**MEB and Anomaly Detection using variants of Frank-Wolfe**

# 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, symbolic learning, identification of landmarks or prototypes, analysis of data streams, and anomaly detection. (**TODO**: ADD HERE THAT IN THIS PROJECT WE FOCUS ON THE LAST PART). MEBs are also at the heart of one class support vector machines, support vector clustering, or support vector data description. 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.

Minimum enclosing balls have numerous important applications in 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 large-scale instances of the MEB problem, and a reasonably small accuracy suffices for such applications.

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 basic Frank-Wolfe algorithm and its variations. We will discuss this in detail in this project. (**TODO**: change last sentence)

Notation

We use bold letters to denote vectors and matrices, while non-bold letters represent constants. For a vector , 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.

bold – vector/matrix

– set of points ( – number of points, – dimension of each point)

( - i-th point of the set of points (with dimension )

– ball with center (with dimension = n) and radius (constant)

is a vector of Lagrange multipliers

– Lagrangian dual function

# Problem definition

## The primal MEB problem

In the minimum enclosing ball problem, given a finite set of vectors , the objective is to determine the smallest n-dimensional ball that contains all the points in :

Since Euclidean balls are uniquely defined in terms of their center and radius , the basic problem is therefore to determine the optimal choices and for these two parameters. In this report, we approach this problem from the point of view of constrained convex optimization. Indeed, noting the equivalences:

we can immediately formalize the problem of finding and as an inequality constrained quadratic minimization problem, namely

Which we can rewrite as:

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.

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

By setting , we obtain:

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

where is a vector of Lagrange multipliers.

## Deriving the dual problem

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

Notice that, when expanding the expression in (TODO) to the one in (TODO), we introduced the shorthand just to reduce notational clutter.

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

where is a data matrix whose columns correspond to the data points in . Hence, the Lagrangian in (7) can also be written as:

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

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

Furthermore, by plugging the expressions from (TODO) and (TODO) back into the Lagrangian in (TODO), we obtain the Lagrangian dual, since:

is a function that depends solely on the Lagrange multipliers .

Now, if we further note that the KKT 3 conditions (dual feasibility) require , all our (intermediate) results and considerations so far indicate that the dual of (TODO) amounts to

Finally, it's worth noting that is concave in , making is convex. Therefore, the solution to the maximization problem in (TODO) can also be found by considering an equivalent minimization problem, namely:

## From Lagrange multipliers to MEB parameters

If we could solve the dual problem in (TODO), we would obtain an optimal Lagrange multiplier vector, denoted as . Assuming we could determine , it would immediately enable us to compute the center of the sought-after MEB, as per equation (TODO):

If we could solve the dual problem in (TODO) we would obtain a vector of optimal Lagrange multipliers. Assuming we could determine , it would immediately enable allow us to compute the center of the sought-after MEB because, according to (TODO), we have:

But what about the corresponding radius ?

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

Summing these equations over all i then yields

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

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

# Frank-Wolfe for Minimum enclosing balls

Reiterating what we discussed above, we emphasize that the dual MEB problem in (TODO) seeks a minimizer of the convex function (TODO). Additionally, the non-negativity () and sum-to-one () constraints in (TODO) imply that any feasibly solution must reside within the standard simplex:

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

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 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 (TODO equation above).

The Frank-Wolfe algorithm is an iterative solver. First, it makes an initial feasible guess for the solution. The midpoint of is contained within and is therefore feasible. This means that we can use 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 that minimizes the inner product and applies a conditional gradient update , where the step size [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.

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

Therefore, each Frank-Wolfe iteration has to solve:

Note that the expression on the right-hand side is linear in and needs to be minimized over , 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 correspond to the standard basis vectors of , we only need to determine which of these minimizes the inner product mentioned above. Hence, the problem in (TODO) simplifies to:

To make it explicit that the resulting will be a standard basis vector , we can also express it as:

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:

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

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.

When the optimal solution lies on the boundary of , 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).

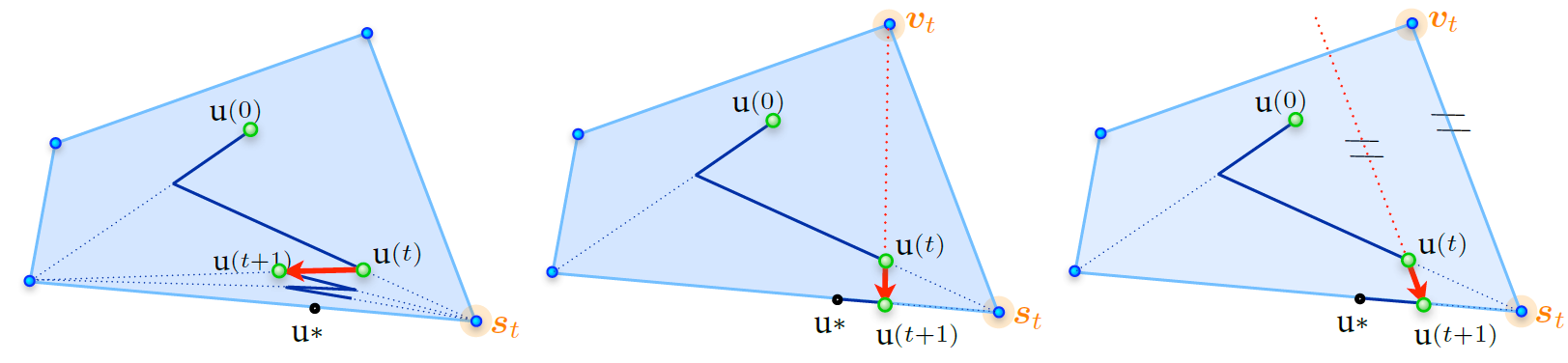


Figure 1: (left) The FW algorithm zig-zags when the solution lies on the boundary. (middle) Adding the possibility of an away step mitigates this problem. (right) As an alternative, a pairwise FW step.

To address the zig-zagging issue in the FW algorithm, Wolfe proposed adding the possibility of moving away from an active atom in (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 **,** is defined in line 4 by finding the atom in that maximizes the potential descent given, by . It's important to note that this search is performed over the (typically small) active set , making it fundamentally easier than the linear oracle . The maximum step-size ,defined in line 9, ensures that the new iterate remains within conv() ⊆ , 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 along a specific line, which is generally possible for arbitrary . 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 , unlike the standard FW algorithm. Finally, as in classical FW, the FW gap serves as an upper bound on the unknown suboptimality and can be used as a stopping criterion:

(by convexity)

If , then we refer to this step as a drop step, as it completely removes the atom from the currently active set of atoms, , (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).

The weight updates for lines 12 and 13 take the following form: For a FW step, we have if ; otherwise . Additionally, we have and for . For an away step, we have if (a drop step); otherwise .Also, we have and for .

**Algorithm 1: Away-steps Frank-Wolfe algorithm**

1. ,  **//** *so that for* *and 0 otherwise*
2. **For** **do**
3. and // *the FW direction*
4. and // *the away direction*
5. **if** **then** **return**
6. **if** **then**
7. and // *choose the FW direction*
8. **else**
9. and // *choose away direction; maximum feasible step-size*
10. **end if**
11. Line-search:
12. **//** *and accordingly for the weights*
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.

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.

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

In Algorithm 2, if the local pairwise gap is smaller than the Frank-Wolfe gap , a FW step (line 16-18) is taken. Otherwise, the weights of the active atoms in are locally optimized by the Pairwise Conditional Gradients (PCG). If the step size is larger than , the away vertex is removed from the active set , 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.

Due to the structure of the BPCG algorithm, the sparsity of the solutions is expected since the new atoms are not added to 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.

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 .

**Algorithm 2: Blended Pairwise Conditional Gradients (BPCG)**

1. ,
2. **for**  **do**:
3. // *away vertex*
4. // *local FW*
5. // *global FW*
6. **if** **then**


10. **if** **then**
11. // *descent step*
12. **else**
13. // *drop step*
14. **end if**
15. **else**

18. (or **if )** // *FW step*
19. **end if**
21. **end for**

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 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.
* **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.
* **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.

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.

**-approximation to MEB**

Given a finite set of points , our goal is to compute an approximation to the minimum enclosing ball of , denoted by MEB(). Given a given ε > 0, we define a ball to be a -approximation to MEB() if:

A subset is said to be an -core set (or a core set) of if:

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 . If a small -core set is available, then solving the problem on already yields a good approximation to MEB(). Since the center of MEB() lies within the convex hull of , Carathéodory’s theorem guarantees the existence of a 0-core set with a size of at most .

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 -approximation to MEB() in operations and converges in iterations, matching the currently best-known dependence on epsilon. In addition, it explicitly computes a ‘core set’ of size , which is independent of both and .

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.

**Algorithm 3: *-*approximation to MEB**

1. **While** and **do**
2. **loop**







11. **end loop**

We will now provide a more detailed description of Algorithm 3. In step 1, the algorithm calculates the furthest point from and then computes the furthest point from . Steps 2 and 3 initialize the vector . It’s worth noting that is a feasible solution of the dual problem (TODO). The core set is initialized at step 4. In each iteration, the algorithm implicitly constructs a ‘trial’ ball with center and radius . This ball coincides with MEB() if and only if is an optimal solution of (TODO). Otherwise, it means that at least one point in lies outside of this ball. Here, is such that , where is the furthest point from . Consequently, the trial ball encloses if its radius is expanded by a factor of , i.e., . Unless the termination criterion is satisfied, the new center is computed by moving toward the furthest point , which is then added to the working core set . is updated accordingly to ensure that dual feasibility is maintained. The algorithm continues iteratively, computing a new trial ball corresponding to .

At each iteration of Algorithm 3, the quadratic objective function of (TODO) is linearized at the current feasible solution . Since the feasible region of (TODO) is the unit simplex, the unit vector , where is the index of the furthest point in from , solves the linearized subproblem. It is easy to verify that:

We remark that Algorithm 3 uses only the first-order approximation to the objective function . 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.

Lemma 3.1. satisfies , whereand are the optimal solution and the optimal value of (TODO), respectively. Furthermore, .

Lemma 3.2: For each the following relationship is satisfied:

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.

The initial working core set provides the first approximation to the given input set with only two points. Let denote the objective function of the dual formulation of the minimum enclosing ball problem for and let denote the optimal value of the aforementioned semi-infinite primal formulation.

Similarly, let denote the objective function of the dual formulation of the minimum enclosing ball problem for . At iteration t in each algorithm, provides the current finite approximation to . Let denote the current center. Algorithm 3 computes the furthest point in from . In Algorithm 3, is obtained by adding this point to . Unless the furthest point in already belongs to , the dual formulation for differs from that for in only one variable. Therefore, is a feasible solution for the new dual formulation that satisfies which implies that the improvement in each iteration still obeys the relation given by Lemma 3.2, with replaced by and by . Note that the dimension of is one more than that of uk in this case.

Testing

Refrences:

[(PDF) Lecture Notes on Machine Learning: Minimum Enclosing Balls (researchgate.net)](https://www.researchgate.net/publication/331152293_Lecture_Notes_on_Machine_Learning_Minimum_Enclosing_Balls)

[(PDF) Lecture Notes on Machine Learning: Frank-Wolfe for Minimum Enclosing Balls (researchgate.net)](https://www.researchgate.net/publication/331315423_Lecture_Notes_on_Machine_Learning_Frank-Wolfe_for_Minimum_Enclosing_Balls)

[(PDF) NumPy / SciPy Recipes for Data Science: Frank-Wolfe for Minimum Enclosing Balls (researchgate.net)](https://www.researchgate.net/publication/344138310_NumPy_SciPy_Recipes_for_Data_Science_Frank-Wolfe_for_Minimum_Enclosing_Balls)