mathsf{P})/4$. For $j \geq 1$, let $n_j = 4/(\varepsilon_j)^2$, and observe that, after $\nu_j = n_j + \sum_{k=0}^{j-1} n_k$ iterations, we have that

$$\ell_{\nu_j} \leq \Big(\mathsf{d}(\mathsf{q},\mathcal{CH}(\mathsf{P})) + \varepsilon_{j-1}\nabla\Big)/2 \leq \mathsf{d}(\mathsf{q},\mathcal{CH}(\mathsf{P})) + \varepsilon_j\nabla.$$

In particular, stopping as soon as $\varepsilon_j \leq \varepsilon$, we have the desired guarantee, and the number of iterations needed is $M = O(1) + \sum_{j=0}^{\lceil \lg 1/\varepsilon \rceil} 4/\varepsilon_j^2 = O(1/\varepsilon^2)$. ∎

Theorem 21.3.3 (Fractional Carathéodory). *Let P be a set of n points in $\mathbb{R}^d$. Given a point $q \in \mathcal{CH}(P)$ and a parameter $\varepsilon \in (0,1)$, one can compute $k = O(1/\varepsilon^2)$ points $p_1,\ldots,p_k \in P$, and convex coefficients $\alpha_1,\ldots,\alpha_k \in (0,1)$ such that $q' = \sum_i \alpha_i p_i$ is "close" to q. That is, formally we have that $\sum_i \alpha_i = 1$, and $\|qq'\| \leq \varepsilon\mathrm{diam}(P)$.*

References

[HKMR15] S. Har-Peled, N. Kumar, D. Mount, and B. Raichel. Space exploration via proximity search. *Proc. 31st Int. Annu. Sympos. Comput. Geom.* (SoCG), vol. 34. 374–389, 2015.

[HRZ07] S. Har-Peled, D. Roth, and D. Zimak. Maximum margin coresets for active and noise tolerant learning. *Proc. 21st Int. Joint Conf. Art. Intell.* (IJCAI), 836–841, 2007.