NumPy / SciPy Recipes for Data Science: Frank-Wolfe for Minimum Enclosing Balls

Technical Report · September 2020

CITATION
1

READS
860

1 author:

Christian Bauckhage
University of Bonn
526 PUBLICATIONS **9,556** CITATIONS

SEE PROFILE

NumPy / SciPy Recipes for Data Science: Frank-Wolfe for Minimum Enclosing Balls

Christian Bauckhage
Computer Science, University of Bonn, Germany
Fraunhofer IAIS, Sankt Augustin, Germany

***Abstract*—This is the first in a series of notes in which we study minimum enclosing balls (MEBs) for data mining, pattern recognition, and machine learning. Here, we focus on the problem of computing Euclidean MEBs which we approach from the point of view of constrained quadratic optimization. We discuss the corresponding primal- and dual forms of the problem and find that the latter is an instance of a problem that can be solved via the Frank-Wolfe algorithm. We discuss this idea in detail and finally present corresponding *NumPy* code.**

I. Introduction

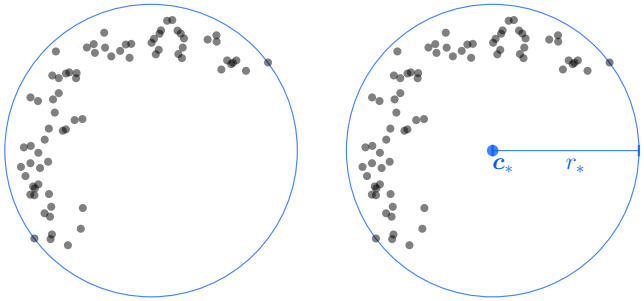
Minimum enclosing balls (MEBs) provide a very versatile data representation for a wide range of learning and analysis tasks. Use cases include (but are not limited to) accelerated training of support vector machines or other classifiers [1], [2], symbolic learning [3], [4], identification of landmarks or prototypes [5], [6], analysis of data streams [7], and anomaly detection [8]. MEBs are also at the heart of one class support vector machines [9], support vector clustering [10], or support vector data description [11].

While there exists a whole spectrum of algorithms for estimating the MEB of a given data set, a particularly general approach is to formalize the problem in terms of constrained quadratic optimization and we, too, will adhere to this strategy. In later notes, this will allow us to invoke the kernel trick so as to compute MEBs in high dimensional Hilbert spaces which yields the versatility alluded to above. For now, however, we focus on the problem of computing Euclidean MEBs as shown in Fig. 1.

While the primal problem of estimating an MEB is rather unwieldy, we will see that the corresponding dual problem can be solved rather easily. Indeed, the dual MEB problem is of a form that allows for using simple solvers such as the Frank-Wolfe algorithm [12]. We will discuss this in detail and finally show how to turn all our theoretical considerations into efficient *NumPy* code.

As always, we first summarize the underlying mathematical ideas (section II). Note that this review will be more extensive than those in most our other *NumPy* / *SciPy* recipes for data science. We also assume that readers know about the notion of Lagrange duality and the Karush-Kuhn-Tucker conditions. Those not quite so familiar with this material are encouraged to consult our introductory lecture notes in [13]–[16].

Again as always, we then put our theoretical considerations to work and discuss simple *NumPy* code for MEB computation (section III).



(a) data points and MEB (b) center point and radius

Fig. 1: Data points and their minimum enclosing ball (which in 2D is a circle). Throughout, we write $\boldsymbol{c}_*$ and r_* to denote center and radius of the MEB of a particular data set.

Readers who would like to try out our code snippets should have some experience with *NumPy* and *SciPy* [17] and import the following modules / functions

```
import numpy as np
import matplotlib.pyplot as plt
from matplotlib.patches import Circle
```

II. Theory

In this section, we first look into the *primal problem* of MEB computation, then derive the corresponding *dual problem*, and finally discuss how to apply Frank-Wolfe optimization to solve the latter almost effortlessly.

A. The Primal MEB Problem

Given a data set $\mathcal{X} = \{\boldsymbol{x}_1, \ldots, \boldsymbol{x}_n\} \subset \mathbb{R}^m$, the minimum enclosing ball problem asks for the smallest Euclidean ball

$$\mathcal{B}(\boldsymbol{c}, r) = \left\{ \boldsymbol{x} \in \mathbb{R}^m \;\middle|\; \left\|\boldsymbol{x} - \boldsymbol{c}\right\|^2 \leq r^2 \right\}$$

that contains each of the points in $\mathcal{X}$.

Since Euclidean balls are uniquely defined in terms of their center $\boldsymbol{c} \in \mathbb{R}^m$ and radius $r \in \mathbb{R}$, the basic problem therefore is to determine optimal choices $\boldsymbol{c}_*$ and r_* for these parameters.

In this note, we will approach this problem from the point of view of constrained convex optimization. Indeed, noting the equivalence $\left\|\boldsymbol{x}-\boldsymbol{c}\right\|^2 \leq r^2 \Leftrightarrow \left\|\boldsymbol{x}-\boldsymbol{c}\right\|^2 - r^2 \leq 0$, we may cast

the *primal problem* of estimating $\boldsymbol{c}_*$ and r_* as an inequality constrained quadratic minimization problem, namely

$$\begin{aligned} \boldsymbol{c}_*, r_* = \underset{\boldsymbol{c}, r}{\operatorname{argmin}} \quad & r^2 \\ \text{s.t.} \quad & \left\|\boldsymbol{x}_1 - \boldsymbol{c}\right\|^2 - r^2 \leq 0 \\ & \left\|\boldsymbol{x}_2 - \boldsymbol{c}\right\|^2 - r^2 \leq 0 \\ & \quad \vdots \\ & \left\|\boldsymbol{x}_n - \boldsymbol{c}\right\|^2 - r^2 \leq 0. \end{aligned} \tag{1}$$

In plain(er) English, this means that we are simultaneously searching for an appropriate center and radius. The (squared) radius should be as small as possible but not smaller. That is, the (squared) distance between each of the given data points and the center of the ball must be less than or equal to the (squared) radius of the ball.

While there are algorithms that can solve (1) directly, a more general approach consists in working with the corresponding dual problem which derive next.

B. The Dual MEB Problem

To get toe the dual MEB problem, we first of all observe that we may write the n individual inequality constraints in (1) in terms of a single matrix-vector inequality. To this end, we let $\mathbf{0}, \mathbf{1} \in \mathbb{R}^n$ denote vectors of all zeros and ones and gather the given data points $\boldsymbol{x}_j$ in a data matrix

$$\boldsymbol{X} = \left[\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n\right] \in \mathbb{R}^{m \times n}.$$

Given these ingredients, we may convince ourselves [13] that the inequality

$$\mathbf{1}^\intercal \left[\boldsymbol{X} - \boldsymbol{c}\mathbf{1}^\intercal\right]^\intercal \left[\boldsymbol{X} - \boldsymbol{c}\mathbf{1}^\intercal\right] - r^2\, \mathbf{1}^\intercal \preceq \mathbf{0}^\intercal$$

simultaneously captures all of the constraints in (1).

Using this compact notation, we can write the Lagrangian of our optimization problem as

$$L\big(\boldsymbol{c}, r, \boldsymbol{\mu}\big) = r^2 + \boldsymbol{z}^\intercal \boldsymbol{\mu} - 2\, \boldsymbol{c}^\intercal \boldsymbol{X}\, \boldsymbol{\mu} + \boldsymbol{c}^\intercal \boldsymbol{c} \cdot \mathbf{1}^\intercal \boldsymbol{\mu} - r^2\, \mathbf{1}^\intercal \boldsymbol{\mu} \tag{2}$$

where $\boldsymbol{z} \in \mathbb{R}^n$ is a vector with entries $z_j = \boldsymbol{x}_j^\intercal \boldsymbol{x}_j$ and $\boldsymbol{\mu} \in \mathbb{R}^n$ denotes a vector whose entries μ_j are Lagrange multipliers.

Given this Lagrangian, we next evaluate the Karush-Kuhn-Tucker conditions. Considering the KKT 1 conditions (stationarity), we have

$$\mathbf{0} \overset{!}{=} \frac{\partial \mathcal{L}}{\partial \boldsymbol{c}} = -2\, \boldsymbol{X}\, \boldsymbol{\mu} + 2\, \big(\mathbf{1}^\intercal \boldsymbol{\mu}\big)\, \boldsymbol{c}$$

and

$$0 \overset{!}{=} \frac{\partial \mathcal{L}}{\partial r} = r - r\, \mathbf{1}^\intercal \boldsymbol{\mu} = r\, \big(1 - \mathbf{1}^\intercal \boldsymbol{\mu}\big).$$

After straightforward rearrangements, these equations provide us with

$$\boldsymbol{c} = \frac{\boldsymbol{X}\, \boldsymbol{\mu}}{\mathbf{1}^\intercal \boldsymbol{\mu}} \tag{3}$$

$$\mathbf{1}^\intercal \boldsymbol{\mu} = 1 \tag{4}$$

and, plugging the latter into the former, we find

X is mxn dimensional (n is #points)
mu is n-dimensional
=> center is m dimensional

$$\boldsymbol{c} = \boldsymbol{X}\, \boldsymbol{\mu}. \tag{5}$$

Moreover, plugging (4) and (5) back into the Lagrangian in (2), we obtain the Lagrangian dual, because

$$\begin{aligned} \mathcal{L}\big(\boldsymbol{c}, r, \boldsymbol{\mu}\big) &= r^2 + \boldsymbol{z}^\intercal \boldsymbol{\mu} - 2\, \boldsymbol{c}^\intercal \boldsymbol{X}\, \boldsymbol{\mu} + \boldsymbol{c}^\intercal \boldsymbol{c} \cdot \mathbf{1}^\intercal \boldsymbol{\mu} - r^2\, \mathbf{1}^\intercal \boldsymbol{\mu} \\ &= r^2 + \boldsymbol{z}^\intercal \boldsymbol{\mu} - 2\, \boldsymbol{c}^\intercal \boldsymbol{X}\, \boldsymbol{\mu} + \boldsymbol{c}^\intercal \boldsymbol{c} - r^2 \\ &= \boldsymbol{z}^\intercal \boldsymbol{\mu} - 2\, \boldsymbol{\mu}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu} + \boldsymbol{\mu}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu} \\ &= \boldsymbol{z}^\intercal \boldsymbol{\mu} - \boldsymbol{\mu}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu} \\ &= \mathcal{D}\big(\boldsymbol{\mu}\big) \end{aligned}$$

is a function that only depends on the Lagrange multipliers $\boldsymbol{\mu}$.

Now, if we further note that the KKT 3 conditions (dual feasibility) demand $\boldsymbol{\mu} \succeq \mathbf{0}$, all our (intermediate) results and considerations so far reveal that the dual of (1) amounts to

$$\begin{aligned} \boldsymbol{\mu}_* = \underset{\boldsymbol{\mu}}{\operatorname{argmax}} \quad & \boldsymbol{z}^\intercal \boldsymbol{\mu} - \boldsymbol{\mu}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu} \\ \text{s.t.} \quad & \mathbf{1}^\intercal \boldsymbol{\mu} = 1 \\ & \boldsymbol{\mu} \succeq \mathbf{0}. \end{aligned} \tag{6}$$

Finally, we point out that $\mathcal{D}(\boldsymbol{\mu}) = \boldsymbol{z}^\intercal \boldsymbol{\mu} - \boldsymbol{\mu}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu}$ is concave in $\boldsymbol{\mu}$ so that $-\mathcal{D}(\boldsymbol{\mu})$ is convex. The solution to the maximization problem in (6) can therefore also be found by considering an equivalent minimization problem, namely

$$\begin{aligned} \boldsymbol{\mu}_* = \underset{\boldsymbol{\mu}}{\operatorname{argmin}} \quad & \boldsymbol{\mu}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu} - \boldsymbol{z}^\intercal \boldsymbol{\mu} \\ \text{s.t.} \quad & \mathbf{1}^\intercal \boldsymbol{\mu} = 1 \\ & \boldsymbol{\mu} \succeq \mathbf{0}. \end{aligned} \tag{7}$$

C. From Lagrange Multipliers to MEB Parameters

If we could solve the dual problem in (7) we would obtain a vector $\boldsymbol{\mu}_*$ of optimal Lagrange multipliers. Assuming that we could determine them, they would immediately allow us to compute the center $\boldsymbol{c}_*$ of the sought after MEB because, according to (5), we have

$$\boldsymbol{c}_* = \boldsymbol{X}\, \boldsymbol{\mu}_*. \tag{8}$$

But what about the corresponding radius r_*?

To determine this MEB parameter, we consider the KKT 4 conditions (complementary slackness). Using our compact matrix-vector notation, these dictate that, at a solution, we must have

$$\left[\boldsymbol{z} - 2\, \boldsymbol{X}^\intercal \boldsymbol{c}_* + \boldsymbol{c}_*^\intercal \boldsymbol{c}_* \cdot \mathbf{1} - r_*^2 \cdot \mathbf{1}\right]^\intercal \boldsymbol{\mu}_* = 0$$

Computing the inner product and using (8) this becomes

$$\boldsymbol{z}^\intercal \boldsymbol{\mu}_* - 2\, \boldsymbol{\mu}_*^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu}_* + \boldsymbol{\mu}_*^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu}_* \cdot \mathbf{1}^\intercal \boldsymbol{\mu}_* - r_*^2 \cdot \mathbf{1}^\intercal \boldsymbol{\mu}_* = 0$$

which, when using (4), simplifies to

$$\boldsymbol{z}^\intercal \boldsymbol{\mu}_* - \boldsymbol{\mu}_*^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu}_* - r_*^2 = 0$$

and yields

$$r_* = \sqrt{\boldsymbol{z}^\intercal \boldsymbol{\mu}_* - \boldsymbol{\mu}_*^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu}_*}. \tag{9}$$

D. Solving the Dual MEB Problem

Reiterating what we said above, we emphasize that the dual MEB problem in (7) asks for a minimizer of the convex function

$$-\mathcal{D}(\boldsymbol{\mu}) = \boldsymbol{\mu}^\intercal \boldsymbol{X}^\intercal \boldsymbol{X}\, \boldsymbol{\mu} - \boldsymbol{z}^\intercal \boldsymbol{\mu}.$$

Also, the non-negativity ($\boldsymbol{\mu} \succeq \mathbf{0}$) and sum-to-one ($\mathbf{1}^\intercal \boldsymbol{\mu} = 1$) constraints in (7) imply that any feasibly solution must reside in the standard simplex

$$\Delta^{n-1} = \Big\{ \boldsymbol{\mu} \in \mathbb{R}^n \;\Big|\; \mathbf{1}^\intercal \boldsymbol{\mu} = 1 \wedge \boldsymbol{\mu} \succeq \mathbf{0} \Big\}.$$

This insight allows us to write our problem in an even more compact manner, namely

$$\boldsymbol{\mu}_* = \underset{\boldsymbol{\mu} \in \Delta^{n-1}}{\operatorname{argmin}} \; -\mathcal{D}(\boldsymbol{\mu}) \tag{10}$$

Written like this, the dual MEB problem is now clearly recognizable as an instance of a convex minimization problem over a compact convex set. This is great because the Frank-Wolfe algorithm [12] provides a simple tool for solving this particular kind of problem [18].

Recall that the Frank-Wolfe algorithm is an iterative solver that proceeds like this: Given an initial feasible guess $\boldsymbol{\mu}_0$ for the solution, each iteration determines which $\boldsymbol{\nu}_t \in \Delta^{n-1}$ minimizes the inner product $-\boldsymbol{\nu}^\intercal \nabla D(\boldsymbol{\mu}_t)$ and applies a conditional gradient update $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \beta_t \cdot [\boldsymbol{\nu}_t - \boldsymbol{\mu}_t]$ where the step size $\beta_t = \frac{2}{t+2} \in [0, 1]$ decreases over time (for details as to this choice of step size, see [19], [20]). Updates will thus never leave the feasible set and the efficiency of the algorithm is due to the fact that it turns a quadratic problem into a series of linear ones.

Since the gradient of $-D(\boldsymbol{\mu})$ is $-\nabla D(\boldsymbol{\mu}) = 2\, \boldsymbol{X}^\intercal \boldsymbol{X} \boldsymbol{\mu} - \boldsymbol{z}$, each Frank-Wolfe iteration has to compute

$$\boldsymbol{\nu}_t = \underset{\boldsymbol{\nu} \in \Delta^{n-1}}{\operatorname{argmin}} \; \boldsymbol{\nu}^\intercal \big[2\, \boldsymbol{X}^\intercal \boldsymbol{X} \boldsymbol{\mu}_t - \boldsymbol{z} \big]. \tag{11}$$

This objective is linear in $\boldsymbol{\nu}$ and needs to be minimized over a compact convex set. Since minima of a linear functions over compact convex sets are necessarily attained at a vertex of that set, we realize that the solution of (11) must coincide with a vertex of Δ^{n-1}. Therefore, since the vertices of Δ^{n-1} correspond to the standard basis vectors $\boldsymbol{e}_j \in \mathbb{R}^n$, we can cast (11) simply as

$$\boldsymbol{e}_i = \underset{\boldsymbol{e}_j \in \mathbb{R}^n}{\operatorname{argmin}} \; \boldsymbol{e}_j^\intercal \big[2\, \boldsymbol{X}^\intercal \boldsymbol{X} \boldsymbol{\mu}_t - \boldsymbol{z} \big] \tag{12}$$

$$\Leftrightarrow \quad i = \underset{j \in \{1,\dots,n\}}{\operatorname{argmin}} \; \big[2\, \boldsymbol{X}^\intercal \boldsymbol{X} \boldsymbol{\mu}_t - \boldsymbol{z} \big]_j \tag{13}$$

All in all, solving the dual MEB problem in (7) is therefore as easy as shown in Algorithm 1.

Algorithm 1 Frank-Wolfe algorithm for the MEB problem

initialize a feasible point in Δ^{n-1}, for instance

$$\boldsymbol{\mu}_0 = \tfrac{1}{n}\mathbf{1}$$

for $t = 0, \dots, t_{\max}$ **do**

determine the step direction

$$i = \underset{j \in \{1,\dots,n\}}{\operatorname{argmin}} \big[2\, \boldsymbol{X}^\intercal \boldsymbol{X} \boldsymbol{\mu}_t - \boldsymbol{z} \big]_j$$

update the current estimate

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \tfrac{2}{t+2} \cdot [\boldsymbol{e}_i - \boldsymbol{\mu}_t]$$

III. Practice

Based on the pseudo-code in Alg. 1, it is straightforward to come up with *NumPy* code for MEB computation.

Throughout, we assume that we are given a *NumPy* array

```
matX = ...
```

which represents an $m \times n$ data matrix $\boldsymbol{X}$ whose columns $\boldsymbol{x}_j \in \mathbb{R}^m$ are the data points for which we want to determine the minimum enclosing ball.

Given this array `matX`, we can compute an array `matXtX` representing the matrix $\boldsymbol{X}^\intercal \boldsymbol{X}$ which features prominently in our problem. To this end, we simply use

```
matXtX = np.dot(matX.T, matX)
```

The next preparatory step is to compute the vector $\boldsymbol{z}$ whose entries are given by $z_j = \boldsymbol{x}_j^\intercal \boldsymbol{x}_j$. We therefore note that the inner products $\boldsymbol{x}_j^\intercal \boldsymbol{x}_j$ can be found along the diagonal of $\boldsymbol{X}^\intercal \boldsymbol{X}$, But this is to say that

$$\boldsymbol{z} = \operatorname{diag}\big[\boldsymbol{X}^\intercal \boldsymbol{X}\big]$$

so that we can compute

```
vecZ = np.diag(matXtX)
```

Given `matXtX` and `vecZ`, we can now run the Frank-Wolfe algorithm to determine the solution $\boldsymbol{\mu}_*$ to (7).

Listings 1 and 2 present two possible implementations *fwMEB_V1* and *fwMEB_V2* details of which we will discuss below. Here, we use *fwMEB_V2* to compute

```
vecM = fwMEB_V2(matXtX, vecZ)
```

Having determined $\boldsymbol{\mu}_*$, we apply (8) and (9) to determine the center $\boldsymbol{c}_*$ and radius r_* of the sought after MEB. This is as easy as

```
vecC = np.dot(matX, vecM)
r = np.sqrt(np.dot(vecZ, vecM) - \
            np.dot(vecM, np.dot(matXtX, vecM)))
```

Finally, in order to convince ourselves that we really just determined the minimum enclosing ball of the data in $\boldsymbol{X}$, we may plot our results (at least for 2D data) using

```
fig = plt.figure()
axs = fig.add_subplot()
axs.set_aspect(aspect='equal')
axs.set_axis_off()
axs.plot(matX[0,:], matX[1,:], 'o', c='k')
axs.add_patch(Circle(vecC, r, fc='none', ec='b'))
plt.show()
```

which should produce a picture similar to the one in Fig. 1(a).

Listing 1: Frank-Wolfe algorithm for solving (7) (version 1)

```
1  def fwMEB_V1(matXtX, vecZ, tmax=1000):
2      m, n = matXtX.shape
3
4      matI = np.identity(n)
5      vecM = np.ones(n) / n
6
7      for t in range(tmax):
8          beta = 2. / (t+2)
9          grad = 2. * np.dot(matXtX, vecM) - vecZ
10
11         imin = np.argmin(grad)
12
13         vecM += beta * (matI[imin] - vecM)
14
15     return vecM
```

Listing 2: Frank-Wolfe algorithm for solving (7) (version 2)

```
def fwMEB_V2(matXtX, vecZ, tmax=1000):
    m, n = matXtX.shape

    vecM = np.ones(n) / n

    for t in range(tmax):
        beta = 2. / (t+2)
        grad = 2. * np.dot(matXtX, vecM) - vecZ

        imin = np.argmin(grad)

        vecM *= (1-beta)
        vecM[imin] += beta

    return vecM
```

To conclude our discussion in this section, we still need to look at our implementations of the Frank-Wolfe algorithm in listings 1 and 2. While the former is more readable but less efficient, the latter is less readable but more efficient. To better convey our ideas behind these pieces of code, we first discuss function *fwMEB_V1* line by line:

- arguments of *fwMEB_V1* are the $n \times n$ matrix $\boldsymbol{X}^\intercal \boldsymbol{X}$, the n vector $\boldsymbol{z}$, and a parameter $t_{\max}$ indicating the desired number of Frank-Wolfe iterations
- in line 2, we determine the dimensions of $\boldsymbol{X}^\intercal \boldsymbol{X}$
- in line 4, we introduce a "helper" matrix, namely the $n \times n$ identity matrix $\boldsymbol{I}$ for which we note that

$$\begin{aligned} \boldsymbol{I} &= \begin{bmatrix} 1 & 0 & \cdots & 0 \\ 0 & 1 & \cdots & 0 \\ \vdots & \vdots & \ddots & \vdots \\ 0 & 0 & \cdots & 1 \end{bmatrix} \\ &= \begin{bmatrix} \boldsymbol{e}_1 & \boldsymbol{e}_2 & \cdots & \boldsymbol{e}_n \end{bmatrix} \end{aligned}$$

- in line 5, we initialize the vector $\boldsymbol{\mu}$ to $\frac{1}{n}\mathbf{1}$ and observe that this is the first step of algorithm 1
- the *for* loop in line 7 then realizes a total of $t_{\max}$ Frank-Wolfe iterations
- in line 8, we compute the current step size

$$\beta_t = \tfrac{2}{t+2}$$

- in line 9, we compute the current gradient

$$-\nabla D(\boldsymbol{\mu}_t) = 2\,\boldsymbol{X}^\intercal \boldsymbol{X} \boldsymbol{\mu}_t - \boldsymbol{z}$$

- in line 11, we use *argmin* to determine the index i of the smallest entry of the current gradient
- in line 13, we make use of β_t and i to compute the update

$$\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \beta_t \cdot [\boldsymbol{e}_i - \boldsymbol{\mu}_t]$$

  Note: the expression `matI[imin]` is a *NumPy*-thonic way of representing $\boldsymbol{e}_i$
- finally, upon termination of the *for* loop, line 15 returns the iteratively optimized vector $\boldsymbol{\mu}$

Our reason for believing that this implementation is easy to read is that it involves an almost literal implementation of the update $\boldsymbol{\mu}_{t+1} = \boldsymbol{\mu}_t + \beta_t \cdot [\boldsymbol{e}_i - \boldsymbol{\mu}_t]$ in algorith 1. However, this comes at the cost of having to work with the "helper" matrix $\boldsymbol{I}$ which entails some memory overhead. This overhead could be substantial, especially if n was large.

A more efficient and less memory intensive implementation can be based on the following considerations:

First of all we note that

$$\begin{aligned} \boldsymbol{\mu}_{t+1} &= \boldsymbol{\mu}_t + \beta_t \cdot [\boldsymbol{e}_i - \boldsymbol{\mu}_t] \\ &= (1 - \beta_t) \cdot \boldsymbol{\mu}_t + \beta_t \cdot \boldsymbol{e}_i. \end{aligned}$$

Second of all, we observe that only the i-th entry of $\boldsymbol{e}_i$ differs from zero so that the addition of $\beta_t \cdot \boldsymbol{e}_i$ only affects the i-th entry of $\boldsymbol{\mu}$. Also, since the the i-th entry of $\boldsymbol{e}_i$ is one, adding $\beta_t \cdot \boldsymbol{e}_i$ to $\boldsymbol{\mu}$ is the same as adding β_t to the i-th entry of $\boldsymbol{\mu}$.

Function *fwMEB_V2* in Listing 2 exploits these "tricks". Its computational steps are largely identical to the ones we discussed above, however, it makes do without computing / representing Euclidean basis vectors $\boldsymbol{e}_i$.

IV. Summary and Outlook

In this note, we were concerned with computing the minimum enclosing ball of a given set of data points. We saw that this problem can be formalized as a constrained quadratic optimization problem whose dual can be solved using the Frank-Wolfe algorithm. All in all, our discussion revealed that computing the MEB of a given data set is but a matter of only a few lines of *NumPy* code.

Note that, once center and radius of a the MEB of a set of data have been determined, they allow for a characteristic, very compressed representation of that data. After all, all the given data reside within the ball with center $\boldsymbol{c}_*$ and radius r_*. In other words, we could consider only two parameters to represent a (ball-shaped) region in a data- or feature space that surrounds the given sample. Also note that learning- or analysis based on such "surrounding region" representations is typically easy to accomplish and often yields interpretable or explainable results [21]–[25]. To substantiate these claims, we will study corresponding methods in in several upcoming notes. Interestingly enough, the Frank-Wolfe algorithm will again feature prominently in these discussions.

ACKNOWLEDGMENT

The material has been produced in project P3ML which was funded by the Ministry of Education and Research of Germany (BMBF) under grant number 01IS17064.

REFERENCES

[1] I. Tsang, J. Kwok, and P.-M. Cheung, "Core Vector Machines: Fast SVM Training on Very Large Data Sets," *J. of Machine Learning Research*, vol. 6, 2005.

[2] F. Nielsen and R. Nock, "Approximating Smallest Enclosing Balls with Application to Machine Learning," *Int. J. Computational Geometry & Applications*, vol. 19, no. 5, 2009.

[3] T. Dong, C. Bauckhage, H. Jin, J. Li, O. Cremers, D. Speicher, A. Cremers, and J. Zimmermann, "Imposing Category Trees onto Word-Embeddings Using a Geometric Construction," in *Proc. ICLR*, 2019.

[4] T. Dong, Z. Wang, J. Li, C. Bauckhage, and A. Cremers, "Triple Classification Using Regions and Fine-Grained Entity Typing," in *Proc. AAAI*, 2019.

[5] F.-M. Schleif, A. Gisbrecht, and P. Tino, "Supervised Low Rank Indefinite Kernel Approximation Using Minimum Enclosing Balls," *Neurocomputing*, vol. 318, no. Nov, 2018.

[6] C. Bauckhage and R. Sifa, "Joint Selection of Central and Extremal Prototypes Based on Kernel Minimum Enclosing Balls," in *Proc. DSAA*. IEEE, 2019.

[7] Y. Wang, Y. Li, and K.-L. Tan, "Coresets for Minimum Enclosing Balls over Sliding Windows," in *Proc. KDD*. ACM, 2019.

[8] R. Sifa and C. Bauckhage, "Novelty Discovery with Kernel Minimum Enclosing Balls," in *Proc. LION*. Springer, 2020.

[9] B. Schölkopf, R. Williamson, A. Smola, J. Shaw-Taylor, and J. Platt, "Support Vector Method for Novelty Detection," in *Proc. NIPS*, 1999.

[10] A. Ben-Hur, D. Horn, H. Siegelmann, and V. Vapnik, "Support Vector Clustering," *J. of Machine Learning Research*, vol. 2, 2001.

[11] D. Tax and R. Duin, "Support Vector Data Description," *Machine Learning*, vol. 54, no. 1, 2004.

[12] M. Frank and P. Wolfe, "An Algorithm for Quadratic Programming," *Naval Research Logistics Quarterly*, vol. 3, no. 1–2, 1956.

[13] C. Bauckhage and T. Dong, "Lecture Notes on Machine Learning: Minimum Enclosing Balls," B-IT, University of Bonn, 2019.

[14] C. Bauckhage and T. Dong, "Lecture Notes on Machine Learning: Frank-Wolfe for Minimum Enclosing Balls," B-IT, University of Bonn, 2019.

[15] C. Bauckhage and D. Speicher, "Lecture Notes on Machine Learning: The Karush-Kuhn-Tucker Conditions (Part 1)," B-IT, University of Bonn, 2019.

[16] C. Bauckhage and T. Dong, "Lecture Notes on Machine Learning: The Karush-Kuhn-Tucker Conditions (Part 2)," B-IT, University of Bonn, 2019.

[17] T. Oliphant, "Python for Scientific Computing," *Computing in Science & Engineering*, vol. 9, no. 3, 2007.

[18] R. Sifa, "An Overview of Frank-Wolfe Optimization for Stochasticity Constrained Interpretable Matrix and Tensor Factorization," in *Proc. ICANN*, 2018.

[19] K. Clarkson, "Coresets, Sparse Greedy Approximation, and the Frank-Wolfe Algorithm," *ACM Trans. on Algorithms*, vol. 6, no. 4, 2010.

[20] M. Jaggi, "Revisiting Frank-Wolfe: Projection-Free Sparse Convex Optimization," *J. of Machine Learning Research*, vol. 28, no. 1, pp. 427–435, 2013.

[21] H. Cevikalp, B. Triggs, and R. Polikar, "Nearest Hyperdisk Methods for High-Dimensional Classification," in *Proc. ICML*, 2008.

[22] C. Thurau, "Nearest Archetype Hull Methods for Large-scale Data Classification," in *Proc. ICPR*. IEEE, 2010.

[23] G. Evangelidis and C. Bauckhage, "Efficient Subframe Video Alignment Using Short Descriptors," *IEEE Trans. Pattern Analysis and Machine Intelligence*, vol. 35, no. 10, 2013.

[24] R. Sifa, C. Bauckhage, and A. Drachen, "Archetypal Game Recommender Systems," in *Proc. KDML-LWA*, 2014.

[25] M. Cheema, A. Eweiwi, and C. Bauckhage, "High Dimensional Low Sample Size Activity Recognition Using Geometric Classifiers," *Digital Signal Processing*, vol. 42, 2015.