UNIVERSITÀ
DEGLI STUDI
DI PADOVA

Minimum Enclosing Ball and Anomaly Detection using variants of Frank-Wolfe

Optimization for Data Science

Project report

Authors:
Marija Cveevska
Dejan Dichoski
Suleyman Erim

Supervisor:
Francesco Rinaldi

September 22, 2023

Abstract

This study explores the efficacy of three Frank Wolfe algorithm adaptations, namely Away-steps Frank Wolfe, Blended Pairwise Conditional Gradient, and (1+ε)-approximation to MEB, in solving the MEB problem. Our investigation reveals that Blended Pairwise Conditional Gradient consistently outperforms the other algorithms, demonstrating superior convergence rates and efficiency. While Away-steps Frank Wolfe exhibits satisfactory results, (1+ε)-approximation to MEB shows variable performance, with substantial improvements observed when using a larger ε. We emphasize the dataset's influence on algorithm performance, represented by high F1 scores in well-separable data like the Breast Cancer Dataset and lower scores in less separable data like the Iranian Churn Dataset. This underscores the need for a nuanced understanding of data characteristics when selecting optimization algorithms. Our research provides valuable insights into optimizing the Minimum Enclosing Ball problem, offering practical implications for diverse project applications.

Introduction

The Minimum Enclosing Ball (MEB) problem can be succinctly described as follows: given a dataset of numerical values, the objective is to identify the smallest possible ball that encompasses a certain percentage of the data points. However, as is often the case in mathematics, a seemingly straightforward problem statement does not necessarily equate to a straightforward solution.

MEBs offer a highly versatile means of data representation, serving a wide array of purposes in both learning and analytical contexts. These applications include, among others, the acceleration of training support vector machines and other classifiers, symbolic learning, the identification of landmarks or prototypes, the analysis of data streams, and the detection of anomalies.

In this project, our primary focus lies on anomaly detection. MEBs also form the core of concepts like one-class support vector machines, support vector clustering, and support vector data description. While there exists a plethora of algorithms for estimating the MEB for a given dataset, a notably versatile approach involves framing the problem within the context of constrained quadratic optimization—a strategy that we, too, will adopt.

The significance of minimum enclosing balls extends across various domains, including clustering, nearest neighbor search, data classification, support vector machines, machine learning, facility location, collision detection, computer graphics, and military operations. In particular, many of these applications give rise to MEB problems of considerable scale, where achieving a reasonably high level of accuracy suffices.

Though the primal problem of estimating an MEB can be complicated, we will discover that its dual counterpart lends itself to relatively straightforward solutions. Indeed, the dual MEB problem assumes a form that permits the utilization of uncomplicated solvers, such as the basic Frank-Wolfe algorithm and its variants. This project will delve into this aspect in comprehensive detail. Moreover, we will undertake the implementation and testing of three Frank-Wolfe variant algorithms, followed by a comparison of their results.

Notation

We use bold letters to denote vectors and matrices, while non-bold letters represent constants. For a vector p, p_i^t denotes its i-th component at iteration t. Inequality statements involving vectors apply to each of their components. Functions and operators are represented using uppercase letters. Uppercase script letters are employed to denote various objects, including sets and balls.

Problem definition

The primal MEB problem

In the minimum enclosing ball problem, given a finite set of vectors $\boldsymbol{\mathcal{A}} = \{a_1, \ldots, a_m\} \subset \mathbb{R}^n$, the objective is to determine the smallest n-dimensional ball that contains all the points in $\boldsymbol{\mathcal{A}}$ [1]:

$$\boldsymbol{\mathcal{B}}_{\boldsymbol{c},r} = \{\boldsymbol{a} \in \mathbb{R}^n \mid \|\boldsymbol{a} - \boldsymbol{c}\| \leq r\} \qquad (1)$$

Since Euclidean balls are uniquely defined in terms of their center $c \in \mathbb{R}^n$ and radius $r \in \mathbb{R}$, the basic problem is therefore to determine the optimal choices c^* and r^* for these two parameters. In this report, we approach this problem from the point of view of constrained convex optimization. Indeed, noting the equivalences [2]:

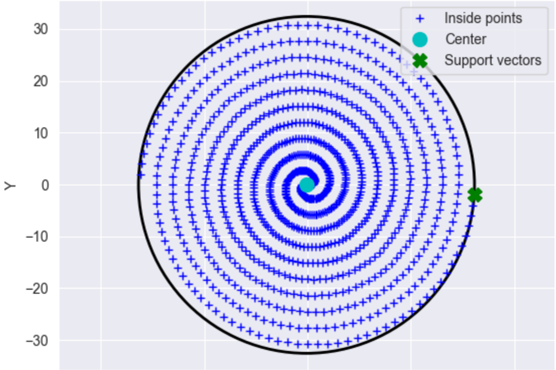

Figure 1: Data points forming a Fermat's spiral, and their minimum enclosing ball (which in 2D is a circle).

$$\|\boldsymbol{a} - \boldsymbol{c}\| \leq r \iff \|\boldsymbol{a} - \boldsymbol{c}\|^2 \leq r^2 \iff \|\boldsymbol{a} - \boldsymbol{c}\|^2 - r^2 \leq 0 \qquad (2)$$

we can immediately formalize the problem of finding $\boldsymbol{c}^*$ and r^* as an inequality constrained quadratic minimization problem, namely [1]:

$$\boldsymbol{c}^*, r^* = \underset{\boldsymbol{c},r}{\operatorname{argmin}}\, r^2 \;\; s.t. \;\; \begin{array}{c} \|\boldsymbol{a_1} - \boldsymbol{c}\|^2 - r^2 \leq 0 \\ \|\boldsymbol{a_2} - \boldsymbol{c}\|^2 - r^2 \leq 0 \\ \vdots \\ \|\boldsymbol{a_m} - \boldsymbol{c}\|^2 - r^2 \leq 0 \end{array} \qquad (3)$$

which we can rewrite as:

$$\boldsymbol{c}^*, r^* = \underset{\boldsymbol{c},r}{\operatorname{argmin}}\, r^2 \;\; s.t. \;\; \|\boldsymbol{a_i} - \boldsymbol{c}\|^2 - r^2 \leq 0, \qquad i = 1, \ldots, m \qquad (4)$$

In simpler terms, 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. In other words, 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 [1].

While there are algorithms that can solve (4) directly, a more general approach consists of working with the corresponding dual problem which we will derive next.

By setting $r^2 = \gamma$, we obtain:

$$\boldsymbol{c}^*, r^* = \underset{\boldsymbol{c},\gamma}{\operatorname{argmin}}\, \gamma\,, \qquad \text{s.t.} \quad \boldsymbol{a_i^T a_i} - 2\boldsymbol{a_i^T c} + \boldsymbol{c^T c} - \gamma \leq 0\,, \qquad i = 1, \ldots, m \qquad (5)$$

Our goal is to show that the above primal (minimization) problem has the following dual (maximization) problem [1]:

$$\boldsymbol{u}^* = \underset{\boldsymbol{u}}{\operatorname{argmax}} \sum_{i=1}^{m} u_i \boldsymbol{a_i^T a^i} - \left(\sum_{i=1}^{m} u_i \boldsymbol{a_i}\right)^T \sum_{i=1}^{m} u_i \boldsymbol{a_i}\,, \qquad s.t. \; \sum_{i=1}^{m} u_i = 1, \boldsymbol{u} \geq \boldsymbol{0} \qquad (6)$$

where $\boldsymbol{u} \in \mathbb{R}^n$ is a vector of Lagrange multipliers.

Deriving the dual problem

To begin with, we write down the Lagrangian of the problem in (5). Letting u_i denote the Lagrange multiplier for the i-th inequality in (3), we have:

$$\mathcal{L}(\boldsymbol{c}, r, \boldsymbol{u}) = r^2 + \sum_{i=1}^{m} u_i \left(\|\boldsymbol{a_i} - \boldsymbol{c}\| - r^2 \right) \tag{7}$$

$$= r^2 + \sum_{i=1}^{m} u_i \left(\boldsymbol{a_i^T a_i} - 2\boldsymbol{c^T a_i} + \boldsymbol{c^T c} - r^2 \right) \tag{8}$$

$$= r^2 + \sum_{i=1}^{m} u_i \, z_i - 2\boldsymbol{c^T} \left[\sum_{i=1}^{m} \boldsymbol{a}_i u_i \right] + \boldsymbol{c^T c} \sum_{i=1}^{m} u_i - r^2 \sum_{i=1}^{m} u_i \tag{9}$$

Notice that, when expanding the expression in (8) to the one in (9), we introduced the shorthand $z_i = \boldsymbol{a_i^T a_i}$ just to reduce notational clutter [1].

In fact, we can write our Lagrangian even more compactly [1]. To do this, we first note that:

$$\sum_{i=1}^{m} u_i = \boldsymbol{u^T 1}, \qquad \sum_{i=1}^{m} u_i z_i = \boldsymbol{u^T z}, \qquad \sum_{i=1}^{m} \boldsymbol{a}_i u_i = \boldsymbol{Au} \tag{10}$$

where $A = [a_1, \ldots, a_m] \in \mathbb{R}^{nxm}$ is a data matrix whose columns correspond to the data points in $\mathcal{A}$. Hence, the Lagrangian in (9) can also be written as:

$$\mathcal{L}(\boldsymbol{c}, r, \boldsymbol{u}) = r^2 + \boldsymbol{u^T z} - 2\boldsymbol{c^T A u} + \boldsymbol{c^T c} \cdot \boldsymbol{u^T 1} - r^2 \boldsymbol{u^T 1} \tag{11}$$

Given this Lagrangian, we will now evaluate the Karush-Kuhn-Tucker conditions [1]. Focusing on the KKT 1 conditions (stationarity), we have:

$$\boldsymbol{0} \overset{!}{=} \frac{\partial \mathcal{L}}{\partial c} = -2\boldsymbol{Au} + 2(\boldsymbol{u^T 1})c \;\Rightarrow\; \boldsymbol{c} = \frac{\boldsymbol{Au}}{\boldsymbol{u^T 1}} \tag{12}$$

$$\boldsymbol{0} \overset{!}{=} \frac{\partial \mathcal{L}}{\partial r} = r - r\boldsymbol{u^T 1} = r(1 - \boldsymbol{u^T 1}) \Rightarrow \boldsymbol{u^T 1} = 1 \tag{13}$$

When we substitute the result from (13) into (12), the latter simplifies to:

$$\boldsymbol{c} = \boldsymbol{Au} \tag{14}$$

Furthermore, by plugging the expressions from (13) and (14) back into the Lagrangian in (11), we obtain the Lagrangian dual [1], since:

$$\mathcal{L}(\boldsymbol{c}, r, \boldsymbol{u}) = r^2 + \boldsymbol{u^T z} - 2\boldsymbol{c^T A u} + \boldsymbol{c^T c} \cdot \boldsymbol{u^T 1} - r^2 \boldsymbol{u^T 1} \tag{15}$$

$$= r^2 + \boldsymbol{u^T z} - 2\boldsymbol{c^T A u} + \boldsymbol{c^T c} - r^2 \tag{16}$$

$$= \boldsymbol{u^T z} - 2\boldsymbol{u^T A^T A u} + \boldsymbol{u^T A^T A u} \tag{17}$$

$$= \boldsymbol{u^T z} - \boldsymbol{u^T A^T A u} \tag{18}$$

$$= \Phi(\mathbf{u}) \tag{19}$$

is a function that depends solely on the Lagrange multipliers $\boldsymbol{u}$.

Now, if we further note that the KKT 3 conditions (dual feasibility) require $\boldsymbol{u} \geq \boldsymbol{0}$, all our (intermediate) results and considerations so far indicate that the dual of (4) amounts to [1]:

$$\boldsymbol{u}^* = \underset{\boldsymbol{u}}{\operatorname{argmax}} \, \Phi(\boldsymbol{u}) = \underset{\boldsymbol{u}}{\operatorname{argmax}} \, \boldsymbol{u^T z} - \boldsymbol{u^T A^T A u} \quad s.t. \; \boldsymbol{u^T 1} = 1, \boldsymbol{u} \geq \boldsymbol{0} \tag{20}$$

Finally, it's worth noting that $\Phi(\boldsymbol{u}) = \boldsymbol{u^T z} - \boldsymbol{u^T A^T A u}$ is concave in $\boldsymbol{u}$, making$-\Phi(\boldsymbol{u})$ is convex. Therefore, the solution to the maximization problem in (20) can also be found by considering an equivalent minimization problem, namely:

$$\boldsymbol{u}^* = \underset{\boldsymbol{u}}{\operatorname{argmin}} \, -\Phi(\boldsymbol{u}) = \underset{\boldsymbol{u}}{\operatorname{argmin}} \, \boldsymbol{u^T A^T A u} - \boldsymbol{u^T z} \quad s.t. \quad \boldsymbol{u^T 1} = 1, \boldsymbol{u} \geq \boldsymbol{0} \tag{21}$$

From Lagrange multipliers to MEB parameters

If we could solve the dual problem in (20), we would obtain an optimal Lagrange multiplier vector, denoted as $\boldsymbol{u}^*$ [2]. Assuming we could determine $\boldsymbol{u}^*$, it would immediately enable us to compute the center $\boldsymbol{c}^*$ of the sought-after MEB, as per equation (14):

$$\boldsymbol{c}^* = \boldsymbol{A}\boldsymbol{u}^* \tag{22}$$

But what about the corresponding radius r^*?

The radius r^* of the minimum enclosing ball can be determined by considering the KKT 4 conditions (complementary slackness) [3]. These conditions dictate that at a solution, we must have:

$$u_i^*\left(z_i - 2\boldsymbol{c}^{*\boldsymbol{T}}\boldsymbol{a_i} + \boldsymbol{c}^{*\boldsymbol{T}}\boldsymbol{c}^* - r^{*T}\right) = 0, \qquad i = 1, \dots, m \tag{23}$$

Summing these equations over all i then yields [3]:

$$\sum_{i=1}^{m} u_i^*\left(z_i - 2\boldsymbol{c}^{*\boldsymbol{T}}\boldsymbol{a_i} + \boldsymbol{c}^{*\boldsymbol{T}}\boldsymbol{c}^* - r^{*2}\right) = 0 \tag{24}$$

With additional algebraic manipulations based on our previous calculations, we arrive at:

$$r^{*T} = \boldsymbol{u}^{*\boldsymbol{T}}\boldsymbol{z} - \boldsymbol{u}^{*\boldsymbol{T}}\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{u}^* \tag{25}$$

In other words, we find that the radius of the minimum enclosing ball for the given data is:

$$r^* = \sqrt{-\Phi(\boldsymbol{u}^*)} \tag{26}$$

Frank-Wolfe for Minimum enclosing balls

Reiterating what we discussed above, we emphasize that the dual MEB problem in (21) seeks a minimizer of the convex function (21). Additionally, the non-negativity ($\boldsymbol{u} \geq \boldsymbol{0}$) and sum-to-one ($\boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{1} = 1$) constraints in (21) imply that any feasibly solution must reside within the standard simplex [2]:

$$\Delta^{m-1} = \{\boldsymbol{u} \in \mathbb{R}^m \mid \boldsymbol{u}^{\boldsymbol{T}}\boldsymbol{1} = 1 \wedge \boldsymbol{u} \geq \boldsymbol{0}\} \tag{27}$$

This insight allows us to express our problem in an even more compact manner, namely [2]:

$$\boldsymbol{u}^* = \underset{\boldsymbol{u} \in \Delta^{m-1}}{\operatorname{argmin}} -\Phi(\boldsymbol{u}) \tag{28}$$

Written like this, the dual MEB problem is now clearly recognizable as an instance of a convex minimization problem over a compact convex set [2]. This is significant because the Frank Wolfe algorithm provides a straightforward approach for solving this particular kind of problem. Next, we show how to specialize the general form of the Frank-Wolfe to the minimum enclosing ball problem in (28).

The Frank-Wolfe algorithm is an iterative solver. First, it makes an initial feasible guess u_0 for the solution. The midpoint of Δ^{m-1} is contained within Δ^{m-1} and is therefore feasible. This means that we can use $\boldsymbol{u_0} = \frac{1}{n}\boldsymbol{1}$ as a starting point. However, for simplicity in our implementation, we decided to use the first row of the basis matrix.

In each iteration, the algorithm determines $\boldsymbol{s_t} \in \Delta^{m-1}$ that minimizes the inner product $-\boldsymbol{s_t}\nabla\Phi(\boldsymbol{u_t})$ and applies a conditional gradient update $\boldsymbol{u^{t+1}} = \boldsymbol{u^t}_{\boldsymbol{t}} + \alpha_t \cdot [\boldsymbol{s^t} - \boldsymbol{u^t}]$, where the step size $\alpha_t \in [0, 1]$ decreases over time. Updates will thus never leave the feasible set and the efficiency of the algorithm is due to its ability to transform a quadratic problem into a series of linear ones [2].

Calculating the step direction $\boldsymbol{s_t}$ involves determining the gradient of the objective function at the current estimate. In our specific case, this gradient is given by:

$$\nabla\left(-\Phi(\boldsymbol{u})\right) = 2\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{u} - \boldsymbol{z} \tag{29}$$

Therefore, each Frank-Wolfe iteration has to solve:

$$\boldsymbol{s_t} = \underset{\boldsymbol{s}\in\Delta^{m-1}}{\operatorname{argmin}}\ \boldsymbol{s}^{\boldsymbol{T}}[2\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{u_t} - \boldsymbol{z}] \tag{30}$$

Note that the expression on the right-hand side is linear in $\boldsymbol{s}$ and needs to be minimized over Δ^{m-1}, which is a compact convex set. The minimum of a linear function over a convex set is necessarily achieved at a vertex of said set. Since the vertices of Δ^{m-1} correspond to the standard basis vectors $\boldsymbol{e_1}, \dots, \boldsymbol{e_m}$ of $\mathbb{R}^m$, we only need to determine which of these minimizes the inner product mentioned above. Hence, the problem in (30) simplifies to [2]:

$$\boldsymbol{s_t} = \underset{\boldsymbol{e_i}\in\mathbb{R}^m}{\operatorname{argmin}}\ \boldsymbol{e_i^T}[2\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{u_t} - \boldsymbol{z}] \tag{31}$$

To make it explicit that the resulting $\boldsymbol{s_t}$ will be a standard basis vector $\boldsymbol{e_j}$, we can also express it as:

$$\boldsymbol{e_j} = \underset{\boldsymbol{e_i}\in\mathbb{R}^m}{\operatorname{argmin}}\ \boldsymbol{e_i}^{\boldsymbol{T}}[2\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{u_t} - \boldsymbol{z}] \tag{32}$$

We can solve this without computing inner products; we only need to identify the index corresponding to the smallest component of the gradient, which is [3]:

$$j = \underset{i\in\{1,\dots,m\}}{\operatorname{argmin}}\ [2\boldsymbol{A}^{\boldsymbol{T}}\boldsymbol{A}\boldsymbol{u_t} - \boldsymbol{z}]_i \tag{33}$$

Given these considerations, the general update step for the Frank-Wolfe algorithm thus specializes to [3]:

$$\boldsymbol{u_{t+1}} = \boldsymbol{u_t} + \alpha_t\big[\boldsymbol{e_j} - \boldsymbol{u_t}\big] \tag{34}$$

In conclusion, solving the dual MEB problem can be fairly straightforward by applying the original Frank-Wolfe algorithm and its variations. In this project, we examined, implemented, and compared three different versions of the Frank-Wolfe algorithm. Next, we will introduce each of these algorithms and present our results by applying them to address the MEB problem in the context of anomaly detection.

Away-steps Frank-Wolfe algorithm

The convergence rate of the original Frank Wolfe algorithm is known to be slow (sublinear) when the solution lies on the boundary. A simple improvement is to introduce the possibility of taking 'away steps' during optimization, an operation that importantly does not require a feasibility oracle. The away-steps variant of the Frank-Wolfe algorithm, which can also remove weight from 'bad' atoms in the current active set, was proposed by P. Wolfe. The precise method is outlined below in Algorithm 1 [4].

When the optimal solution lies on the boundary of $\boldsymbol{C} = conv(\{\boldsymbol{e_1}, \dots, \boldsymbol{e_n}\})$, the iterates of the classical FW algorithm start to zig-zag between the vertices defining the face containing the optimal solution. This results in a slow convergence rate (sublinear) [4].

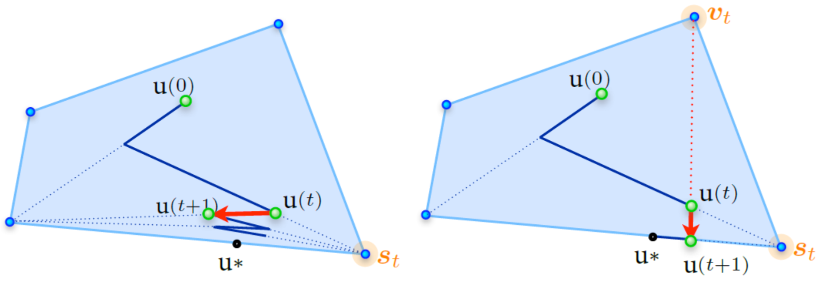

Figure 2: (left) The FW algorithm zig-zags when the solution $\boldsymbol{u}^*$ lies on the boundary. (middle) Adding the possibility of an away step mitigates this problem. (right) As an alternative, a pairwise FW step.

To address the zig-zagging issue in the FW algorithm, Wolfe proposed adding the possibility of moving away from an active atom in $\boldsymbol{S^t}$ (as shown in the middle of Figure 1). This simple modification is sufficient to achieve linear convergence for strongly convex functions. We describe the away-steps variant of Frank-Wolfe algorithm in Algorithm 1.

The away direction, denoted as $\boldsymbol{d_t^A}$**,** is defined in line 4 by finding the atom $\boldsymbol{v^t}$ in $\boldsymbol{S^t}$ that maximizes the potential descent given, by $g\_A^t \coloneqq \langle -\nabla\Phi(\boldsymbol{u^t}), \boldsymbol{u^t} - \boldsymbol{v^t}\rangle$. It's important to note that this search is performed over the (typically small) active set $\boldsymbol{S^t}$, making it fundamentally easier than the linear oracle LMO_A. The maximum step-size α_{max}, defined in line 9, ensures that the new iterate $\boldsymbol{u^t} + \alpha\boldsymbol{d\_A^t}$ remains within conv($\boldsymbol{S^t}$) $\subseteq C$, which guarantees that the convex representation is maintained. Computing the true maximum feasible step-size would require the ability to know when we cross the boundary of $\boldsymbol{C}$ along a specific line, which is generally possible for arbitrary $\boldsymbol{C}$. By using the conservative maximum step-size outlined in line 9, we eliminate the need for this more powerful oracle. This is why Algorithm 1 requires maintaining $\boldsymbol{S^t}$, unlike the standard FW algorithm. Finally, as in classical FW, the FW gap g_FW^t serves as an upper bound on the unknown suboptimality and can be used as a stopping criterion [4]:

$$g_{FW}{}^t \coloneqq \langle -\nabla\Phi(\boldsymbol{u^t}), \boldsymbol{d_{FW}}{}^t\rangle \geq \langle -\nabla\Phi(\boldsymbol{u^t}), \boldsymbol{u}^* - \boldsymbol{u^t}\rangle \geq \Phi(\boldsymbol{u^t}) - \Phi(\boldsymbol{u}^*) \quad (by\ convexity) \tag{35}$$

If $\alpha^t = \alpha_{max}$, then we refer to this step as a drop step, as it completely removes the atom $\boldsymbol{v^t}$ from the currently active set of atoms, $\boldsymbol{S^t}$, (by settings its weight to zero). We cannot guarantee sufficient progress in this case, but the drop step decreases the active set size by one, and thus, they cannot occur too frequently (not more than half the time) [4].

The weight updates for lines 12 and 13 take the following form: For a FW step, we have $\boldsymbol{S^{t+1}} = \{\boldsymbol{s^t}\}$ if $\alpha^t = 1$; otherwise $\boldsymbol{S^{t+1}} = \boldsymbol{S^t} \cup \{\boldsymbol{s^t}\}$. Additionally, we have $\omega_{\boldsymbol{S_t}}^{t+1} \leftarrow (1-\alpha^t)\omega_{\boldsymbol{S_t}}^t + \alpha^t$ and $\omega_{\boldsymbol{v_t}}^{t+1} \leftarrow (1-\alpha^t)\omega_{\boldsymbol{v_t}}^t$ for $\boldsymbol{v} \in \boldsymbol{S^t} \setminus \{\boldsymbol{s^t}\}$. For an away step, we have $\boldsymbol{S^{t+1}} = \{\boldsymbol{v^t}\}$ if $\alpha^t = \alpha_{max}$ (a drop step); otherwise $\boldsymbol{S^{t+1}} = \boldsymbol{S^t}$. Also, we have $\omega_{\boldsymbol{v_t}}^{t+1} \leftarrow (1+\alpha^t)\omega_{\boldsymbol{v_t}}^t + \alpha^t$ and $\omega_{\boldsymbol{v}}^{t+1} \leftarrow (1+\alpha^t)\omega_{\boldsymbol{v}}^t$ for $\boldsymbol{v} \in \boldsymbol{S^t}\setminus\{\boldsymbol{v^t}\}$ [4].

Algorithm 1: Away-steps Frank-Wolfe algorithm

```
1:  $\boldsymbol{u^0} \leftarrow [\boldsymbol{1\ 0\ 0 \ldots 0}] \in \mathbb{R}^m$, $\boldsymbol{S^0} \leftarrow \{\boldsymbol{u^0}\}$        // so that $\omega_v^0 = 1$ for $\boldsymbol{v} = \boldsymbol{u^0}$ and 0 otherwise
2:  For $t = 0, \ldots, maxIter - 1$ do
3:      $\boldsymbol{s^t} \leftarrow \mathrm{LMO}_A(\nabla\Phi(\boldsymbol{u^t}))$ and $d\_FW^t \leftarrow \boldsymbol{s^t} - \boldsymbol{u^t}$        // the FW direction
4:      $\boldsymbol{v^t} \leftarrow \operatorname{argmax}_{\boldsymbol{v} \in \boldsymbol{S^t}} \langle\nabla\Phi(\boldsymbol{u^t}), \boldsymbol{v}\rangle$ and $d\_A^t \leftarrow \boldsymbol{u^t} - \boldsymbol{v^t}$        // the away direction
5:      if $(g\_FW^t \leftarrow \langle -\nabla\Phi(\boldsymbol{u_t}), \boldsymbol{d\_FW^t}\rangle) \leq \varepsilon$ then return $\boldsymbol{u^t}$
6:      if $\langle -\nabla\Phi(\boldsymbol{u_t}), \boldsymbol{d\_FW^t}\rangle \geq \langle -\nabla\Phi(\boldsymbol{u^t}), \boldsymbol{d\_A^t}\rangle$ then
7:          $\boldsymbol{d^t} \leftarrow \boldsymbol{d\_FW^t}$, and $\alpha_{max} = 1$        // choose the FW direction
8:      else
9:          $\boldsymbol{d^t} \leftarrow \boldsymbol{d\_A^t}$, and $\alpha_{max} = \frac{\omega_{v^t}}{1 - \omega_{v^t}}$        // choose away direction; maximum feasible step-size
10:     end if
11:     Line-search: $\alpha^t \in \operatorname{argmin}_{\alpha \in [0, \alpha_{max}]} \Phi(\boldsymbol{u^t} + \alpha\boldsymbol{d^t})$
12:     $\boldsymbol{u^t} \leftarrow \boldsymbol{u^{t-1}} + \alpha^t\boldsymbol{d^t}$        // and accordingly for the weights $\boldsymbol{\omega^t}$
13:     $\boldsymbol{S^t} \leftarrow \{\boldsymbol{v}\ \mathrm{s.t.}\ \omega_v^t > 0\}$
14: end for
```

Blended Pairwise Conditional Gradients (BPCG)

The Pairwise Conditional Gradients algorithm is a natural generalization of the Away-step Frank-Wolfe algorithm. Moreover, it uses the blending criterion introduced in (Braun et al., 2019a) to efficiently combine local PCG steps with global Frank-Wolfe steps. Unfortunately, PCG exhibits so-called swap steps that might not provide sufficient primal progress. The number of these suboptimal steps is bounded by a function of the dimension, and therefore, known guarantees do not generalize to the infinite-dimensional case [5].

A simpler modification of the PCG algorithm can be achieved by combining it with the blending criterion from the Blended Conditional Gradients Algorithm. This new algorithm is called the Blended Pairwise Conditional Gradients. BPCG eliminates the occurrence of swap steps. It is straightforward to implement, and does not require any internal gradient alignment procedures. The convergence rate of BPCG is basically that of PCG, if no drop steps would occur. As such, it improves the convergence rates of PCG. Additionally, numerical experiments show that BPCG's solutions are much sparser than those of PCG [5].

The Blended Pairwise Conditional Gradients algorithm (BPCG) is presented in Algorithm 2 [5]. As mentioned earlier, our focus is on the general smooth convex case and the strongly convex case over polytopes.

Algorithm 2: Blended Pairwise Conditional Gradients (BPCG)

```
1:  $\boldsymbol{u^0} \leftarrow [\mathbf{1\ 0\ 0} \ldots \mathbf{0}] \in \mathbb{R}^m$, $\boldsymbol{S^0} \leftarrow \{\boldsymbol{u^0}\}$
2:  for $t = 0, \ldots, maxIter - 1$ do:
3:      $a^t \leftarrow \underset{\boldsymbol{v} \in \boldsymbol{S^t}}{\text{argmax}} \langle \nabla\Phi(\boldsymbol{u^t}),\ \boldsymbol{v} \rangle$                // away vertex
4:      $s^t \leftarrow \underset{\boldsymbol{v} \in \boldsymbol{S^t}}{\text{argmax}} \langle \nabla\Phi(\boldsymbol{u^t}),\ \boldsymbol{v} \rangle$                // local FW
5:      $w^t \leftarrow \text{argmax}\, \nabla\Phi(\boldsymbol{u^t})$                // global FW
6:      if $\langle \nabla\Phi(\boldsymbol{u^t}), \boldsymbol{a^t} - \boldsymbol{s^t} \rangle \geq \langle \nabla\Phi(\boldsymbol{u^t}), \boldsymbol{u^t} - \boldsymbol{w^t} \rangle$ then
7:      $\boldsymbol{d^t} = \boldsymbol{a^t} - \boldsymbol{s^t}$
8:      $\alpha_{max} \leftarrow \boldsymbol{u^t}[a_t]$
9:      $\alpha^t \leftarrow \underset{\alpha \in [0, \alpha_{max}]}{\text{argmax}}\ \Phi(\boldsymbol{u^t} - \alpha \boldsymbol{d^t})$
10:     if $\alpha^t < \alpha_{max}$ then
11:         $\boldsymbol{S^t} \leftarrow \boldsymbol{S^{t-1}}$                // descent step
12:     else
13:         $\boldsymbol{S^t} \leftarrow \boldsymbol{S^{t-1}} \setminus \{\boldsymbol{a^t}\}$                // drop step
14:     end if
15:     else
16:         $\boldsymbol{d^t} = \boldsymbol{u^t} - \boldsymbol{w^t}$
17:         $\alpha^t \leftarrow \underset{\alpha \in [0,1]}{\text{argmin}}\, \Phi(\boldsymbol{u^t} - \alpha \boldsymbol{d^t})$
18:         $\boldsymbol{S^t} \leftarrow \boldsymbol{S^{t-1}} \cup \{\boldsymbol{w^t}\}$ (or $\boldsymbol{S^t} \leftarrow \{\boldsymbol{w^t}\}$ if $\alpha^t = 1$)                // FW step
19:     end if
20:     $\boldsymbol{u^t} \leftarrow \boldsymbol{u^{t-1}} - \alpha^t \boldsymbol{d^t}$
21: end for
```

In Algorithm 2, if the local pairwise gap $\langle \nabla\Phi(\boldsymbol{u_t}),\ \boldsymbol{a^t} - \boldsymbol{s^t} \rangle$ is smaller than the Frank-Wolfe gap $\langle \nabla\Phi(\boldsymbol{u^t}),\ \boldsymbol{u^t} - \boldsymbol{w^t} \rangle$, a FW step (line 16-18) is taken. Otherwise, the weights of the active atoms in $\boldsymbol{S^t}$ are locally optimized by the Pairwise Conditional Gradients (PCG). If the step size α^t is larger than α_{max}, the away vertex is removed from the active set $\boldsymbol{S^t}$, and we refer to this step as a drop step (line 13). Otherwise, it is called a descent step (line 11). Collectively, the descent step and drop step are referred to as pairwise steps [5].

Due to the structure of the BPCG algorithm, the sparsity of the solutions is expected since the new atoms are not added to $\boldsymbol{S^t}$ until the local pairwise gap decreases sufficiently. Additionally, since the PCG is implemented locally, BPCG does not exhibit swap steps in which weight shifts from the away atom to the Frank-Wolfe atom [5].

Note that in Algorithm 2, we use a line search for simplicity in presentation. However, it can alternatively be executed with a simpler decaying strategy, such as $\frac{t}{t+2}$.

Advantages of the BPCG algorithm compared to other CG (FW) variants:

- **Compared to the vanilla CG**: The BPCG algorithm ensures linear convergence for the strongly convex polyhedral case, while the vanilla CG does not. Although both algorithms guarantee a $O\left(\frac{1}{k}\right)$ convergence for infinite-dimensional cases, we expect the BPCG algorithm to produce much sparser solutions due to its aggressive removal of unnecessary vertices from the convex combination and the addition of new vertices only when needed for primal progress [5].
- **Compared to the Away CG algorithm**: BPCG and Away CG have the same convergence guarantee for the general smooth convex case and strongly convex polyhedral case. However, BPCG empirically generates significantly sparser solutions [5].
- **Compared to the Pairwise CG algorithm**: BPCG's convergence rate aligns more closely with that of the Away-steps Frank-Wolfe algorithm than with PCG. Moreover, since the iterations of BPCG include many local updates in which no new atoms are added, it is expected that the BPCG algorithm outputs sparser solutions than the PCG algorithm in terms of the support size of the supporting convex combination [5].

BPCG also extends to the infinite-dimensional setting, and its (empirical) sparsity is very high, making it a prime candidate for problems such as sparse signal recovery, matrix completion, and numerical integration [5].

$(\mathbf{1}+\boldsymbol{\varepsilon})$-approximation to MEB

Given a finite set of points $\mathcal{A} = \{a_1, \ldots, a_m\} \subset \mathbb{R}^n$, our goal is to compute an approximation to the minimum enclosing ball of $\mathcal{A}$, denoted by MEB($\mathcal{A}$). Given a given ε > 0, we define a ball $\boldsymbol{\mathcal{B}_{c,r}}$ to be a $(1+\varepsilon)$-approximation to MEB($\mathcal{A}$) if [6]:

$$\mathcal{A} \subset \boldsymbol{\mathcal{B}_{c,r}}, \qquad r \le (1+\varepsilon)r^* \tag{36}$$

A subset $\boldsymbol{S} \subseteq \mathcal{A}$ is said to be an ε-core set (or a core set) of $\mathcal{A}$ if [6]:

$$r_{\boldsymbol{S}} \le \; r^* \le (1+\varepsilon)r_{\boldsymbol{S}} \tag{37}$$

Small core sets play an important role in designing efficient algorithms for large-scale problems because they provide a compact representation of the input set $\mathcal{A}$. If a small ε-core set $\boldsymbol{S}$ is available, then solving the problem on $\boldsymbol{S}$ already yields a good approximation to MEB($\mathcal{A}$). Since the center c^* of MEB($\mathcal{A}$) lies within the convex hull of $\mathcal{A}$, Carathéodory's theorem guarantees the existence of a 0-core set with a size of at most $n+1$ [6].

Algorithm 3 is closely related to the Frank–Wolfe algorithm with proper initialization, applied to the dual formulation of the minimum enclosing ball problem. In each iteration, the algorithm can only add points to the working core set. This algorithm computes a $(1+\varepsilon)$-approximation to MEB($\mathcal{A}$) in $O\left(\frac{mn}{\varepsilon}\right)$ operations and converges in $O\left(\frac{1}{\varepsilon}\right)$ iterations, matching the currently best-known dependence on epsilon. In addition, it explicitly computes a 'core set' of size $O\left(\frac{1}{\varepsilon}\right)$, which is independent of both m and n [6].

Algorithm 3 constructs a sequence of balls with strictly increasing radii in each iteration, and it updates both the radius and the error bound at each iteration. In each iteration, the center is shifted towards the furthest point from the center of the current ball, but the center's movement is limited to only a fraction of this distance. Also, a termination criterion is checked in each iteration, which has the potential advantage of earlier termination than that what is predicted by the theoretical worst-case estimate [6].

We will now provide a more detailed description of Algorithm 3. In step 1, the algorithm calculates the furthest point $\boldsymbol{a_p} \in \boldsymbol{\mathcal{A}}$ from $\boldsymbol{a_1} \in \boldsymbol{\mathcal{A}}$ and then computes the furthest point $\boldsymbol{a_q} \in \boldsymbol{\mathcal{A}}$ from $\boldsymbol{a_p}$. Steps 2 and 3 initialize the vector $\boldsymbol{u^0} \in \mathbb{R}^m$. It's worth noting that $\boldsymbol{u^0}$ is a feasible solution of the dual problem Φ. The core set $\boldsymbol{S^0}$ is initialized at step 4. In each iteration, the algorithm implicitly constructs a 'trial' ball with center $\boldsymbol{c^t}$ and radius $\sqrt{\gamma^t}$. This ball coincides with MEB($\boldsymbol{\mathcal{A}}$) if and only if $\boldsymbol{u^t}$ is an optimal solution of Φ. Otherwise, it means that at least one point in $\boldsymbol{\mathcal{A}}$ lies outside of this ball. Here, δ^t is such that $\|\boldsymbol{a_\kappa} - \boldsymbol{c^t}\|^2 = (1+\delta^t)\gamma^t$, where $\boldsymbol{a_\kappa} \in \boldsymbol{\mathcal{A}}$ is the furthest point from $\boldsymbol{c^t}$. Consequently, the trial ball encloses $\boldsymbol{\mathcal{A}}$ if its radius is expanded by a factor of $\sqrt{(1+\delta^t)}$, i.e., $\boldsymbol{\Phi}(\boldsymbol{u^t}) \leq \boldsymbol{\Phi}(\boldsymbol{u}^*) \leq (\mathbf{1}+\delta^t)\boldsymbol{\Phi}(\boldsymbol{u^t})$. Unless the termination criterion is satisfied, the new center $\boldsymbol{c^{t+1}}$ is computed by moving $\boldsymbol{c^t}$ toward the furthest point $\boldsymbol{a_\kappa}$, which is then added to the working core set $\boldsymbol{S^{t+1}}$. $\boldsymbol{u^{t+1}}$ is updated accordingly to ensure that dual feasibility is maintained. The algorithm continues iteratively, computing a new trial ball corresponding to $\boldsymbol{u^{t+1}}$ [6].

Algorithm 3: $(\mathbf{1}+\boldsymbol{\varepsilon})$-approximation to MEB

```
$p \leftarrow \underset{i=1,\dots,n}{\operatorname{argmax}} \|u_i - u_1\|^2$ , $q \leftarrow \underset{i=1,\dots,n}{\operatorname{argmax}} \|u_i - u_p\|^2$
$\boldsymbol{u^0} \leftarrow \mathbf{0}$
$u_p^0 \leftarrow \frac{1}{2}, u_q^0 \leftarrow \frac{1}{2}$
$\boldsymbol{S^0} \leftarrow \{\boldsymbol{a_p}, \boldsymbol{a_q}\}$
$\boldsymbol{c^0} \leftarrow \sum_{\boldsymbol{i=1}}^{\boldsymbol{n}} u_i^0 \boldsymbol{a_i} = \langle \boldsymbol{u^0}, \boldsymbol{a} \rangle$
$\gamma^0 \leftarrow \boldsymbol{\Phi}(\boldsymbol{u^0})$
$\kappa \leftarrow \underset{i=1,\dots,n}{\operatorname{argmax}} \|\boldsymbol{a_i} - \boldsymbol{c^0}\|^2$
$\delta^0 \leftarrow (\|\boldsymbol{a_\kappa} - \boldsymbol{c^0}\|^2/\gamma^0) - 1$
$t \leftarrow 0$
While $\delta^t > (1+\varepsilon)^2 - 1$ and $t < maxIter$ do
loop
    $\alpha^t \leftarrow \delta^t/[2(1+\delta^t)]$
    $t \leftarrow t+1$
    $\boldsymbol{u^t} \leftarrow (1-\alpha^{t-1})\boldsymbol{u^{t-1}} + \alpha^{t-1}\boldsymbol{e}_\kappa$
    $\boldsymbol{c^t} \leftarrow (1-\alpha^{t-1})\boldsymbol{c^{t-1}} + \alpha^{t-1}\boldsymbol{a}_\kappa$
    $\boldsymbol{S^t} \leftarrow \boldsymbol{S^t} \cup \{\boldsymbol{a_\kappa}\}$
    $\gamma^t \leftarrow \boldsymbol{\Phi}(\boldsymbol{u^t})$
    $\kappa \leftarrow \underset{i=1,\dots,n}{\operatorname{argmax}} \|\boldsymbol{a_i} - \boldsymbol{c^t}\|^2$
    $\delta^t \leftarrow (\|\boldsymbol{a_\kappa} - \boldsymbol{c^t}\|^2/\gamma^t) - 1$
end loop
```

At each iteration of Algorithm 3, the quadratic objective function $\boldsymbol{\Phi}(\boldsymbol{u})$ is linearized at the current feasible solution $\boldsymbol{u^t}$. Since the feasible region of the dual problem is the unit simplex, the unit vector $\boldsymbol{e}_\kappa$, where κ is the index of the furthest point in $\boldsymbol{\mathcal{A}}$ from $\boldsymbol{c^t}$, solves the linearized subproblem. It is easy to verify that [6]:

$$\alpha^t = \underset{\alpha \in [0,1]}{\operatorname{argmax}} \, \Phi\big((1-\alpha)\boldsymbol{u^t} + \alpha \boldsymbol{e_\kappa}\big) \tag{38}$$

We remark that Algorithm 3 uses only the first-order approximation to the objective function Φ. As such, each iteration is fairly cheap, but the number of iterations is usually significantly higher than other algorithms that use second-order information such as interior-point methods. However, such general-purpose algorithms become computationally infeasible for larger problems, since each iteration is usually much more expensive [6].

Lemma 3.1. $\boldsymbol{u^t} \in \mathbb{R}^n$ satisfies $\gamma^t \leftarrow \boldsymbol{\Phi}(\boldsymbol{u^t}) \geq \frac{1}{3}\boldsymbol{\Phi}(\boldsymbol{u}^*) = \frac{1}{3}\gamma^*$, where $\boldsymbol{u^t}$ and γ^* are the optimal solution and the optimal value of the dual problem, respectively. Furthermore, $\delta^0 \leq 8$ [6].

Lemma 3.2: For each $t = 0,1, \dots$ the following relationship is satisfied [6]:

$$\gamma^t = \gamma^{t-1}\left(1 + \frac{(\delta^{k-1})^2}{4(1+\delta^{k-1})}\right) \tag{39}$$

The main idea behind Algorithm 3 is to approximate the given input set using only a carefully selected finite subset of points and then to refine this approximation by adding more points if necessary. This leads to an approximation of the primal formulation with only a finite number of constraints, and this approximation is refined by adding more constraints. In the dual formulation, we therefore start with a finite number of variables and add more variables if necessary [6].

The initial working core set $\boldsymbol{S^0}$ provides the first approximation to the given input set with only two points. Let $\boldsymbol{\Phi}(\boldsymbol{u^0})$ denote the objective function of the dual formulation of the minimum enclosing ball problem for $\boldsymbol{S^0}$, and let γ^* denote the optimal value of the aforementioned semi-infinite primal formulation. Similarly, let $\boldsymbol{\Phi}(\boldsymbol{u^t})$ denote the objective function of the dual formulation of the minimum enclosing ball problem for $\boldsymbol{S} \subseteq \mathcal{A}$. At iteration t in each algorithm, $\boldsymbol{S^t}$ provides the current finite approximation to $\mathcal{A}$. Let $\boldsymbol{c^t} \in \mathbb{R}^n$ denote the current center. Algorithm 3 computes the furthest point in $\mathcal{A}$ from $\boldsymbol{c^t}$. In Algorithm 3, $\boldsymbol{S^t}$ is obtained by adding this point to $\boldsymbol{S^{t-1}}$. Unless the furthest point in $\mathcal{A}$ already belongs to $\boldsymbol{S^{t-1}}$, the dual formulation for $\boldsymbol{S^t}$ differs from that for $\boldsymbol{S^{t-1}}$ in only one variable. Therefore, $[(\boldsymbol{u^t})^{\boldsymbol{T}}, \boldsymbol{0}]^{\boldsymbol{T}}$ is a feasible solution for the new dual formulation that satisfies $\boldsymbol{\Phi}([(\boldsymbol{u^t})^{\boldsymbol{T}}, \boldsymbol{0}]^{\boldsymbol{T}}) = \boldsymbol{\Phi}(\boldsymbol{u^t}),$ which implies that the improvement in each iteration still obeys the relation given by Lemma 3.2, with γ^t replaced by $\boldsymbol{\Phi}(\boldsymbol{u^t})$ and γ^{t-1} by $\boldsymbol{\Phi}(\boldsymbol{u^{t-1}})$. Note that the dimension of $\boldsymbol{u^t}$ is one more than that of uk in this case [6].

Line search strategies

Algorithms 1 and 2 need to perform a line search in each iteration t to find a suitable learning rate α^t that is within the range $[0, \alpha_{max}]$. There are several options available for performing this line search.

Inverse time decay learning rate

In the initial stages of our work, we began with a relatively straightforward strategy. We employed a basic formula to establish a diminishing learning rate, one that relied solely on the current iteration - represented as $\frac{t}{t+2}$. While this approach did yield satisfactory results in terms of generating reasonable parameters (center and radius) for the MEB and kept CPU time to a minimum, both algorithms struggled to converge.

Increasing the number of iterations would lead to a significant increase in CPU time, which was against our objective of being efficient. This emphasized the necessity for a more sophisticated and refined strategy.

Golden section search

The golden section line search is a technique for finding the minimum of a unimodal function within a specified interval, such as $(0, \alpha_{max}]$ in our case. The method works by iteratively shrinking the search interval while ensuring that the ratio of the smaller subinterval to the larger subinterval remains the golden ratio, approximately 1.618. By doing this, it narrows down the search space efficiently, converging towards the optimal solution.

The golden section line search is relatively simple and relies solely on (dual) function evaluations. Our experiments demonstrated its high efficiency when used in conjunction with Algorithm 1. It consistently converged well before reaching the maximum iteration limit (1000). However, in combination with Algorithm 2, the performance was poor. It failed to converge, and the returned radius was much smaller than the optimal value.

Armijo’s rule

We proceeded by implementing an inexact backtracking line search, also known as Armijo’s rule. In this case, the learning rate is determined through an iterative search process until a sufficient decrease of the objective function is achieved. We fixed the parameters as follows: $\delta = 0.5, \gamma = 0.1$, and initiated with a large starting step-size $\Delta = \alpha_{max} \in (0,1)$. We then iteratively (and geometrically) decreased α by multiplying its current value by δ, represented as $\alpha = \delta^t \Delta$, where $t = 1, 2, 3, \ldots$ until inequality [7]:

$$\Phi(\boldsymbol{u} - \alpha \nabla \Phi(\boldsymbol{u})) \leq \Phi(\boldsymbol{u}) - \gamma\alpha \|\nabla \Phi(\boldsymbol{u})\|_2^2 \quad (40)$$

is satisfied, and then we return $\alpha = \alpha^t$.

Even though the obtained results for the center and radius using Armijo's rule were close to the optimal ones, it was evident that this line search strategy is very slow, and often failed to converge. This issue can be attributed to the problem of ill-conditioning. A high condition number typically leads to slow convergence, and given the randomness of data sampling, such a scenario was likely to occur.

An illustrative example of this situation arises when data points are nearly collinear or closely aligned on the same hyperplane. In such cases, it becomes challenging for the algorithm to identify a unique and well-defined minimum enclosing ball. For instance, in a 2D context, one feature might have small values, while another feature may exhibit significantly larger values.

Given that we had no control over the ill-conditioning, which depended entirely on the feature values of the data points in the datasets, and considering our intention to evaluate the algorithms on various benchmark datasets, we made the decision to proceed with the best option: an exact line search.

Exact line search

In this approach, instead of relying on iterative approximations like in some other line search methods (e.g., Armijo's rule or golden section search), the exact value of the learning rate that would guide us to the minimum can be computed analytically, as the objective function is quadratic and convex.

When examining the minimization version of our dual function, i.e., $-\Phi(\boldsymbol{u}) = \boldsymbol{u^T A^T A u} - \boldsymbol{u^T z}$, we can represent it in the form of $f(\boldsymbol{x}) = \frac{1}{2}\boldsymbol{x^T Q x} + \boldsymbol{c^T x}$, where $\boldsymbol{Q}$ is a symmetric matrix, by substituting: $\boldsymbol{Q} = 2\boldsymbol{A^T A}$ and $\boldsymbol{c} = -\boldsymbol{z}$ [7].

Suppose that we have selected a step direction $\boldsymbol{d^t}$. In this case, to find the exact step-size, we need to minimize $\Phi(\boldsymbol{u^t} + \alpha \boldsymbol{d^t})$, which can be expressed as [8]:

$$\varphi(\alpha) = \Phi(\boldsymbol{u^t} + \alpha \boldsymbol{d^t}) = \frac{1}{2}(\boldsymbol{u^t} + \alpha \boldsymbol{d^t})^T \boldsymbol{Q}(\boldsymbol{u^t} + \boldsymbol{\alpha d^t}) + \boldsymbol{c}^T(\boldsymbol{u^t} + \alpha \boldsymbol{d^t}) \tag{41}$$

This is a quadratic function of α, and thus we can compute the optimal step-size by finding the α, such that $\varphi'(\alpha) = 0$. By expanding out the quadratic term, it is easy to show that [8]:

$$\varphi'(\alpha) = \alpha \boldsymbol{d^{t^T} Q d^t} + \boldsymbol{d^{t^T} Q u^t} + \boldsymbol{c^T d^t} \tag{42}$$

Setting this equal to zero and solving for α yields the step-size [8]:

$$\alpha = -\frac{\boldsymbol{d^{t^T} Q u^t} + \ \boldsymbol{c d^{t^T}}}{\boldsymbol{d^{t^T} Q d^t}} = -\frac{\boldsymbol{d^{t^T}}(\boldsymbol{Q u^t} + \ \boldsymbol{c})}{\boldsymbol{d^{t^T} Q d^t}} = -\frac{\nabla\Phi(\boldsymbol{u^t})^T \boldsymbol{d^t}}{\boldsymbol{d^{t^T} Q d^t}} \tag{43}$$

or equivalently:

$$\alpha = -\frac{\nabla\Phi(\boldsymbol{u^t})^T \boldsymbol{d^t}}{2\boldsymbol{d^{t^T} A^T A d^t}} \tag{44}$$

where the gradient at $\boldsymbol{u^t}$ is $\nabla\Phi(\boldsymbol{u^t}) = \ 2\boldsymbol{A^T A u} - \boldsymbol{z}$, or equivalently $\nabla\Phi(\boldsymbol{u^t}) = \ \boldsymbol{Q u^t} + \boldsymbol{c}$.

Experiments

The experiments aim to assess the quality of three given algorithms on both synthetic and real-world datasets. The experimental section consists of two parts. The first part involves training and testing the algorithms on synthetic datasets. In the second part, we apply a similar approach to real-world datasets, also incorporating epsilon (ε) as a variable hyperparameter. All experimental results were obtained using a Windows workstation with a 2.4GHz Intel i5-1135G7 CPU and 8GB DDR4 3200MHz Memory. The algorithms were implemented in Python 3.10.

I. Random datasets

In this section, we evaluate the quality of the MEB returned by all three algorithms, train and test the algorithms using artificially created data and conduct a comparative analysis of the results.

1. Uniform distribution

This initial experiment aimed to assess the quality of the MEB returned by the three algorithms. To achieve this, we created two closely spaced yet separable clusters: one for training with 8000 data points sampled from U[0.0, 0.7), and the other for testing (containing anomalies) with 2000 points sampled from U[0.7, 1.0).

In order to improve visualization and comprehension of the results, we began with only two features. This allowed us to display the data in 2D alongside the generated circle (using the center and radius obtained), as depicted in Figure 3. With this configuration, all three algorithms achieved perfect accuracy. This was expected since the datasets were sampled using different ranges, ensuring cluster separability.

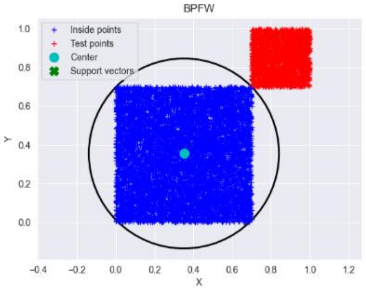

Figure 3: The blue cluster represents nominal points, while the red one represents anomaly points. The circle shows the MEB constructed by Algorithm 2.

We then proceeded to increase the number of features. For every dimension we explored, we consistently obtained perfect results. Table 1 presents our findings for a dataset with an arbitrarily chosen dimension of 15, while Figure 4 displays various plots that comprehensively depict the training process (the construction of the MEB).

	Training					**Test**		
	8000 MEB samples from U[0,7)	**Iterations**	**CPU time (s)**	**Radius**	**Active set size**	*2000 MEB samples from U[0.7,1)*	**Recall (%)**	**F1 score (%)**
ASFW		379	23.632	1.019375	15		100	100
BPFW		272	16.606	1.019365	14		100	100
(1+ε)-approx.		747	22.108	1.019697	18		100	100

Table 1: Computational results for the three Algorithms on an artificial dataset – Uniform distribution

The results in Table 1 and Figure 4 highlight that Algorithm 2 (BPFW) stands out, requiring the fewest iterations (272) and the least total CPU time to converge. This implies that BPFW achieves the most significant progress with each iteration. Conversely, Algorithm 1 (ASFW) requires fewer iterations than Algorithm 3 (APPFW), but slightly more CPU time to reach convergence.

As the primal error is incalculable due to the unknown optimal solution (c*, r*), we rely on the duality gap as a valuable measure of optimality (stopping criterion) obtained effortlessly from our algorithms. The second row of graphs illustrates how the dual gap, represented as delta, diminishes with each iteration and CPU time. It is evident that none of the algorithms exhibit linear convergence. Upon closer examination, we find that BPFW exhibits the best convergence rate, followed by ASFW, and lastly, APPFW.

Moving to the last row of plots, we notice that BPFW also outperforms the other two algorithms in terms of the size of the active set, denoted as $\boldsymbol{S^t}$. This is particularly advantageous, especially for large datasets, where storing the active set can be resource-intensive. When the line (function) decreases by one, it signifies the removal of an atom from the active set, indicating a drop step by the algorithm. While we cannot guarantee substantial progress in such cases, fortunately, they occur infrequently. ASFW experienced 8 drop steps out of 379 iterations, while BPFW had 10 out of 239 iterations. Notably, APPFW doesn't involve drop steps, as evident in the plots, where the green line consistently increases with each iteration. In Algorithm 3, when a new center is computed, the current center shifts towards the furthest point, which is subsequently added to the working core set.

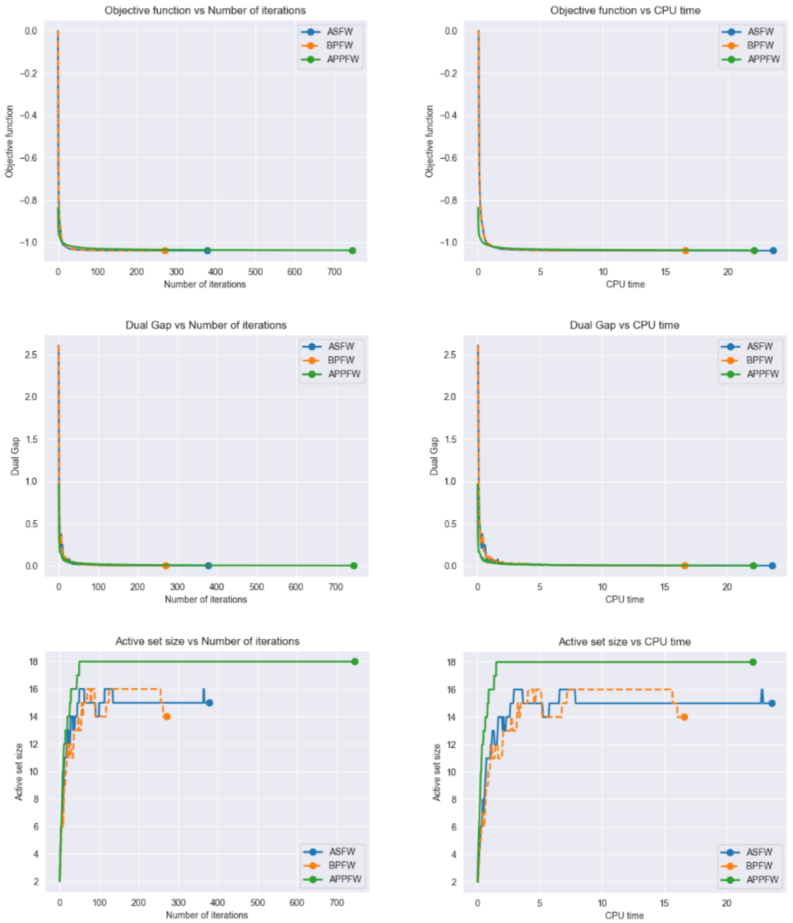

Figure 4: Results obtained from training (the construction of the minimum enclosing ball) using a training set consisting of 8000 data points with a dimension of 15. These data points were drawn from U[0, 0.7). The plots on the left depict how the objective function, dual gap, and size of the active set change over each iteration. Meanwhile, the plots on the right showcase the corresponding changes in CPU time. Each algorithm is represented by a distinct color on the graphs.

2. Gaussian distribution

Next, another synthetic dataset was generated, consisting of 8000 samples for training and 2000 samples for testing, with each sample having 10 features. The training dataset was created by randomly sampling from normal distribution N(0,1). For the testing dataset, half of the samples were drawn from the same Gaussian distribution as the training dataset, and these are referred to as normal data. The other half of the testing dataset was generated by sampling from a different Gaussian distribution N(7,1) deliberately chosen to create well-separated clusters. These samples are referred to as novel or anomaly data.

Same as above, we initially conducted the experiment using only 2 features to enhance visual comprehension. The outcomes for this setup are presented in Figure 5. Following that, we performed training and testing using the 10-feature dataset. The training results per iteration and per CPU time are presented in the Figure 6 and Table 2.

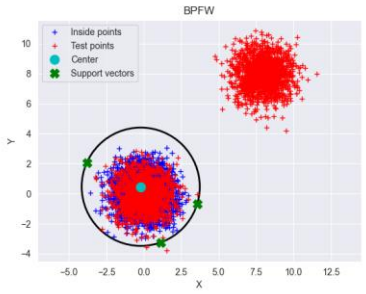

Figure 5: The blue cluster represents training points, while the red one represents test points. The circle shows the MEB constructed by Algorithm 2. Support vectors are training points that lie on the boundary (MEB). The red points around the center of the MEB are test points belonging to the nominal class, while the red cluster on the top right represents the anomaly points in the test set.

Both the graphs and the table reveal that APPFW takes the most iterations to converge and consumes the most CPU time. This can be attributed to the high epsilon value used in this experiment (0.001). To test this claim, we reduced the epsilon value to 0.1, yielding the following results: 0.201s and 7 iterations for APPFW, 2.884s and 51 iterations for BPFW, and 3.454s and 62 iterations for ASFW. This confirms our claim and suggests that the epsilon value allows us to strike a balance between correctness and speed. The APPFW algorithm, being an approximation algorithm, can be highly efficient in scenarios where a very fast solution is needed, even at the cost of a slightly increased error. Increasing the epsilon value decreased the CPU time for BPFW and ASFW by approximately a factor of 3, whereas for APPFW, this decrease is significantly higher (100 times faster).

The analysis of the dual gap/delta value per iteration yields the same finding as before: BPFW exhibits the best convergence rate, indicating the highest progress per iteration.

The active set per iteration/CPU time for ASFW is almost identical to BPFW. For this experiment, both of these algorithms take just two drop steps. In contrast, the third algorithm adds atoms to the active set up until iteration 80, after which the size of the active set (core set) remains constant at 12.

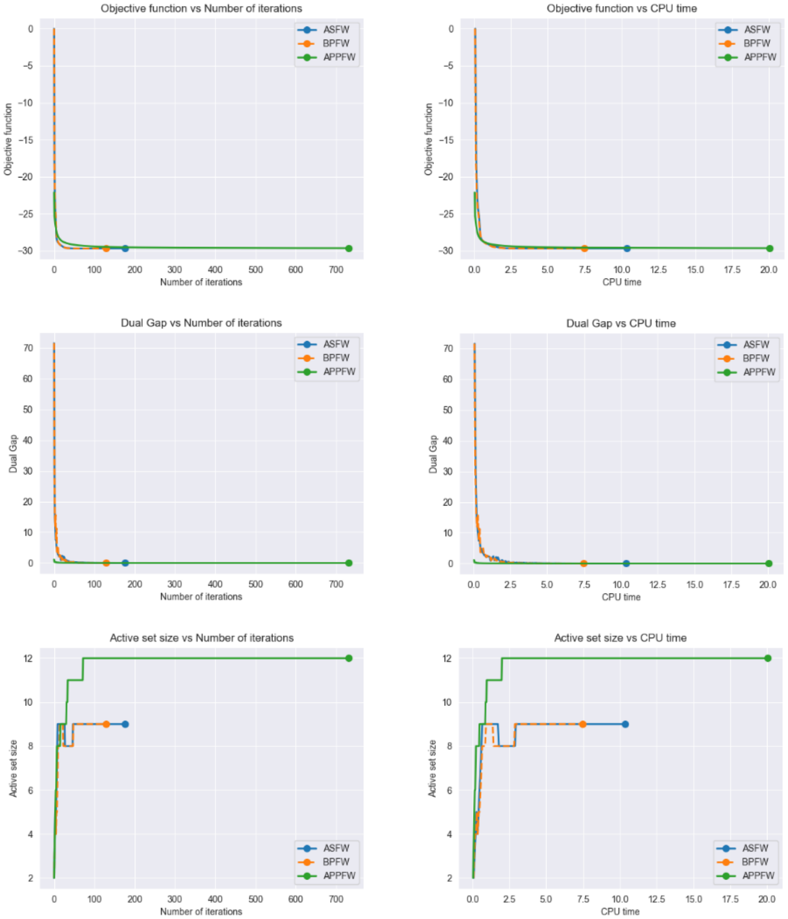

Figure 6: Results obtained from training (the construction of the MEB) using a training set consisting of 8000 data points with a dimension of 10. These data points were drawn from N(0,1). The plots on the left depict how the objective function, dual gap, and size of the active set change over each iteration. Meanwhile, the plots on the right showcase the corresponding changes in CPU time. Each algorithm is represented by a distinct color on the graphs.

Table 2 provides a summary of the results from training and testing with the three different algorithms can be seen in the table above. As observed from the table, in the second experiment, the F1 score is not perfect. We conducted an investigation to understand why this could happen, and here we present our findings.

	Training					Testing		
	8000 MEB samples from N(0,1)	**Iterations**	**CPU Time (s)**	**Radius**	**Active set size**	*1000 MEB samples from N(0,1) + 1000 anomalies from N(7,1)*	**Recall (%)**	**F1 Score (%)**
ASFW		177	9.614	5.45032	9		100	99.85
BPFW		130	7.219	5.45032	9		100	99.85
(1+ε)-approx.		733	21.904	5.45191	12		100	99.85

Table 2: Computational results for the three Algorithms on an artificial dataset – Gaussian distribution

The test dataset contains 2000 points: 1000 anomalies and 1000 nominal. All anomaly points are correctly classified as anomalies (true positives). We can gain a better understanding of these anomalies by referring to Figure 5, where they are represented by the red cluster in the top right of each plot.

However, there are instances of false positives: 3 out of 1000 points that were intended to be classified as nominal have been mistakenly classified as anomalies. These points can also be observed in Figure 5 as the red points positioned below the MEB. The reason for these false positives is that some nominal points fall just slightly outside the MEB. We do not consider this to be a significant issue.

One potential solution could be to classify as anomalies only those points that are farther than δ*r from the center, where δ is a constant greater than 1, and r represents the radius of the MEB. However, we have chosen not to pursue this approach because we anticipated that real-world datasets may exhibit overlapping characteristics, and selecting an appropriate value for δ would require additional fine-tuning.

II. Benchmarking on real-world datasets

In this section, we evaluate all three algorithms using two distinct real-world datasets and conduct a comparative analysis of the results. We also provide comprehensive descriptions of the datasets and explain the preprocessing procedures applied. Different ε (epsilon) values were used for each experiment to vary the tightness of the minimum enclosing ball.

Preprocessing

For both datasets, as described in the sections below, we initially ensured the data's validity. After a brief exploratory data analysis (EDA), it became evident that feature scaling was necessary due to the features being on different scales. This step is essential to prevent one feature from dominating the radius computation when we are dealing with Euclidean distances. We considered two options for scaling: standard scaling and min-max scaling. After experimentation, we opted for standard scaling since it yielded better results.

During the preprocessing phase, we converted binary labels from strings to numeric values (0 for nominal data, 1 for anomaly data) and removed irrelevant features. Subsequently, the data was divided into two subsets: one containing normal cases (*nominal_features* and *nominal_labels*) and the other anomaly cases (*anomaly_features* and *anomaly_labels*). Importantly, a portion of the nominal data was allocated for training, while the remaining nominal data and all anomaly data comprised the test set. Finally, the data underwent standardization for feature scaling. To fit the scaler, we exclusively used the training data. Subsequently, we separately transformed the training and test data using this scaler, ensuring that we used the mean and standard deviation from the training set's portion of the nominal data.

As only the training data is used by the algorithms, they will only learn the distribution of the normal data, including the center and the enclosing radius. In the end, we hope that this will enable us to differentiate between normal and anomaly data.

1. Breast Cancer Wisconsin Dataset

The “Breast Cancer Wisconsin (Diagnostic)” [9], sourced from Kaggle, provides a comprehensive collection of information derived from digitized images of fine needle aspirates (FNA) of breast masses. These characteristics and features describe the cell nuclei found within the images, with the primary aim of distinguishing between malignant and benign cases of breast cancer. This dataset consists of 569 samples and 32 features.

The dataset encompasses the following attributes [9]:

- *ID number*: A unique identifier for each data point.
- *Radius*: The mean distance from the center of the nucleus to points on its perimeter.
- *Texture*: The standard deviation of gray-scale values within the nucleus.
- *Perimeter*: The perimeter of the nucleus.
- *Area*: The area enclosed by the nucleus.
- *Smoothness*: A measure of local variation in the lengths of the nucleus radii.
- *Compactness*: A derived measure, computed as (perimeter^2 / area - 1.0).
- *Concavity*: A measure of the severity of concave portions within the contour of the nucleus.
- *Concave points*: The count of concave portions present in the contour of the nucleus.
- *Symmetry*: A measure of symmetry in the nucleus.
- *Fractal dimension*: A measure based on the coastline approximation," reflecting the complexity of the contour.
- *Diagnosis*: The outcome of interest, categorized into two classes: 'M' for malignant and 'B' for benign.

Our model included all features except the ID number, which was omitted because IDs should not be used in classification models. For all the other features, three statistics are provided: the mean, standard error, and "worst" or largest (mean of the three largest values) [9]. Notably, this dataset does not contain any missing values, ensuring data completeness. In terms of class distribution, it comprises 357 benign cases and 212 malignant cases, making it a valuable resource for developing predictive models to aid in the diagnosis of breast cancer and use it for anomaly detection as we are doing.

To create training and testing datasets, we first separated the nominal data from the anomaly data. Half of the nominal data was used for training (178 samples), and the other half for testing. The anomaly data was solely used for testing. Therefore, the testing dataset consisted of 391 samples (179 nominal and 212 anomalies). All samples were 33-dimensional. Various thresholds (ε) were employed as stopping criteria, and these are reported in Table 3.

	Training					Testing			
	ε	Iterations	CPU Time (in ms)	Radius	Active set size	FN	FP	Recall (%)	F1 Score (%)
ASFW	0.1	64	26.38	11.077366	6	50	5	76.415	85.488
	0.01	92	36.97	11.077386	6	50	5	76.415	85.488
	0.001	118	98.634	11.077386	6	50	5	76.415	85.488
BPFW	0.1	32	18.165	11.077378	6	50	5	76.415	85.488
	0.01	50	24.112	11.077386	6	50	5	76.415	85.488
	0.001	66	31.402	11.077386	6	50	5	76.415	85.488
(1+ε)-approx.	0.1	4	2.004	11.758557	5	59	3	72.17	83.152
	0.01	35	11.0	11.130932	7	50	5	76.415	85.488
	0.001	608	146.278	11.080393	7	50	5	76.415	85.488

Table 3: Computational results for the three Algorithms on the Breast Cancer Wisconsin dataset

On the right side of Table 3, the test results are presented. It is evident that varying the ε value had minimal impact on the outcomes. The only noticeable deviation occurred with ε=0.1 for the third algorithm. However, the decrease in the F1 score is negligible, and therefore, it can be safely disregarded.

The testing results lead us to the conclusion that the classes in the Breast Cancer dataset are well separated, and even a straightforward approach like this one performs well. With this in mind, we aimed to find a contrasting scenario - a dataset in which the classes overlap - to gain a different perspective on the suitability of this approach for more complex situations.

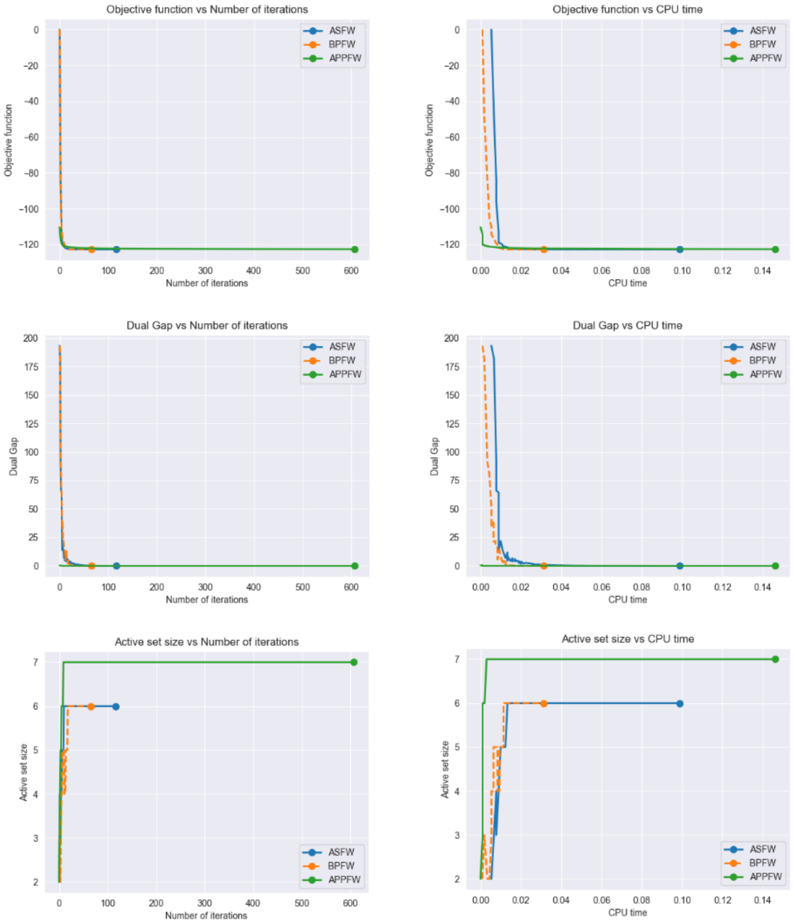

Figure 7: Results obtained from training set consisting of 50% of the nominal data points of the Breast Cancer Wisconsin Dataset. The plots on the left depict how the objective function, dual gap, and size of the active set change over each iteration. Meanwhile, the plots on the right showcase the corresponding changes in CPU time. Each algorithm is represented by a distinct color on the graphs.

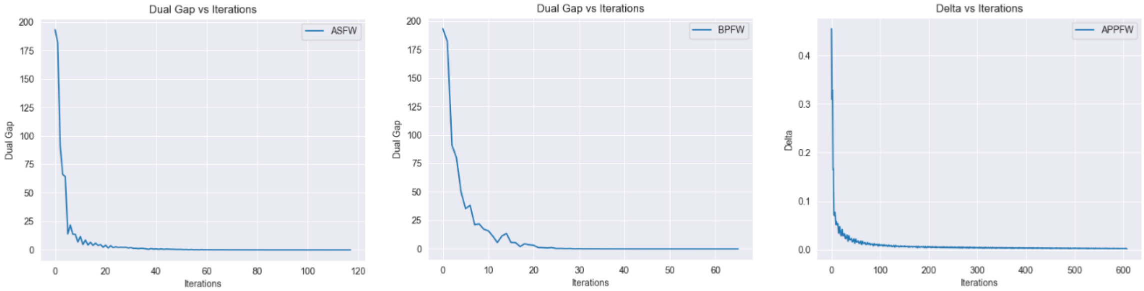

Figure 8: Dual gap / Delta per iteration for all three algorithms during training on Breast Cancer Wisconsin Dataset

The graphs presented in Figure 8 display the results obtained with ε=0.001. It's evident that for achieving this desired level of correctness in the MEB, the BPFW algorithm converges the quickest, both in terms of iterations and CPU time. In contrast, ASFW requires significantly fewer iterations compared to APPFW (118 vs. 608) and is also notably faster in terms of CPU time (60% faster).

The active set size graphs may appear crowded, but they generally display an increasing trend, indicating a low number of drop steps. A more detailed analysis confirms this observation, revealing only one drop step for ASFW and two for BPFW. This implies that the majority of iterations exert a substantial influence. However, a considerable number of away/descent steps are taken, as indicated by the zig-zagging pattern in the dual gap per iteration/CPU time plots.

The value of ε has the most significant impact on the third algorithm. Higher values result in a drastic reduction in CPU time, decreasing from 146ms to 2ms when epsilon is increased from 0.001 to 0.1. The number of iterations also decreases significantly, from 608 to 4. However, as a consequence, there is a slight drop in F1 score and recall. In this case, the trade-off appears favorable, as the decrease in testing metrics is minimal, while the decrease during training is substantial.

2. Iranian Churn Dataset

The "Iranian Churn Dataset" from the UCI Machine Learning Repository [10] is a collection of data randomly gathered from a telecommunications company in Iran, spanning a period of 12 months. This dataset contains a total of 3,150 samples, with each sample representing a unique customer. The information is organized into 14 features, each providing insights into customer behavior and characteristics.

The key attributes included in this dataset are as follows [10]:

- *Anonymous customer ID*: A unique identifier assigned to each customer, ensuring data privacy.
- *Call failures*: This attribute indicates the number of call failures experienced by each customer.
- *Complaints*: A binary attribute with values 0 (no complaint) and 1 (complaint), denoting whether a customer registered a complaint.
- *Subscription length*: The total number of months for which a customer has maintained a subscription.
- *Charge amount*: An ordinal attribute representing the charge amount, with values ranging from 0 (lowest amount) to 9 (highest amount).
- *Seconds of use*: The total number of seconds a customer has spent making calls.
- *Frequency of use*: This attribute reflects the total number of calls made by a customer.
- *Frequency of SMS*: The total number of text messages sent by a customer.
- *Distinct called numbers*: The total count of distinct phone numbers a customer has called.
- *Age group*: An ordinal attribute categorizing customers into age groups, with values ranging from 1 (younger age) to 5 (older age).
- *Tariff Plan*: A binary attribute indicating the type of service plan a customer is on, with values 1 (pay as you go) and 2 (contractual).
- *Status*: Another binary attribute that indicates whether a customer is currently active (1) or non-active (2).
- *Customer value*: This attribute represents a calculated value that likely encapsulates a customer's overall worth or contribution to the telecom company.
- *Churn*: This binary attribute serves as the class label, distinguishing between customers who have churned (1) and those who have not (0) at the end of the 12-month period. Churn signifies customers who have discontinued their services.

To create training and testing datasets, we first separated the nominal data and the anomaly data. Half of the nominal data was used for training (1327 samples), and the other half for testing. On the other hand, the anomaly data was used just for testing. Therefore the testing dataset comprised of 1823 samples (1327 nominal and 495 anomalies). All of the samples were 14-dimensional. Varying thresholds were used for the stopping criterion, and all of these are reported in the Table 4.

	Training					Testing			
	ε	Iterations	CPU time (in ms)	Radius	Active set size	FN	FP	Recall (%)	F1 Score (%)
ASFW	0.1	55	131.633	6.708818	6	445	2	10.101	18.282
	0.01	80	154.788	6.709065	6	447	2	9.697	17.615
	0.001	100	217.184	6.709067	6	447	2	9.697	17.615
BPFW	0.1	32	74.47	6.709004	6	448	3	9.495	17.248
	0.01	40	97.306	6.709067	6	447	2	9.697	17.615
	0.001	54	116.559	6.709067	6	447	2	9.697	17.615
(1+ε)-approx.	0.1	3	5.293	7.184624	4	491	1	0.808	1.6
	0.01	13	14.184	6.753838	6	454	1	8.283	15.27
	0.001	178	178.887	6.711496	7	447	2	9.697	17.615

Table 4: Computational results for the three Algorithms on Iranian Churn dataset

On the right side of Table 4, we can observe the testing results. It's clear that altering the ε value generally had a minimal impact on the overall outcome. The only notable deviation occurred when ε was set to 0.1 for the third algorithm. The significant decrease in the F1 score at ε=0.1 suggests that the data samples are closely distributed to each other. This conclusion comes from the fact that changing the epsilon from 0.01 to 0.1 for the third algorithm results in a 7% reduction in the radius, resulting in an F1 score decrease of 10 times. The testing results lead us to the conclusion that the classes in the Iranian Churn dataset are not well separated, indicating the need for tighter boundaries for radius and epsilon to improve the F1 score.

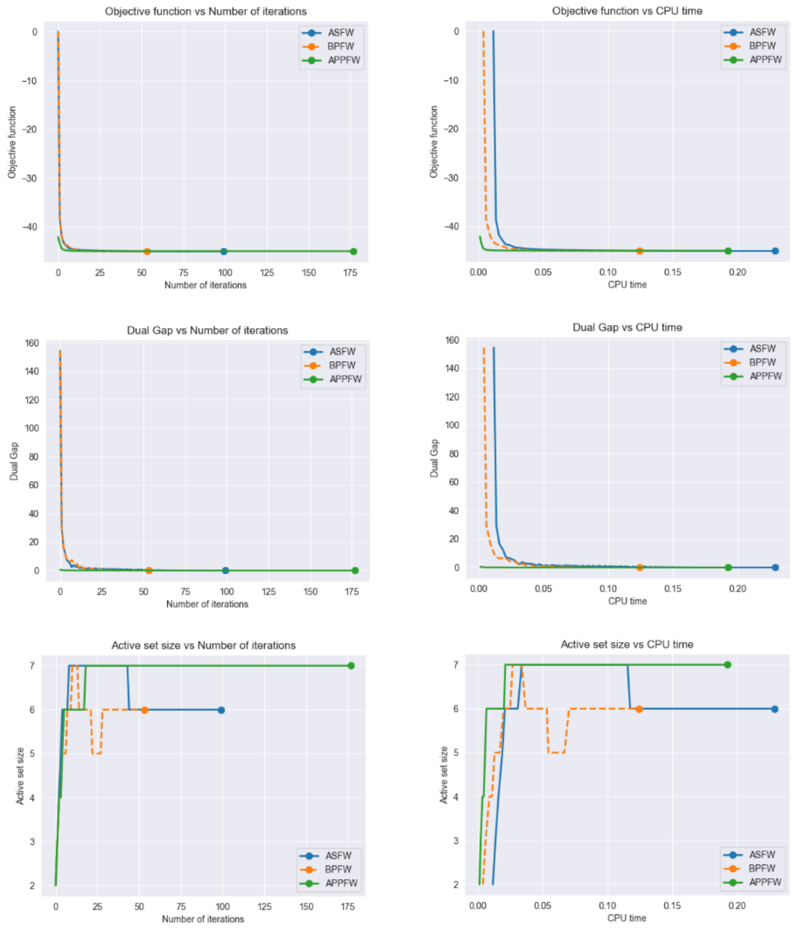

Figure 9: Results obtained from training set consisting of 50% of the nominal data points. The plots on the left depict how the objective function, dual gap, and size of the active set change over each iteration. Meanwhile, the plots on the right showcase the corresponding changes in CPU time. Each algorithm is represented by a distinct color on the graphs.

The graphs in Figure 10 display the results obtained using ε=0.001. It's apparent that for achieving this desired level of correctness in the MEB, the BPFW algorithm converges the quickest, both in terms of iterations and CPU time. On the other hand, ASFW has fewer iterations and less CPU time compared to APPFW, similar to the Breast Cancer dataset.

The value of epsilon has the most significant impact on the third algorithm. Higher values result in a drastic reduction in CPU time, decreasing from 178ms to 5ms when epsilon is increased from 0.001 to 0.1. The number of iterations also decreases significantly, going from 178 to 3. In this case, the trade-off is less favorable because, although training is much faster, the decrease in testing metrics is substantial. Additionally, despite the third algorithm having the highest number of iterations, it takes less time per iteration since it employs a naive approximation rather than solving a linear minimization oracle.

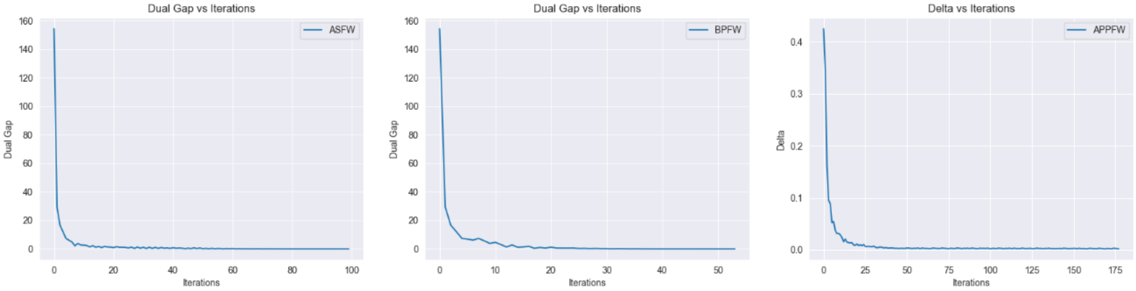

Figure 10: Dual gap / Delta per iteration for all three algorithms during training on Breast Cancer Wisconsin Dataset

Conclusion

Our project focused on implementing three adaptations of the Frank Wolfe algorithm: Away-steps Frank Wolfe, Blended Pairwise Conditional Gradient, and (1+ε)-approximation to MEB. Through our experimentation and analysis, we discovered that BPCG consistently outperformed the other two algorithms across various metrics, including a smaller number of iterations, CPU time, and active set size. More importantly, it exhibited a better convergence rate, even though it was still sublinear.

Algorithm 2, i.e., Away-steps FW, produced satisfactory results in all scenarios. Even though it was not the best-performing of the three algorithms we tested, it again demonstrated that it is a significant improvement over the vanilla FW.

On the other hand, Algorithm 3, (1+ε)-approximation to MEB, usually underperformed when using a strict bound for the stopping criterion (0.001). In such cases, its CPU time, number of iterations, and size of the active set were higher compared to the other two algorithms. However, when using a larger ε, its performance improved drastically. When using a value of 0.1 for ε, it consistently showed dominant performance, beating the other two algorithms by a factor of 10 in terms of CPU time and number of iterations. Thus, ε allows us to make a trade-off between performance and correctness.

Our evaluation on the Breast Cancer Dataset, where we achieved an F1 score of 85%, demonstrates that all three algorithms performed admirably. This high F1 score suggests that the dataset contained well-separable data points, and the algorithms effectively exploited this separability. On the contrary, our results from testing on the Iranian Churn Dataset indicated a different scenario. With an F1 score of only around 17%, it became evident that the classes in this dataset were not well separated.

From the results, it can be seen that the algorithms' performance is highly dependent on the structure of the dataset in use. This outcome emphasizes the importance of understanding the underlying data distribution and characteristics when choosing optimization algorithms, as the effectiveness of these methods can vary significantly based on the nature of the data.

References

[1] C. a. D. T. Bauckhage, Lecture Notes on Machine Learning: Minimum Enclosing Balls, 2019.

[2] C. Bauckhage, NumPy / SciPy Recipes for Data Science: Frank-Wolfe for Minimum Enclosing Balls, 2020.

[3] C. a. D. T. Bauckhage, Lecture Notes on Machine Learning: Frank-Wolfe for Minimum Enclosing Balls, 2019.

[4] S. L.-J. a. M. Jaggi, "On the Global Linear Convergence of Frank-Wolfe Optimization Variants," 2015.

[5] K. T. S. P. Kazuma Tsuji, "Pairwise Conditional Gradients without Swap Steps and Sparser Kernel Herding," 2022.

[6] E. A. Yildirim, "Two Algorithms for the Minimum Enclosing Ball Problem," *SIAM Journal on Optimization,* vol. 19, pp. 1368-1391, 01 2008.

[7] B. Jiang, "John Hopcroft Center for Computer Science," 9 November 2020. [Online]. Available: https://jhc.sjtu.edu.cn/public/courses/CS257/2020/slides/lec10.pdf.

[8] M. A. Davenport, "Line search methods and convergence of gradient descent revisited," Fall 2021. [Online]. Available: https://mdav.ece.gatech.edu/ece-3803-fall2021/notes/11-notes-3803-f21.pdf.

[9] K. L. Clarkson, "Coresets, sparse greedy approximation, and the Frank-Wolfe algorithm," *ACM Transactions on Algorithms,* vol. Article No.: 63, pp. 1-30, 3 September 2010.

[10] B. Gabor, S. Pokutta, T. Dan and S. Wr, "Blended Conditional Gradients: The Unconditioning of Conditional Gradients," 2019.

[11] Kaggle, "Breast Cancer Wisconsin (Diagnostic) Data Set," [Online]. Available: https://www.kaggle.com/datasets/uciml/breast-cancer-wisconsin-data.

[12] U. M. L. Repository, "Iranian Churn Dataset," 2020. [Online]. Available: https://archive.ics.uci.edu/dataset/563/iranian+churn+dataset.