Contents

1. Classification with SVM

Assume we are given a finite number, say $m \in \mathbb{N}$, of point pairs, $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, where $i \in [m] = \{1, \ldots, m\}$ and $\mathcal{Y} = \{-1, +1\}$. Our objective is to find a *maximum margin hyperplane* in a Hilbert space, $\mathcal{H}$, that separates the features $\{\Phi(\mathbf{x}_i) : y_i = -1\}$ and $\{\Phi(\mathbf{x}_i) : y_i = +1\}$, where $\Phi : \mathcal{X} \to \mathcal{H}$ is a feature mapping. The benefit of working in a feature space, $\mathcal{H}$, is that it allows us to capture $\mathbf{x}$-y relationships that cannot be modelled using a linear classifier in the input space, $\mathcal{X}$.

Hard-Margin Classifiers

A traditional SVM formulation, called hard-margin SVM, assumes that the points are linearly separable in the feature space, and aims to find an error-free classifier that maximises the "margin" (distance) of the classification hyperplane/boundary from the nearest points in each class (see Appendix A.2 for a formal treatment). Intuitively, this can be seen as finding a classifier robust to observation noise (Smola et al., 2000, Section 1.1.3). Theoretically, a maximum margin classifier minimizes the PAC bound on the generalization error (Shawe-Taylor et al., 1998, Theorem 4.17; Herbrich and Graepel, 2000, Theorem 3).

Soft-Margin Classifiers

Linear separability is, in practice, a strong assumption, since it is hard to *a-priori* guarantee that in a high dimensional feature space. Moreover, it is also not always favorable to find an error-free classifier since this can lead to over-fitting – observations are usually noisy and an error-free classifier can end up fitting to the noise in the data, leading to poor generalization Smola et al. (2000). To tackle this, regularization techniques balance the maximum margin objective with a penalty incurred on the mistakes made by the classifier on the training dataset. This leads to a soft-margin SVM formulation, which can tackle the case of a linearly inseparable dataset – a situation hard-margin SVMs are not designed for. In Appendix A.3, we present an L_1-SVM. As the name suggests, this removes the error-free classification constraint as in hard-margin SVM, and adds an L_1-regularization to the objective for penalizing misclassifications.

Why an L_1 Penalty?

One might ask why we choose to penalise the objective using an L_1 penalty, instead of, say, an L_p penalty, where we replace ξ_i in the objective with ξ_i^p. There are two reasons for that:

1. **Quadratic programming (QP) problem**: With $p = 1$, the primal problem is particularly convenient – it can be formulated as a QP for which we have dedicated optimization routines.

2. **Robustness guarantees**: With $p = 1$, our estimator enjoys robustness to outliers (Scholkopf and Smola, 2001, Proposition 7.7).

1.1. The L_2-SVM Optimization Problem

Several works (Lee and Mangasarian, 2001a,b; Mangasarian and Musicant, 2000) have formulated the SVM problem with L_2 regularization, i.e. $p = 2$. This comes at the cost of losing robustness guarantees we have for L_1-SVM. However, in practice, L_1-SVM and L_2-SVM have similar classification accuracy (Lee and Mangasarian, 2001b; Tsang et al., 2005). On the plus side, the L_2-SVM formulation opens up a suite of algorithms that can speed up the optimization process. Particularly, as we will discuss in Section 2, L_2-SVM are equivalent to the Minimal Enclosing Ball (MEB) problem (Tsang et al., 2005), for which we can find a $(1+\epsilon)$-approximate solution in $\mathcal{O}\left(1/\epsilon\right)$ steps (Bădoiu and Clarkson, 2008).

1.1.1. Primal Problem

The L_2-SVM was proposed by Lee and Mangasarian (2001b):

$$\begin{aligned} \min_{\mathbf{w},b,\boldsymbol{\xi},\rho} \quad & \frac{1}{2}\left(\|\mathbf{w}\|_{\mathcal{H}}^2 + b^2 + C\sum_{i\in[m]}\xi_i^2\right) - \rho \\ \text{subject to} \quad & y_i\left(\left\langle\Phi\left(\mathbf{x}_i\right),\mathbf{w}\right\rangle_{\mathcal{H}} + b\right) \geq \rho - \xi_i, \quad \forall i \in [m] \end{aligned} \tag{1}$$

Comparing this primal problem with the L_1-SVM primal (36), we note three things:

- The objective includes b, which induces strong convexity with little to no effect to the problem (Mangasarian and Musicant, 1999, Proposition 2.1).
- We have introduced squared slack variables, which removes the need for the non-negativity constraint (Mangasarian and Musicant, 2001, Equation 7). This can be interpreted as follows: since the objective is an even function of ξ^*, an optimum $\xi_i^* < 0$ simultaneously implies another feasible optimum $\xi^* > 0$. Hence, the constraint $\xi^* > 0$ is implicit.
- Finally, the dependence on the margin width is made explicit through ρ.

Furthermore, we make the following observations about the problem:

- The objective function is convex in $\mathbf{w}$, b, $\boldsymbol{\xi}$ and ρ, and its domain $\mathcal{D} = \mathcal{H} \times \mathbb{R} \times \mathbb{R}^m \times \mathbb{R}$ is convex.
- The constraints $\rho - \xi_i - y_i\left(\left\langle\Phi\left(\mathbf{x}_i\right),\mathbf{w}\right\rangle_{\mathcal{H}} + b\right) \leq 0$ are linear in $\mathbf{w}$, b, $\boldsymbol{\xi}$ and ρ.

This tells us that we have a convex problem. Since the constraints are affine, Slater's conditions are satisfied trivially and we conclude that strong duality holds.

1.1.2. Dual Problem

The Lagrangian of (1) is given by

$$L\left(\mathbf{w},b,\boldsymbol{\xi},\rho,\boldsymbol{\alpha}\right) = \frac{1}{2}\left(\|\mathbf{w}\|_{\mathcal{H}}^2 + b^2 + C\sum_{i\in[m]}\xi_i^2\right) - \rho + \sum_{i\in[m]}\alpha_i\left(\rho - \xi_i - y_i\left(\left\langle\Phi\left(\mathbf{x}_i\right),\mathbf{w}\right\rangle_{\mathcal{H}} + b\right)\right)$$

with dual variable constraints $\alpha_i \geq 0$, $\forall i \in [m]$. Our dual objective is given by

$$\begin{aligned} g(\boldsymbol{\alpha}) &= \inf_{\mathbf{w},b,\boldsymbol{\xi},\rho} L(\mathbf{w},b,\boldsymbol{\xi},\rho,\boldsymbol{\alpha}) \\ &= \inf_{\mathbf{w}} \left(\frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 - \sum_{i\in[m]} \alpha_i y_i \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} \right) + \inf_b \left(\frac{1}{2} b^2 - b \sum_{i\in[m]} \alpha_i y_i \right) \\ &\quad + \sum_{i\in[m]} \inf_{\xi_i} \left(\frac{C}{2} \xi_i^2 - \xi_i \alpha_i \right) + \inf_\rho \rho \left(\sum_{i\in[m]} \alpha_i - 1 \right) \end{aligned}$$

The first term is the unique minimum of a 1-strongly convex function, which we can derive by setting the gradient with respect to $\mathbf{w}$ to 0:

$$\frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 - \sum_{i\in[m]} \alpha_i y_i \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} \right) = \mathbf{w} - \sum_{i\in[m]} \alpha_i y_i \Phi(\mathbf{x}_i) = 0 \Longrightarrow \mathbf{w} = \sum_{i\in[m]} \alpha_i y_i \Phi(\mathbf{x}_i) \tag{2}$$

which gives

$$\inf_{\mathbf{w}} \left(\frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 - \sum_{i\in[m]} \alpha_i y_i \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} \right) = -\frac{1}{2} \left\| \sum_{i\in[m]} \alpha_i y_i \Phi(\mathbf{x}_i) \right\|^2$$

Similarly, for the second term,

$$\inf_b \left(\frac{1}{2} b^2 - b \sum_{i\in[m]} \alpha_i y_i \right) = -\frac{1}{2} \left(\sum_{i\in[m]} \alpha_i y_i \right)^2 \quad \text{at} \quad b^{\min} = \sum_{i\in[m]} \alpha_i y_i \tag{3}$$

and the third term,

$$\sum_{i\in[m]} \inf_{\xi_i} \left(\frac{C}{2} \xi_i^2 - \alpha_i \xi_i \right) = -\frac{1}{2C} \sum_{i\in[m]} \alpha_i^2 \quad \text{at} \quad \xi_i^{\min} = \frac{\alpha_i}{C}, \quad \forall i \in [m] \tag{4}$$

As for the final term, it takes the infimum of an affine functions of ρ. Hence, it is equal to $-\infty$ if the sum of Lagrange dual variables, α_i, is non-zero. In summary,

$$g(\boldsymbol{\alpha}) = \begin{cases} -\frac{1}{2} \left\| \sum_{i\in[m]} \alpha_i y_i \Phi(\mathbf{x}_i) \right\|^2 - \frac{1}{2} \left(\sum_{i\in[m]} \alpha_i y_i \right)^2 - \frac{1}{2C} \sum_{i\in[m]} \alpha_i^2, & \alpha_i \geq 0 \text{ and } \|\boldsymbol{\alpha}\|_1 = 1 \\ -\infty, & \text{otherwise} \end{cases}$$

This gives us our dual problem,

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\frac{1}{2} \sum_{i,j\in[m]} \alpha_i \alpha_j \left(y_i y_j (k(\mathbf{x}_i, \mathbf{x}_j) + 1) + \frac{\delta_{ij}}{C} \right) \\ \text{subject to} \quad & \sum_{i\in[m]} \alpha_i = 1 \;\text{ and }\; \alpha_i \geq 0, \; \forall i \in [m] \end{aligned}$$

where $k(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$ is the kernel induced by the feature map Φ, and δ is the Kronecker-delta function. For the purpose of optimization with respect to $\boldsymbol{\alpha}$, the factor of $1/2$ in the objective is irrelevant, so we drop it to get the following equivalent optimization problem:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & -\sum_{i,j\in[m]} \alpha_i\alpha_j \left(y_i y_j \left(k\left(\mathbf{x}_i, \mathbf{x}_j\right) + 1\right) + \frac{\delta_{ij}}{C} \right) \\ \text{subject to} \quad & \sum_{i\in[m]} \alpha_i = 1 \;\text{ and }\; \alpha_i \geq 0,\; \forall i \in [m] \end{aligned} \tag{5}$$

1.1.3. Recovering the Primal Optimum

Since Slater's conditions are satisfied, the KKT conditions are necessary and sufficient for optimality. From Equation 2, Equation 3 and Equation 4, at the optimum, we have

$$\begin{aligned} \mathbf{w}^* &= \sum_{i\in[m]} \alpha_i^* y_i \Phi\left(\mathbf{x}_i\right) \\ b^* &= \sum_{i\in[m]} \alpha_i y_i \\ \boldsymbol{\xi}_i^* &= \frac{\alpha_i^*}{C} \end{aligned} \tag{6}$$

Hence, we can solve the dual problem to find $\boldsymbol{\alpha}^*$, and then use Equation 6 to recover the primal optimum, $\mathbf{w}^*$, b^* and $\boldsymbol{\xi}^*$. Finally, the optimum margin width ρ^* can be recovered by noting that we have 0 duality gap.

Our corresponding classifier is given by

$$f\left(\cdot; w^*, b^*\right) = \operatorname{sign}\left(\left\langle w^*, \Phi\left(\cdot\right)\right\rangle + b^*\right) = \operatorname{sign}\left(\sum_{i\in[m]} \alpha_i^* y_i \left(k\left(\mathbf{x}_i, \cdot\right) + 1\right)\right) = f\left(\cdot; \boldsymbol{\alpha}^*\right) \tag{7}$$

This gives us two important insights:

1. **Dual optimum is sufficient for making predictions**: We don't need the primal optimum for defining our classifier at all. We can simply solve the dual problem (5) and use the dual optimum, $\boldsymbol{\alpha}^*$, to define $f\left(\cdot; \mathbf{w}^*; b^*\right)$. This is convenient since we can now use complex non-linear kernels with, possibly, infinite dimensional associated feature mappings.

2. **We can discard the non-SVs**: The classifier only depends on inputs $\mathbf{x}_i$ with corresponding dual variables $\alpha_i^* \neq 0$; we call these points the support vectors (SV) and they usually form a small subset of the all the points. Once the SVM is trained, we can discard points with $\alpha_i^* = 0$ (the non-SVs) since they don't contribute to the final classifier $f\left(\cdot; \boldsymbol{\alpha}^*\right)$. From complementary slackness, we know that these points are the ones that strictly satisfy the primal inequality constraints (1),

   $$y_i \left(\left\langle \Phi\left(\mathbf{x}_i\right), \mathbf{w} \right\rangle_{\mathcal{H}} + b \right) > \rho - \xi_i$$

   In other words, the points lying on the correct side of the hyperplane corresponding to y_i (see Figure 6a) can be discarded at inference time.

1.1.4. Discussion on the Hyperparameter C

Note that as we increase C, the penalty incurred on incorrect classifications increases. Hence, in the case of linear separability, for sufficiently large C, the soft-margin classifier reduces to a hard-margin classifier. In the case of linear inseparability, C balances under-fitting (not learning the data well enough) and over-fitting (fitting too close to the data, ignoring observation noise and outliers). Both extremes – $C \to 0$ and $C \to \infty$ – negatively effect generalization, which makes tuning C an important design choice. Another important consideration is that, practically, the convergence time scales exponentially for dramatically high values (> 10) of C even with good optimization routines (Sentelle et al., 2008, Figure 1).

1.2. To Solve the Primal or the Dual?

Since both the primal and the dual are convex problems, an important question that rises is which one we should solve. To answer this, we note the following:

1. **Faster learning**: The primal problem (1) optimizes over the weight vectors $\mathbf{w} \in \mathcal{H}$, the threshold $b \in \mathbb{R}$, the margin width $\rho \in \mathbb{R}$ and the slack variables $\boldsymbol{\xi} \in \mathbb{R}^m$. On the other hand, the dual optimizes over $\boldsymbol{\alpha} \in \mathbb{R}^m$. So, when the dimensionality of the feature space is much larger than the number of data points, solving the dual will be a lot faster.

2. **Kernel trick**: Resorting to optimizing the dual problem allows us to use the kernel trick, adding flexibility to our model – regardless of our choice of kernel, the dimensionality of the problem remains the same, depending only on the size of the dataset. This also allows us to use infinite dimensional feature mappings, that can model complex decision boundaries, without over-fitting (Frieß et al., 1998, Theorem 2.1).

3. **Faster inference**: If we solve the primal, our classifier will explicitly depend on the primal optimum $\mathbf{w}^*, b^*$:

$$f(\mathbf{x}; \mathbf{w}^*, b^*) = \operatorname{sign}\left(\langle \mathbf{w}^*, \Phi(\mathbf{x})\rangle_{\mathcal{H}} + b^*\right)$$

   Calculating the inner product can be prohibitively expensive if the feature mapping is high dimensional. This also eliminates the option of using infinite dimensional feature mappings. Solving the dual, on the other hand, allows the classifier to only implicitly depend on $\mathbf{w}^*$ (7) – we only need to compute the kernel (typically a constant time operation) between the input and the support vectors, making inference far more efficient.

1.3. Nature of the Optimization Problem

Note that (5) is clearly a QP – we have a quadratic obejctive in $\boldsymbol{\alpha}$ and linear constraints. This opens us up to a range of popular algorithms designed for solving QP; see Pang (1983) for a review. That being said, with a naive implementation, the complexity of this optimization problem is prohibitive for large scale problems with $\mathcal{O}(m^2)$ memory requirement and $\mathcal{O}(m^3)$ runtime complexity. Therefore, we are going to pay special attention to these asymptotic complexities when justifying our choice of optimization algorithm, ensuring that it is scalable with the number of data points, m.

2. Equivalence Between L_2-SVM and MEB Problem

In this section, we first introduce the Minimum Enclosing Ball (MEB) problem, and then show how the L_2-SVM dual Equation 5 is equivalent to the MEB dual under mild conditions on the kernel $\tilde{k}$ induced by the feature mapping $\tilde{\Phi}$. This equivalence was first established in Tsang et al. (2005) and then generalized in Tsang et al. (2006).

2.1. MEB Problem

As the name suggests, the problem deals with finding a closed ball, $\mathcal{B}(\mathbf{c}, r)$, of minimum radius that encloses a set of given points. Formally, consider a set of points $\{\tilde{\mathbf{x}}_i\}_{i\in[m]} \subset \tilde{\mathcal{X}}$ and a feature mapping $\tilde{\Phi} : \tilde{\mathcal{X}} \to \tilde{\mathcal{H}}$. The solution of the MEB problem is the closed ball $\mathcal{B}(\mathbf{c}^*, r^*)$ that is a solution to the following optimization problem:

$$\begin{aligned} \min_{\mathbf{c},\gamma} \quad & \gamma \\ \text{subject to} \quad & \left\| \tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c} \right\|^2 \le \gamma, \quad \forall i \in [m] \end{aligned} \tag{8}$$

where $\gamma = r^2$ and the constraint $\gamma \ge 0$ is implicit since norms are positive definite. The Lagrangian of this MEB problem is

$$L(\mathbf{c}, \gamma, \boldsymbol{\alpha}) = \gamma + \sum_{i\in[m]} \alpha_i \left(\left\| \tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c} \right\|^2 - \gamma \right)$$

where we have introduced the Lagrange multipliers $\boldsymbol{\alpha} \in \mathbb{R}_+^m = \left(\mathbb{R}_+\right)^m$. The dual problem is given by

$$\begin{aligned} g(\mathbf{c}, \gamma) &= \inf_{\mathbf{c},\gamma} L(\mathbf{c}, \gamma, \boldsymbol{\alpha}) \\ &= \inf_{\gamma} \gamma \left(1 - \sum_{i\in[m]} \alpha_i \right) + \inf_{\mathbf{c}} \sum_{i\in[m]} \alpha_i \left\| \tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c} \right\|^2 \end{aligned}$$

The first term is a linear function of γ – it diverges to $-\infty$ when $\|\boldsymbol{\alpha}\|_1 \neq 1$ and is 0 otherwise. The second term is convex quadratic function of $\mathbf{c}$ and can be minimized by setting the partial derivative to 0:

$$0 = \frac{\partial}{\partial \mathbf{c}} \sum_{i\in[m]} \alpha_i \left\| \tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c} \right\|^2 = \sum_{i\in[m]} 2\alpha_i \left(\tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c} \right) \implies \mathbf{c} = \frac{\sum_{i\in[m]} \alpha_i \tilde{\Phi}(\tilde{\mathbf{x}}_i)}{\sum_{i\in[m]} \alpha_i} \tag{9}$$

This gives us the Lagrange dual function (Yildirim, 2008, Section 2):

$$g(\boldsymbol{\alpha}) = \begin{cases} \sum_{i\in[m]} \alpha_i \tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_i) - \sum_{i,j\in[m]} \alpha_i \alpha_j \tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j), & \|\boldsymbol{\alpha}\|_1 = 1 \text{ and } \alpha_i \ge 0, \ \forall i \in [m] \\ -\infty, & \text{otherwise} \end{cases}$$

Hence, the dual problem is given as follows:

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & f(\boldsymbol{\alpha}) = \sum_{i\in[m]} \alpha_i \tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_i) - \sum_{i,j\in[m]} \alpha_i \alpha_j \tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) \\ \text{subject to} \quad & \sum_{i\in[m]} \alpha = 1 \text{ and } \alpha_i \geq 0, \ \forall i \in [m] \end{aligned} \tag{10}$$

This is a concave maximization problem with linear constraints. Hence, Slater's conditions are trivially satisfied and strong duality holds. Moreover, the KKT conditions are necessary and sufficient at the optimum $(\mathbf{c}^*, \gamma^*, \boldsymbol{\alpha}^*)$. Setting the derivative of the Lagrangian with respect to $\mathbf{c}$ to 0, we can recover the primal optimum from the dual optimum $\boldsymbol{\alpha}^*$ using (9) with the dual constraints:

$$\mathbf{c}^* = \sum_{i\in[m]} \alpha_i^* \tilde{\Phi}(\tilde{\mathbf{x}}_i) \tag{11}$$

Finally, since the duality gap is 0, we can recover the radius of the MEB:

$$\gamma^* = f(\boldsymbol{\alpha}^*) = \sum_{i\in[m]} \alpha_i^* \tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_i) - \sum_{i,j\in[m]} \alpha_i^* \alpha_j^* \tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) \tag{12}$$

2.2. Normalizing Condition on the Kernel

Comparing the MEB dual (10) with L_2-SVM (5), we see that the two differ only in the linear term present in the former. In order to establish an equivalence between the two, we restrict our choice of kernel such that it satisfies the following normalization condition: $\tilde{k}(\tilde{\mathbf{x}}, \tilde{\mathbf{x}}) = \tilde{\Delta}^2$, $\forall \tilde{\mathbf{x}} \in \tilde{\mathcal{X}}$. That is, all the mappings lie on a sphere in the feature space $\tilde{\mathcal{H}}$. This condition is fairly unrestrictive, and is satisfied by several choice of kernels in practice (Tsang et al., 2005):

1. **Isotropic kernels**: Kernels of the form $k(\mathbf{x}_i, \mathbf{x}_j) = k'(\|\mathbf{x}_i - \mathbf{x}_j\|)$, e.g. the Gaussian kernel

2. **Dot product kernel**: Kernels of the form $k(\mathbf{x}_i, \mathbf{x}_j) = k'(\langle \mathbf{x}_i, \mathbf{x}_j \rangle)$, e.g. polynomial kernels, with normalized inputs

3. **Normalized kernels**: Kernels of the form $k(\mathbf{x}_i, \mathbf{x}_j) = \frac{k'(\mathbf{x}_i, \mathbf{x}_j)}{\sqrt{k'(\mathbf{x}_i, \mathbf{x}_i) k'(\mathbf{x}_j, \mathbf{x}_j)}}$

With the normalizing condition on the MEB kernel $\tilde{k}$, since $\sum_{i\in[m]} \alpha_i = 1$, we can drop the linear term in (10) and obtain an equivalence with (5) by setting $\tilde{\mathcal{X}} = \mathcal{X} \times \mathcal{Y}$ and

$$\tilde{k}((\mathbf{x}_i, y_i), (\mathbf{x}_j, y_j)) = y_i y_j k(\mathbf{x}_i, \mathbf{x}_j) + y_i y_j + \frac{\delta_{ij}}{C} \tag{13}$$

where k is the L_2-SVM kernel. We note two things here:

1. First, that $\tilde{k} : \tilde{\mathcal{X}} \times \tilde{\mathcal{X}} \to \mathbb{R}$ in this form is a kernel with the associated feature map $\Phi : \tilde{\mathcal{X}} \to \tilde{\mathcal{H}} = \mathcal{H} \times \mathbb{R} \times \mathbb{R}^m$ defined as

$$\tilde{\Phi}(\mathbf{x}, y) = \begin{bmatrix} y_i \Phi(\mathbf{x}) \\ y_i \\ \frac{1}{\sqrt{C}} \mathbf{e}_i \end{bmatrix}$$

where $\Phi : \tilde{\mathcal{X}} \to \mathcal{H}$ is the feature map associated with k and $\mathbf{e}_i \in \mathbb{R}^m$ is the i^{th} canonical basis vector of $\mathbb{R}^m$, i.e. $(\mathbf{e}_i)_j = \delta_{ij}$. Note here that as the (supervised) classification problem is turned into an (unsupervised) MEB problem, the label information y_i is encoded in the feature map $\tilde{\Phi}(\mathbf{x}_i, y_i)$.

2. Second, $\tilde{k}$ satisfies the normalization condition if k satisfies it with $k(\mathbf{x}, \mathbf{x}) = \Delta^2$, since

$$\tilde{\Delta}^2 = \tilde{k}(\mathbf{x}, \mathbf{x}) = k(\mathbf{x}, \mathbf{x}) + 1 + \frac{1}{C} = \Delta^2 + 1 + \frac{1}{C} > 0 \tag{14}$$

Tsang et al. (2005) established this equivalence between the L_2-SVM and the MEB problem to apply the core-set approach introduced in Bădoiu and Clarkson (2008) for the latter problem. Instead, we use a modified Frank-Wolfe (FW) algorithm to solve the MEB problem. This has previously been shown to be a lot faster than the core-set approach (Frandi et al., 2013, Table 2). Several speed-ups for FW-based algorithms have been proposed since then, e.g. Frandi et al. (2015), but we don't explore these accelerations in this project.

3. Frank-Wolfe Algorithm for the MEB-SVM Problem

The Frank-Wolfe (FW) algorithm was originally proposed in Frank and Wolfe (1956) as a method for tackling problems of the form

$$\max_{\boldsymbol{\alpha} \in \boldsymbol{\Sigma}} f(\boldsymbol{\alpha})$$

where $f : \mathbb{R}^m \to \mathbb{R}$ is a continuously differentiable concave function and $\boldsymbol{\Sigma} \subset \mathbb{R}^m$ is a convex polyhedron. Clearly (10), and equivalently (5), are problems in this form with f a concave quadratic function and $\boldsymbol{\Sigma}$ equal to the standard simplex in $\mathbb{R}^m$. See (Frandi et al., 2013, Sections 4.1 and 4.2) for an overview of the FW algorithm.

3.1. Standard FW Algorithm

In simple terms, the FW algorithm starts at a feasible point $\boldsymbol{\alpha}_1 \in \boldsymbol{\Sigma}$, and in each step $k \in \mathbb{N}$, given the current iterate $\boldsymbol{\alpha}_k$, the algorithm finds a point $\mathbf{u}_k \in \boldsymbol{\Sigma}$ that maximizes the 1^{st} order Taylor expansion of f at $\boldsymbol{\alpha}_k$:

$$\mathbf{u}_k = \arg\max_{\mathbf{u} \in \boldsymbol{\Sigma}} \left(f(\boldsymbol{\alpha}_k) + (\mathbf{u} - \boldsymbol{\alpha}_k)^T \nabla f(\boldsymbol{\alpha}_k) \right) = \arg\max_{\mathbf{u} \in \boldsymbol{\Sigma}} (\mathbf{u} - \boldsymbol{\alpha}_k)^T \nabla f(\boldsymbol{\alpha}_k) \tag{15}$$

This tells us that $(\mathbf{u}_k - \boldsymbol{\alpha}_k)$ is an ascent direction unless $\boldsymbol{\alpha}_k$ is a stationary point of f. Then, we simply perform a line search on the line segment connecting $\boldsymbol{\alpha}_k$ and $\mathbf{u}_k$ to set the next iterate $\boldsymbol{\alpha}_{k+1}$. Since $\boldsymbol{\alpha}_k, \mathbf{u}_k \in \boldsymbol{\Sigma}$ and $\boldsymbol{\Sigma}$ is convex, their convex combination $\boldsymbol{\alpha}_{k+1}$ is in $\boldsymbol{\Sigma}$. Moreover, since in each step we are optimizing a linear function in a bounded polyhedron, i.e. a linear programming (LP) problem, $\mathbf{u}_k$ is always a vertex of $\boldsymbol{\Sigma}$. When optimizing over the standard simplex, as in (5) and (10), we have

$$\mathbf{u}_k = \arg\max_{\mathbf{u} \in \boldsymbol{\Sigma}} (\mathbf{u} - \boldsymbol{\alpha}_k)^T \nabla f(\boldsymbol{\alpha}_k) = \arg\max_{\mathbf{u} \in \boldsymbol{\Sigma}} \mathbf{u}^T \nabla f(\boldsymbol{\alpha}_k) = \mathbf{e}_{i^*} \tag{16}$$

where $i^* = \arg\max_{i \in [m]} (\nabla f(\boldsymbol{\alpha}_k))_i$, i.e. we move in the direction given by the largest component of the gradient at the current iterate (Yildirim, 2008).

While this procedure can be proved to converge globally, the convergence is slow near the optimum $\boldsymbol{\alpha}^*$ if it lies on the boundary of the feasible set $\boldsymbol{\Sigma}$. This is because the ascent direction $\mathbf{u}_k - \boldsymbol{\alpha}_k$ starts becoming perpendicular to the gradient $\nabla f(\boldsymbol{\alpha}_k)$, and the trajectory exhibits a zig-zagging behavior (GuéLat and Marcotte, 1986). For this reason, overall, the convergence rate is sublinear. Particularly, this is the case for the L_2-SVM and MEB problems since the feasible set is its own boundary, i.e. $\boldsymbol{\Sigma} = \partial\boldsymbol{\Sigma}$ when $\boldsymbol{\Sigma}$ is a simplex.

3.2. Modified FW Algorithm

Wolfe (1970) introduced a modification to the FW algorithm which quantifiably improves the convergence rate and thus, the number of iterations needed to meet stopping conditions. This modification involves exploring not just in the direction of the maximizer $\mathbf{u}_k$ of the linearization (see Equation 15), but also "away" from the minimizer $\mathbf{v}_k$ of the linearization. When optimizing over the standard simplex, as in (5) and (10), we have

$$\mathbf{v}_k = \arg\min_{\mathbf{v}\in\boldsymbol{\Sigma}} (\mathbf{v} - \boldsymbol{\alpha}_k)^T \nabla f(\boldsymbol{\alpha}_k) = \arg\min_{\mathbf{v}\in\boldsymbol{\Sigma}} \mathbf{v}^T \nabla f(\boldsymbol{\alpha}_k) = \mathbf{e}_{\tilde{j}}$$

where $\tilde{j} = \arg\min_{j\in[m]} (\nabla f(\boldsymbol{\alpha}_k))_j$, i.e. we move "away" in the direction given by the smallest component of the gradient at the current iterate (Yildirim, 2008). Note here that if $(\boldsymbol{\alpha}_k)_{\tilde{j}} = 0$, then we can't take a step in the direction away from $\mathbf{v}_k = \mathbf{e}_{\tilde{j}}$ since we can't have $(\boldsymbol{\alpha}_{k+1})_{\tilde{j}} < 0$. In fact, if $(\boldsymbol{\alpha}_k)_j = 0$, then $(\mathbf{v}_k)_j = 0$ for us to be able to take a non-zero step away from $\mathbf{v}_k$. Therefore, we constrain the components of $\mathbf{v}$ to be 0 when the corresponding component of $\boldsymbol{\alpha}$ is 0:

$$\mathbf{v}_k = \arg\min_{\substack{\mathbf{v}\in\boldsymbol{\Sigma} \\ \mathbf{v}_j = 0 \text{ if } (\boldsymbol{\alpha}_k)_j = 0}} \mathbf{v}^T \nabla f(\boldsymbol{\alpha}_k) = \mathbf{e}_{j^*} \tag{17}$$

where $j^* = \arg\min_{j:(\boldsymbol{\alpha}_k)_j > 0} (\nabla f(\boldsymbol{\alpha}_k))_j$. We call $\mathcal{I}_k = \left\{ j \in [m] : (\boldsymbol{\alpha}_k)_j > 0 \right\}$ the *coreset* at iteration k.

While we are ensured that performing a line search on the line segment connecting $\boldsymbol{\alpha}_k$ and $\mathbf{u}_k$ will yield a point which lies in $\boldsymbol{\Sigma}$ (because of its convexity), the same is not true moving away from $\mathbf{v}_k$. Therefore, our line search needs to be explicitly restricted to $\boldsymbol{\Sigma}$. Since $\mathbf{v}_k$ is of the form $\mathbf{e}_{j^*}$, we note that moving away from it can only violate the dual constraint on α_{j^*}. Therefore, the step sizes λ_k^A need to be restricted to $\left[0, \bar{\lambda}\right]$, where $\bar{\lambda}$ is such that $(\boldsymbol{\alpha}_k)_{j^*} + \bar{\lambda}\left((\boldsymbol{\alpha}_k)_{j^*} - 1\right) = 0$, i.e.

$$\bar{\lambda} = \frac{(\boldsymbol{\alpha}_k)_{j^*}}{1 - (\boldsymbol{\alpha}_k)_{j^*}} \tag{18}$$

The procedure can be summarized in the following steps:

1. Find $\mathbf{u}_k \in \Sigma$ according to Equation 16 and compute

$$d^{FW} = (\mathbf{u}_k - \boldsymbol{\alpha}_k)^T \nabla f(\boldsymbol{\alpha}_k)$$

2. Find $\mathbf{v}_k \in \Sigma$ according to Equation 17 and compute

$$d^{A} = (\boldsymbol{\alpha}_k - \mathbf{v}_k)^T \nabla f(\boldsymbol{\alpha}_k)$$

- If $d^{FW} \geq d^{A}$:

  3.1. Perform a line search to find

$$\lambda_k^{FW} = \underset{\lambda \in [0,1]}{\arg\max} f\left(\boldsymbol{\alpha}_k + \lambda\left(\mathbf{u}_k - \boldsymbol{\alpha}_k\right)\right) \tag{19}$$

  4.1. Update the iterates as

$$\boldsymbol{\alpha}_{k+1} = \boldsymbol{\alpha}_k + \lambda_k^{FW}\left(\mathbf{u}_k - \boldsymbol{\alpha}_k\right) \tag{20}$$

- If $d^{FW} < d^{A}$:

  3.2. Set $\bar{\lambda}$ according to Equation 18 and perform a line search to find

$$\lambda_k^{A} = \underset{\lambda \in \left[0,\bar{\lambda}\right]}{\arg\max} f\left(\boldsymbol{\alpha}_k + \lambda\left(\boldsymbol{\alpha}_k - \mathbf{v}_k\right)\right) \tag{21}$$

  4.2. Update the iterates as

$$\boldsymbol{\alpha}_{k+1} = \boldsymbol{\alpha}_k + \lambda_k^{A}\left(\boldsymbol{\alpha}_k - \mathbf{v}_k\right) \tag{22}$$

3.3. MFW Algorithm for the MEB Problem

Equation 16 and Equation 17 require the gradient of the objective with respect to the optimization variables. With the MEB dual problem (10), this corresponds to

$$\left(\nabla f\left(\boldsymbol{\alpha}_k\right)\right)_i = \tilde{k}\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_i\right) - 2\sum_{j\in[m]}\left(\boldsymbol{\alpha}_k\right)_j \tilde{k}\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j\right) = \tilde{k}\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_i\right) - 2\left\langle \tilde{\Phi}\left(\tilde{\mathbf{x}}_i\right), \mathbf{c}_k \right\rangle_{\mathcal{H}} \tag{23}$$

where $\mathbf{c}_k$ is the current estimate for the center of the MEB, as in Equation 11:

$$\mathbf{c}_k = \sum_{i\in[m]}\left(\boldsymbol{\alpha}_k\right)_i \tilde{\Phi}\left(\tilde{\mathbf{x}}_i\right) \tag{24}$$

Assuming $\tilde{k}$ satisfies the normalizing condition, we obtain $\mathbf{u}_k = \mathbf{e}_{i^*}$, where

$$i^* = \underset{i\in[m]}{\arg\max}\left(\nabla f\left(\boldsymbol{\alpha}_k\right)\right)_i = \underset{i\in[m]}{\arg\max}\left\|\tilde{\Phi}\left(\tilde{\mathbf{x}}_i\right) - \mathbf{c}_k\right\|^2 = \underset{i\in[m]}{\arg\min}\sum_{j\in\mathcal{I}_k}\left(\boldsymbol{\alpha}_k\right)_j \tilde{k}\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j\right) \tag{25}$$

The second equality here gives us an intuition for what FW does for the MEB problem: in each iteration, we find the mapping $\tilde{\Phi}\left(\tilde{\mathbf{x}}_i\right)$ farthest from the current estimate of the center $\mathbf{c}_k$, and move the center in its direction.

Similarly, we can compute $\mathbf{v}_k = \mathbf{e}_{j^*}$ following what we did in Equation 25:

$$j^* = \underset{i\in\mathcal{I}_k}{\arg\min}\left(\nabla f\left(\boldsymbol{\alpha}_k\right)\right)_i = \underset{i\in\mathcal{I}_k}{\arg\min}\left\|\tilde{\Phi}\left(\tilde{\mathbf{x}}_i\right) - \mathbf{c}_k\right\|^2 = \underset{i\in\mathcal{I}_k}{\arg\max}\sum_{j\in\mathcal{I}_k}\left(\boldsymbol{\alpha}_k\right)_j \tilde{k}\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j\right) \tag{26}$$

That is, the "away" steps can be interpreted as finding the feature mapping $\tilde{\Phi}\left(\tilde{\mathbf{x}}_{j^*}\right)$ closest to the current estimate of the center $\mathbf{c}_k$ and moving away from it. This has a geometric intuition as well: using complementary slackness in (8), we know that the components of the optimum $\boldsymbol{\alpha}^*$ are non-zero *if and only if* the corresponding features are on the boundary of the optimum

MEB. Hence, it makes sense to remove the points near the center from the model. This explains the naming of the *coreset* $\mathcal{I}_k$.

3.3.1. Initialization

We follow the initialization scheme laid out in Kumar et al. (2004) for MEBs, which is also followed in Tsang et al. (2005); Frandi et al. (2013) for SVMs. Specifically, we start with an arbitrary point $\tilde{\mathbf{x}}_s$ and find the farthest point $\tilde{\mathbf{x}}_j$ from it. Then, we find the farthest point $\tilde{\mathbf{x}}_k$ from $\tilde{\mathbf{x}}_j$. The initial coreset is then set to $\mathcal{I}_0 = \{j, k\}$. The center $\mathbf{c}_0$ is given by the center of $\tilde{\Phi}(\tilde{\mathbf{x}}_j)$ and $\tilde{\Phi}(\tilde{\mathbf{x}}_k)$, i.e. we have the initialization for the dual variable components, $(\boldsymbol{\alpha}_0)_j = (\boldsymbol{\alpha}_0)_k = \frac{1}{2}$ and 0 otherwise. The initial squared-radius-estimate is given by

$$\gamma_0 = \left(\frac{1}{2}\left\|\tilde{\Phi}(\tilde{\mathbf{x}}_j) - \tilde{\Phi}(\tilde{\mathbf{x}}_k)\right\|\right)^2 = \frac{1}{4}\left(2\tilde{\Delta}^2 - 2\tilde{k}(\tilde{\mathbf{x}}_j, \tilde{\mathbf{x}}_k)\right) = \frac{1}{2}\left(\Delta^2 + 2 + \frac{1}{C} + k(\mathbf{x}_j, \mathbf{x}_k)\right)$$

where we use Equation 13 and Equation 14 to assert the last equality. To add representativeness, we can condition to choose $\tilde{\mathbf{x}}_j$ and $\tilde{\mathbf{x}}_k$ from different classes. Choosing points $\mathbf{x}_j$ and $\mathbf{x}_k$ such that the initial radius is maximized then corresponds to selecting the closest pair of points, one from each of the two classes. This heuristic was also used for initialization of DirectSVM (Roobaert et al., 2006) and SimpleSVM (Vishwanathan et al., 2003).

3.3.2. Optimal Step Sizes

For the optimal step size in Equation 19, we first observe that $f(\boldsymbol{\alpha}_k + \lambda(\mathbf{u}_k - \boldsymbol{\alpha}_k))$ is a quadratic expression in λ. Hence, we can analytically maximize it. To this end, since we can only move along the direction $\mathbf{u}_k - \boldsymbol{\alpha}_k$, we set the directional derivative at $\boldsymbol{\alpha}_k + \lambda(\mathbf{u}_k - \boldsymbol{\alpha}_k)$ to 0:

$$\begin{aligned}
0 &= \nabla_{\mathbf{u}_k - \boldsymbol{\alpha}_k} f(\boldsymbol{\alpha}_k + \lambda(\mathbf{u}_k - \boldsymbol{\alpha}_k)) \\
&= (\mathbf{u}_k - \boldsymbol{\alpha}_k)^T \nabla f(\boldsymbol{\alpha}_k + \lambda(\mathbf{u}_k - \boldsymbol{\alpha}_k)) \\
&= \sum_{i\in[m]} (\mathbf{u}_k - \boldsymbol{\alpha}_k)_i \left[\tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_i) - 2\sum_{j\in[m]} \left((\boldsymbol{\alpha}_k)_j + \lambda(\mathbf{u}_k - \boldsymbol{\alpha}_k)_j\right)\tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)\right] && (27) \\
\implies 0 &= \sum_{i\in[m]}\sum_{j\in[m]} (\mathbf{u}_k - \boldsymbol{\alpha}_k)_i \left((\boldsymbol{\alpha}_k)_j + \lambda(\mathbf{u}_k - \boldsymbol{\alpha}_k)_j\right)\tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j) && (28) \\
&= (1 - 2\lambda)\left\langle \tilde{\Phi}(\tilde{\mathbf{x}}_{i^*}), \mathbf{c}_k \right\rangle_{\mathcal{H}} + (1 - \lambda)\left(f(\boldsymbol{\alpha}_k) - \tilde{\Delta}^2\right) + \lambda\tilde{\Delta}^2 && (29)
\end{aligned}$$

where we substitute Equation 24 to get Equation 27, use the normalizing condition for $\tilde{k}$ along with the fact that the entries of $\mathbf{u}_k$ and $\boldsymbol{\alpha}_k$ sum to 1 to get Equation 28, use $(\mathbf{u}_k)_j = \delta_{i^*j}$ to get Equation 29. Rearranging the terms gives us the optimal step size,

$$\lambda_k^{FW} = \frac{1}{2}\left(1 - \frac{\gamma_k}{\left\|\tilde{\Phi}(\tilde{\mathbf{x}}_{i^*}) - \mathbf{c}_k\right\|^2}\right) = \frac{1}{2}\left(1 - \frac{\gamma_k}{2\tilde{\Delta}^2 - 2\sum_{i\in[m]}(\boldsymbol{\alpha}_k)_i \tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_{i^*})}\right) \qquad (30)$$

where $\gamma_k = f(\boldsymbol{\alpha}_k)$ is our current estimate of the squared-radius of the MEB, as in Equation 12.

Similarly, we can show that the best step in the "away" direction is

$$\tilde{\lambda}_k^A = \frac{1}{2}\left(\frac{\gamma_k}{\left\|\tilde{\Phi}\left(\tilde{\mathbf{x}}_{j^*}\right) - \mathbf{c}_k\right\|^2} - 1\right) = \frac{1}{2}\left(\frac{\gamma_k}{2\tilde{\Delta}^2 - 2\sum_{i\in[m]}\left(\boldsymbol{\alpha}_k\right)_i \tilde{k}\left(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_{j^*}\right)} - 1\right)$$

However, to keep satisfying the dual constraint, we clip this best value using Equation 18:

$$\lambda_k^A = \min\left\{\tilde{\lambda}_k^A, \bar{\lambda}\right\} \tag{31}$$

3.3.3. Updating the Coreset

If we take the standard FW step (see Equation 20), we can update the coreset as $\mathcal{I}_{k+1} = \mathcal{I}_k \cup \{i^*\}$, i.e. if i^* is not currently in the coreset, and we take a step in its direction, then we add it to the set. Else, the coreset remains unchanged.

We recall that in an "away" step, taking a full-step (with step size $\bar{\lambda}$) sets $\left(\boldsymbol{\alpha}_{k+1}\right)_{j^*} = 0$ (see Equation 22). Hence, taking a full-step in the "away" direction implies dropping the corresponding example from the coreset, $\mathcal{I}_{k+1} = \mathcal{I}_k \setminus \{j^*\}$. If we don't take a full-step, the coreset remains unchanged.

3.3.4. Termination Condition

Let η is the total number of iterations in our approximate algorithm; γ_η will be the corresponding dual objective (approximate MEB radius-squared) returned. Let γ^* be the the optimal objective. We say that the algorithm has an approximation ratio $\rho(m)$ for an input size m if

$$\max\left\{\frac{\gamma_\eta}{\gamma^*}, \frac{\gamma^*}{\gamma_\eta}\right\} = \rho(m) \geq 1$$

Intuitively, this ratio measures how bad the approximate solution is compared with the optimal solution. A large (small) approximation ratio means the solution is much worse than (more or less the same as) the optimal solution. If the ratio does not depend on m, we may just write ρ and call the algorithm an ρ-approximation algorithm.

(Yildirim, 2008, Theorem 4.1) states that if the MFW algorithm with the termination condition $1 + \delta_k \leq (1+\epsilon)^2$ is applied to the MEB problem, then γ_η is a $(1+\epsilon)^2$-approximation to γ^*. In other words, the MEB radius is approximated within a $(1+\epsilon)$ factor. This termination condition has also been used in Bădoiu and Clarkson (2008); Tsang et al. (2005), albeit for a different optimization algorithm. We use this in our implementation as well.

3.3.5. Convergence Results

Dual Iterates

For constrained optimization of a $C^1(S)$ function over a polyhedron S, (GuéLat and Marcotte, 1986, Theorem 4) states that MFW is globally convergent under a mild condition:

Theorem 1 (Global Convergence)**.** *Let $\{\boldsymbol{\alpha}_k\}$ be a sequence generated by MFW, with $\boldsymbol{\alpha}_0 \in S$. If ∇f is Lipschitz continuous on S, then $\lim_{k\to\infty} f(\boldsymbol{\alpha}_k) = f(\boldsymbol{\alpha}^*)$.*

We start by noting that for the MEB problem, ∇f is a linear function (see Equation 23): $\nabla f(\boldsymbol{\alpha}) = b - 2K\boldsymbol{\alpha}$. Hence, it is trivially Lipschitz continuous:

$$\left\|\nabla f(\boldsymbol{\alpha}_1) - \nabla f(\boldsymbol{\alpha}_2)\right\|_2 = 4\left\|K(\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2)\right\|_2 \leq 4\left\|K\right\|_2 \left\|\boldsymbol{\alpha}_1 - \boldsymbol{\alpha}_2\right\|_2$$

where $b = \left[\tilde{k}(\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_1) \quad \dots \quad \tilde{k}(\tilde{\mathbf{x}}_m, \tilde{\mathbf{x}}_m)\right] \in \mathbb{R}^m$, $K \in \mathbb{R}^{m\times m}$ is the MEB kernel matrix, $[K]_{ij} = \tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$ and $\|K\|_2$ denotes the L_2 matrix norm. Moreover, our MFW algorithm starts in S and S is a simplex – a special kind of polyhedron. Hence, the conditions of Theorem 1 are satisfied, and we conclude that the MFW algorithm for the MEB problem converges globally.

Dual Objective

(Wolfe, 1970, Theorem 5) initially outlined the linear convergence of MFW at maximizing a concave function over a polytope, given that ∇f is Lipschitz continuous, f is strongly concave, and strict complementarity is observed. Ahipasaoglu et al. (2008) confirmed the linear convergence of a similar algorithm over the unit simplex, albeit under slightly altered assumptions. Unfortunately, these earlier findings do not directly apply to our situation because both sets of conditions require that the optimal solution is unique, a condition not generally met by the dual of the MEB problem Yildirim (2008).

Staying consistent with Yildirim (2008), we will refer to the MFW iterations using standard FW step as a plus-iteration and those using an "away" step as a minus-iteration. The following lemma is from (Yildirim, 2008, Lemma 4.1) which corresponds to the MFW algorithm for the MEB problem:

Lemma 1 (Increasing Dual Objective). *At each plus- or minus-iteration,*

$$\gamma_{k+1} \geq \gamma_k \left(1 + \frac{\delta_k^2}{4(1+\delta_k)}\right)$$

where $\gamma_k = f(\boldsymbol{\alpha}_k)$, $\delta_k = \max\left\{\delta_k^+, \delta_k^-\right\}$ *and*

$$\delta_k^+ = \max_{i\in[m]} \frac{\|\tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c}_k\|^2}{\gamma_k} - 1 \tag{32}$$

$$\delta_k^- = 1 - \min_{i\in\mathcal{I}_k} \frac{\|\tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c}_k\|^2}{\gamma_k} \tag{33}$$

Note that $\delta_k \geq 0$ if at least one of the following is true:

- $\delta_k^+ \geq 0$, or equivalently $\max_{i\in[m]} \|\tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c}_k\|^2 \geq \gamma_k$, i.e. there is at least one mapping not in the current MEB, $\mathcal{B}\left(\mathbf{c}_k, \sqrt{\gamma_k}\right)$; δ_k^+ is the smallest factor by which the current ball should be expanded so that their are no points outside it.
- $\delta_k^- \geq 0$, or equivalently $\gamma_k \geq \min_{i\in\mathcal{I}_k} \|\tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c}_k\|^2$, i.e. a coreset point is not on the boundary of the current MEB; δ_k^- is the smallest factor by which the current ball should be shrunk so that their are no coreset vectors in its interior.

Now, since

$$\min_{i\in\mathcal{I}_k} \|\tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c}_k\|^2 \leq \max_{i\in\mathcal{I}_k} \|\tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c}_k\|^2 \leq \max_{i\in[m]} \|\tilde{\Phi}(\tilde{\mathbf{x}}_i) - \mathbf{c}_k\|^2$$

one of the two conditions above must be satisfied. Hence, $\delta_k \geq 0$, and from Lemma 1, the sequence $\{\gamma_k\}_{k\in\mathbb{N}}$ is non-decreasing.

Now, we present (Yildirim, 2008, Theorem 4.2), which concludes the (asymptotic) linear convergence of the dual objective, f:

Theorem 2 (Asymptotically Linear Convergence)**.** *Given $S = \left\{\tilde{\Phi}\left(\tilde{\mathbf{x}}_i\right) : i \in [m]\right\}$, the MFW algorithm applied to estimate the MEB of S computes iterates $\boldsymbol{\alpha}_k$ such that $f\left(\boldsymbol{\alpha}^*\right) - f\left(\boldsymbol{\alpha}_k\right)$ is non-increasing. Asymptotically, we have*

$$\frac{f\left(\boldsymbol{\alpha}^*\right) - f\left(\boldsymbol{\alpha}_{k+1}\right)}{f\left(\boldsymbol{\alpha}^*\right) - f\left(\boldsymbol{\alpha}_k\right)} \leq M \leq 1 - \frac{1}{36mld_S^2}$$

where l is a positive constant and d_S is the diameter of S.

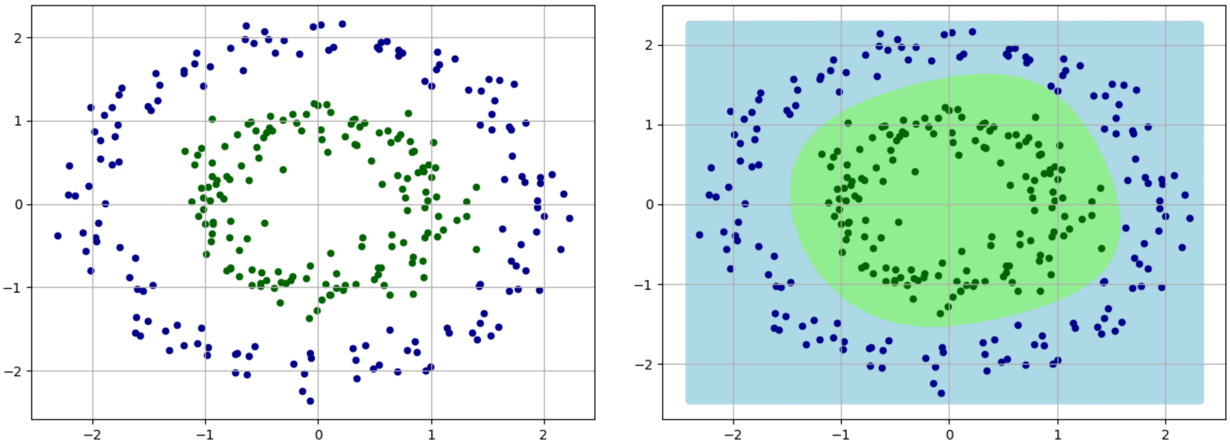
Theorem 2 tells us that after a sufficient number of iterations, the dual objective f converges (locally) linearly to the optimal, and the exact rate is determined by the number of data points m; the rate is independent of the dimension of the feature space $\tilde{\mathcal{H}}$. Finally, note here that a bound on the number of iterations would depend on S as it is not known *a priori* when the linear convergence will kick in.

3.3.6. Complexity Analysis

Algorithm 1 details MFW algorithm for the MEB problem (Frandi et al., 2013, Algorithm 3). We note that computational bottlenecks are finding the search directions in each step, with a complexity of $\mathcal{O}\left(m\left|\mathcal{I}_k\right|\right) = \mathcal{O}\left(m^2\right)$ incurred upon multiplying the $m \times \left|\mathcal{I}_k\right|$ kernel matrix with the $\left|\mathcal{I}_k\right| \times 1$ dual variable components corresponding to the coreset. In practice, the coreset is a small fraction of the total number of points (Kumar et al., 2004), meaning that we save a lot of runtime by using only the dual components for computations. As discussed earlier, the linear convergence is only asymptotic. As such, we cannot guarantee the total number of iterations. However, once linear convergence kicks in, the number of iterations is of the order $\mathcal{O}\left(\log \epsilon / \log M\right)$.

As for the memory complexity, we store the kernel evaluations of the coreset vectors with all the points. Hence, the memory needed is of the order $\mathcal{O}\left(m \cdot \max_k \left|\mathcal{I}_k\right|\right) = \mathcal{O}\left(m^2\right)$. Again, the small size of the coreset allows us to optimize the SVM efficiently. An alternate analysis can be performed using the complexity of the size of the coreset at convergence: $\mathcal{I}_\eta = \mathcal{O}\left(1/\epsilon\right)$ (Yildirim, 2008, Theorem 4.1). This gives $\mathcal{O}\left(m \cdot \max_k \left|\mathcal{I}_k\right|\right) = \mathcal{O}\left(m/\epsilon\right)$.

4. Experiments



(a) Input Dataset

(b) L_2-SVM Decision Function

Figure 1: 2-Circles Dataset. Blue color corresponds to the label -1 and green is for label $+1$.

In this section, we perform classification on a 2-circles dataset, as in Figure 1a, with a total of $m = 300$ points. Clearly, this dataset is not linearly separable. For that reason, we use the Gaussian kernel with the scale parameter $\gamma^2 = 2^{-1}$ to perform classification in a higher dimensional feature space:

$$k(\mathbf{x}_1, \mathbf{x}_2) = e^{-\gamma^2 \|\mathbf{x}_1 - \mathbf{x}_2\|^2}$$

In the L_2-SVM formulation, C is set to 10^3. These hyper-parameters were selected over a logarithmic grid $[2^{-6}, 2^6] \times [10^{-3}, 10^6]$, using cross validation. For the stopping condition, we set $\epsilon = 10^{-6}$, as in (Frandi et al., 2013). The algorithm is trained with a random 80% of the complete dataset, and then tested on the rest 20%. The learnt classification function is shown in Figure 1b. Listing 3 contains the `Python` implementation of Algorithm 1, with the associated SVM Kernel and MEB Kernel implementations in Listing 1 and Listing 2, respectively.

4.1. Convergence Results

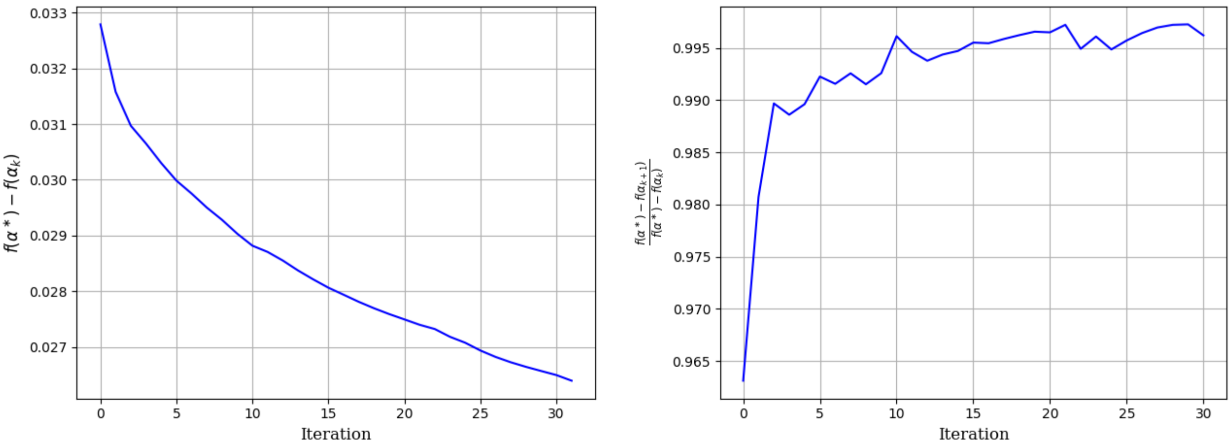


Figure 2: Dual Objective Convergence

Figure 2 shows that the dual objective monotonically decreases, agreeing with Lemma 1. Moreover, it converges linearly, as we expected from Theorem 2, but near the optimum, the rate of convergence is slow. This is in line with the limitations of FW we noted earlier. It seems like the modified FW algorithm doesn't entirely fix the limitations of standard FW near the feasible set's boundary.

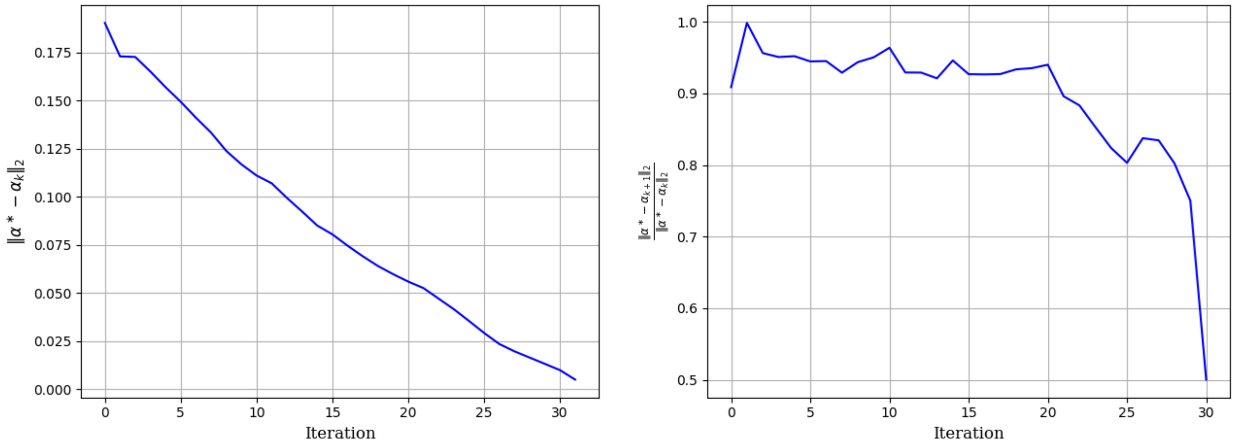
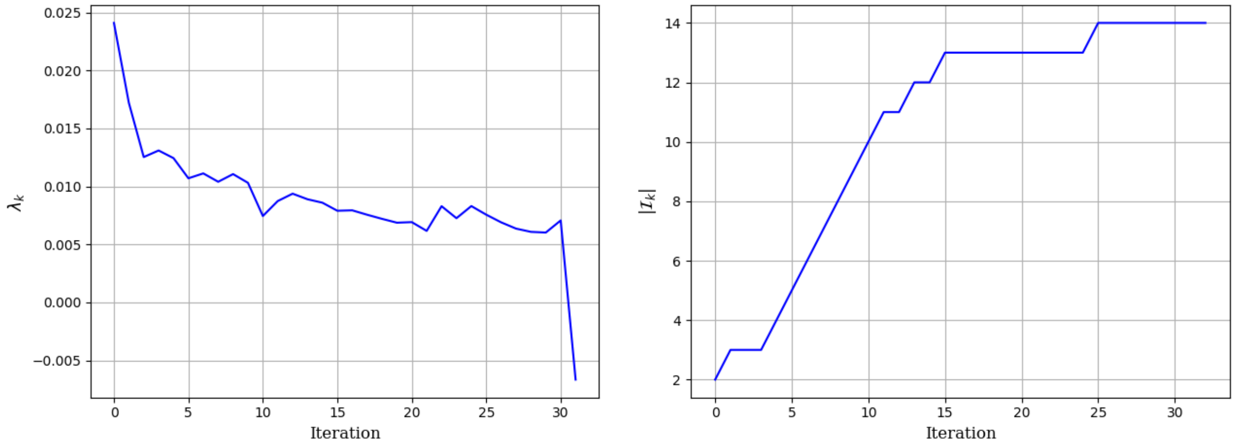


Figure 3: Dual Iterates Convergence

Figure 3 shows that the dual iterates converge linearly to the optimum as well, and while the convergence is slow at the beginning, it gets faster as the optimization progresses.



(a) Step Sizes

(b) Coreset Sizes

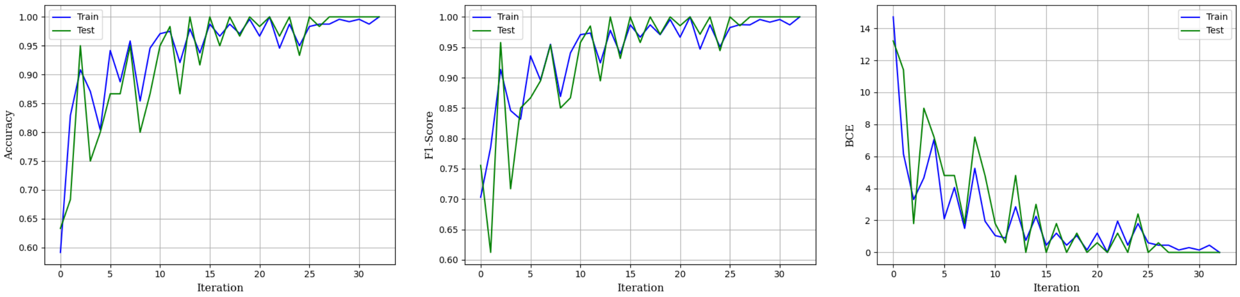
Figure 4

Figure 4a shows that the step sizes became smaller as we optimized. Furthermore, only in the last iteration, we took an "away" step (inidcated by a negative step size). This also explains why the slow convergence of FW near the optimum was not fixed despite the modification – the "away" steps were not chosen enough times. Figure 4b shows that the coreset size only increased over the optimization routine, meaning we never added any points to the coreset that are not a part of the MEB's coreset. Importantly, note that we have at most 14 points in the coreset, much smaller than the total number of 240 training points we used to fit the model, making this a lot more efficient than a naive implementation.

4.2. Physical Cost

The most expensive data structure in our problem is the kernel matrix storing the MEB kernel evaluations between the coreset vectors and all the training inputs. At 64-bit precision, the largest size this (`NumPy`) matrix gets to is 26.880 KBs, at shape $(240, 14)$. From the initialization to the end of the optimization process, the algorithm takes nearly 16 ms. Even with $m = 3,000$ data points, the algorithm takes only around 915 ms, and stores the largest kernel matrix of shape $(2400, 70)$, i.e. with 70 support vectors, at 1.344 MBs. This implies that our method is highly scalable.

4.3. Classification Performance



(a) Accuracy

(b) F1-Score

(c) BCE Loss

Figure 5: Classification Metrics

Figure 5 shows some classification metrics on both training and test sets. Particularly, we note that while the performance improves as we step through time, there is some variability from step to step. This can be expected since the optimization algorithm is not designed to particularly optimize any of these metrics. With enough steps, we see that the model is correctly able to classify all the points in both, training and test sets. Finally, we note that the performance on both sets is similar, indicating that our model is not over-fitting to the training set.

References

Ahipasaoglu, S. D., Sun, P., and Todd, M. J. (2008). Linear convergence of a modified frank-wolfe algorithm for computing minimum-volume enclosing ellipsoids. *Optimization Methods Software*, 23(1):5–19.

Bădoiu, M. and Clarkson, K. L. (2008). Optimal core-sets for balls. *Computational Geometry*, 40(1):14–22.

Frandi, E., Ñanculef, R., Gasparo, M. G., Lodi, S., and Sartori, C. (2013). Training support vector machines using frank-wolfe optimization methods. *International Journal of Pattern Recognition and Artificial Intelligence*, 27(03):1360003.

Frandi, E., Ñanculef, R., and Suykens, J. A. K. (2015). A partan-accelerated frank-wolfe algorithm for large-scale svm classification. In *2015 International Joint Conference on Neural Networks (IJCNN)*, pages 1–8.

Frank, M. and Wolfe, P. (1956). An algorithm for quadratic programming. *Naval Research Logistics Quarterly*, 3(1-2):95–110.

Frieß, T.-T., Cristianini, N., and Campbell, C. (1998). The kernel-adatron algorithm: A fast and simple learning procedure for support vector machines. In *Proceedings of the Fifteenth International Conference on Machine Learning*, ICML '98, page 188–196, San Francisco, CA, USA. Morgan Kaufmann Publishers Inc.

GuéLat, J. and Marcotte, P. (1986). Some comments on wolfe's 'away step'. *Mathematical Programming*, 35(1):110–119.

Herbrich, R. and Graepel, T. (2000). A pac-bayesian margin bound for linear classifiers: Why svms work. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

Kumar, P., Mitchell, J. S. B., and Yildirim, E. A. (2004). Approximate minimum enclosing balls in high dimensions using core-sets. *ACM J. Exp. Algorithmics*, 8:1.1–es.

Lee, Y.-J. and Mangasarian, O. (2001a). Ssvm: A smooth support vector machine for classification. *Computational Optimization and Applications*, 20(1):5–22.

Lee, Y.-J. and Mangasarian, O. L. (2001b). *RSVM: Reduced Support Vector Machines*, volume 1, pages 1–17.

Mangasarian, O. and Musicant, D. (1999). Successive overrelaxation for support vector machines. *IEEE Transactions on Neural Networks*, 10(5):1032–1037.

Mangasarian, O. and Musicant, D. (2000). Active support vector machine classification. In Leen, T., Dietterich, T., and Tresp, V., editors, *Advances in Neural Information Processing Systems*, volume 13. MIT Press.

Mangasarian, O. L. and Musicant, D. R. (2001). Lagrangian support vector machines. *J. Mach. Learn. Res.*, 1:161–177.

Pang, J.-S. (1983). Methods for quadratic programming: A survey. *Computers & Chemical Engineering*, 7(5):583–594.

Roobaert, D., Karakoulas, G., and Chawla, N. V. (2006). *Information Gain, Correlation and Support Vector Machines*, pages 463–470. Springer Berlin Heidelberg, Berlin, Heidelberg.

Scholkopf, B. and Smola, A. J. (2001). *Learning with Kernels: Support Vector Machines, Regularization, Optimization, and Beyond.* MIT Press, Cambridge, MA, USA.

Sentelle, C., Anagnostopoulos, G. C., and Georgiopoulos, M. (2008). A fast revised simplex method for svm training. In *2008 19th International Conference on Pattern Recognition*, pages 1–4.

Shawe-Taylor, J., Bartlett, P., Williamson, R., and Anthony, M. (1998). Structural risk minimization over data-dependent hierarchies. *IEEE Transactions on Information Theory*, 44(5):1926–1940.

Smola, A. J., Bartlett, P., Schölkopf, B., and Schuurmans, D. (2000). *Advances in Large-Margin Classifiers.* The MIT Press.

Tsang, I.-H., Kwok, J.-Y., and Zurada, J. (2006). Generalized core vector machines. *IEEE Transactions on Neural Networks*, 17(5):1126–1140.

Tsang, I. W., Kwok, J. T., and Cheung, P.-M. (2005). Core vector machines: Fast svm training on very large data sets. *Journal of Machine Learning Research*, 6(13):363–392.

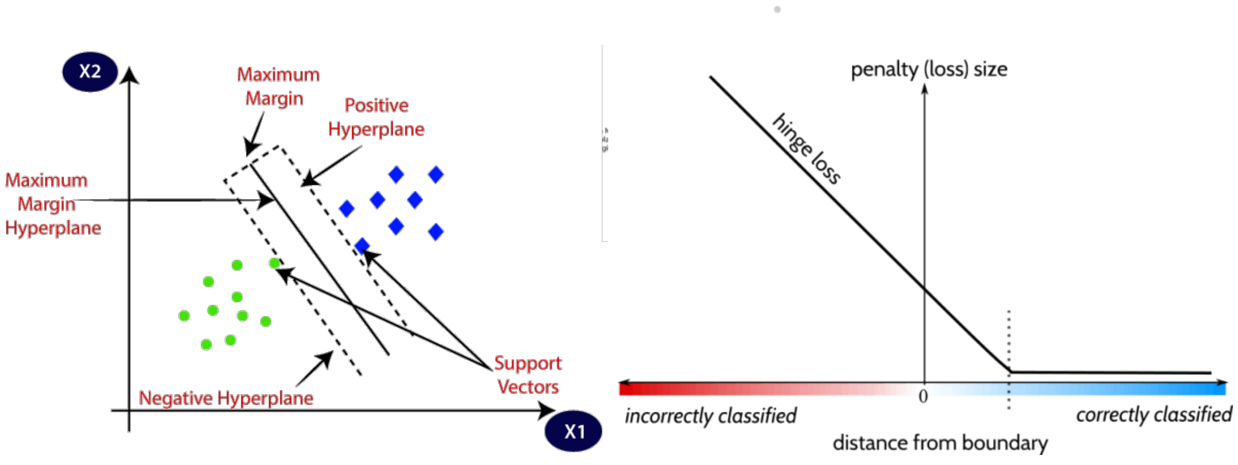
Vishwanathan, S. V. N., Smola, A. J., and Murty, M. N. (2003). Simplesvm. In *Proceedings of the Twentieth International Conference on International Conference on Machine Learning*, ICML'03, page 760–767. AAAI Press.

Wolfe, P. (1970). Convergence theory in nonlinear programming. *Integer and nonlinear programming*, pages 1–36.

Yildirim, E. A. (2008). Two algorithms for the minimum enclosing ball problem. *SIAM Journal on Optimization*, 19(3):1368–1391.

Appendix A. Support Vector Machines

A.1. Problem Description



(a) (b)

Figure 6: *Left*: SVM[a]. *Right*: Hinge loss[b].

[a] Source: analyticsvidhya.com/blog/2021/10/support-vector-machinessvm-a-complete-guide-for-beginners

[b] Source: hackernoon.com/hinge-loss-a-steadfast-loss-evaluation-function-for-the-svm-classification-models-in-ai-and-ml

Assume we are given a finite number, say $m \in \mathbb{N}$, of point pairs, $(\mathbf{x}_i, y_i) \in \mathcal{X} \times \mathcal{Y}$, where $i \in [m] = \{1, \ldots, m\}$ and $\mathcal{Y} = \{-1, +1\}$. Our objective is to find a *maximum margin hyperplane* in a Hilbert space, $\mathcal{H}$, that separates the features $\{\Phi(\mathbf{x}_i) : y_i = -1\}$ and $\{\Phi(\mathbf{x}_i) : y_i = +1\}$, where $\Phi : \mathcal{X} \to \mathcal{H}$ is a feature mapping (see Figure 6a). The benefit of working in a feature space, $\mathcal{H}$, is that it allows us to capture $\mathbf{x}$-y relationships that cannot be modelled using a linear classifier in the input space, $\mathcal{X}$.

We begin by noting that any hyperplane in $\mathcal{H}$ can be written as $\{\phi \in \mathcal{H} : \langle \phi, \mathbf{w} \rangle_{\mathcal{H}} + b = 0\}$, with $\mathbf{w} \in \mathcal{H}$ and $b \in \mathbb{R}$. Such a hyperplane induces a classifier, $f(\cdot; \mathbf{w}, b) : \mathcal{X} \to \mathcal{Y}$, defined as $f(\mathbf{x}; \mathbf{w}, b) = \operatorname{sign}(\langle \Phi(\mathbf{x}), \mathbf{w} \rangle_{\mathcal{H}} + b)$, where $\operatorname{sign}(\cdot) : \mathbb{R} \to \{-1, +1\}$ is defined as

$$\operatorname{sign}(x) = \begin{cases} -1, & x < 0 \\ +1, & x \geq 0 \end{cases}$$

Now, the distance of any feature, $\Phi(\mathbf{x})$, from this hyperplane is given by

$$d(\mathbf{x}; \mathbf{w}, b) = \frac{|\langle \Phi(\mathbf{x}), \mathbf{w} \rangle_{\mathcal{H}} + b|}{\|\mathbf{w}\|_{\mathcal{H}}}$$

A.2. Hard-Margin SVM

For now, assume that the points are linearly separable in $\mathcal{H}$, and that $f(\cdot; \mathbf{w}, b)$ is an error-free classifier, i.e. $f(\mathbf{x}_i; \mathbf{w}, b) = y_i$, $\forall i \in [m]$. Then, the distance of any mapping from the corresponding hyperplane is given by

$$\tilde{d}\left(\mathbf{x}_i; \mathbf{w}, b\right) = \frac{y_i \left(\left\langle \Phi\left(\mathbf{x}_i\right), \mathbf{w}\right\rangle_{\mathcal{H}} + b\right)}{\|\mathbf{w}\|_{\mathcal{H}}}$$

Since this distance does not change when we scale $\mathbf{w}$ and b with the same factor, i.e. $\mathbf{w} \to t\mathbf{w}$ and $b \to tb$, for $t \in \mathbb{R}$, we can simply scale $\mathbf{w}$ and b such that $\min_{i\in[m]} y_i \left(\left\langle \Phi\left(\mathbf{x}_i\right), \mathbf{w}\right\rangle_{\mathcal{H}} + b\right) = 1$, and we have the same error-free classifier. Then, our maximum margin objective is equivalent to the following constrained optimization problem:

$$\begin{aligned} \min_{\mathbf{w},b} \quad & \frac{1}{2}\|\mathbf{w}\|_{\mathcal{H}}^2 \\ \text{subject to} \quad & y_i \left(\left\langle \Phi\left(\mathbf{x}_i\right), \mathbf{w}\right\rangle_{\mathcal{H}} + b\right) \geq 1, \ \forall i \in [m] \end{aligned} \tag{34}$$

which gives us a hard-margin classifier.

A.3. The (Soft-Margin) L_1-SVM Optimization Problem

We relax our error-free classification constraint and instead modify the objective to trade-off the margin width a loss incurred on the training points:

$$\min_{\mathbf{w},b} \left(\frac{1}{2}\|\mathbf{w}\|_{\mathcal{H}}^2 + C \sum_{i\in[m]} \ell\left(y_i \left(\left\langle \Phi\left(\mathbf{x}_i\right), \mathbf{w}\right\rangle_{\mathcal{H}} + b\right)\right) \right) \tag{35}$$

where $C > 0$ controls the trade-off and we use the hinge loss, $\ell : \mathbb{R} \to \mathbb{R}_+$, defined as $\ell(x) = \max(0, 1 - x)$ (see Figure 6b). This gives us a soft-margin classifier – a classifier that can make mistakes even on the points it is fitted on. Clearly, this allows us to tackle the case when the dataset is linearly inseparable in the feature space – a situation hard-margin SVMs are not designed for. Furthermore, we will see in (36) that we can also interpret this as adding L_1 regularization to the objective.

Discussion on the Hinge Loss

The hinge loss, ℓ, can be seen as a smooth approximation of the characteristic function ι_- on the negative reals (which can be used to make the inequality constraint in (35) implicit). Such a smooth approximation is needed to use Newton's method. Hinge loss has a few favorable properties:

- Like ι_-, ℓ is convex and decreasing.
- It equals ι_- on $[1, \infty)$, and approaches ∞ as the argument goes to $-\infty$, like ι_-.
- Unlike ι_-, ℓ is differentiable and closed.

Let's interpret the hinge loss:

- When $y_i \left(\left\langle \Phi\left(\mathbf{x}_i\right), \mathbf{w}\right\rangle_{\mathcal{H}} + b\right) < 0$, the point $\mathbf{x}_i$ is misclassified by the classifier $f(\cdot; \mathbf{w}, b)$ and we incur a penalty equal to the distance from the hyperplane corresponding to y_i.
- When $0 < y_i \left(\left\langle \Phi\left(\mathbf{x}_i\right), \mathbf{w}\right\rangle_{\mathcal{H}} + b\right) < 1$, the point is correctly classified but it lies between the margin and the corresponding hyperplane, i.e. it is too close to the decision boundary.

Therefore, we penalize such a point in the same way, but in this case the hinge loss is between 0 and 1. In other words, we penalise the classifier but not as much as if the point was incorrectly classified.

- Finally, when $1 < y_i \left(\langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} + b \right)$, i.e. the point is on the correct side of the corresponding hyperplane, we don't penalise the classifier.

A.3.1. Primal Problem

We turn our soft margin optimization objective (35) into a differentiable objective by introducing auxiliary variables, $\boldsymbol{\xi} = \{\xi_i\}_{i \in [m]}$:

$$\begin{aligned} \min_{\mathbf{w}, b, \boldsymbol{\xi}} \quad & \frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 + C \sum_{i \in [m]} \xi_i \\ \text{subject to} \quad & y_i \left(\langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} + b \right) \geq 1 - \xi_i, \\ & \xi_i \geq 0, \ \forall i \in [m] \end{aligned} \tag{36}$$

where either $y_i \left(\langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} + b \right) \geq 1$ and $\xi_i = 0$ as in (34), or $y_i \left(\langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} + b \right) < 1$ and $\xi_i = 1 - y_i \left(\langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} + b \right)$. For this reason, ξ_i are called slack variables.

A.3.2. Dual Problem

The Lagrangian of the problem is given by

$$\begin{aligned} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\lambda}) = & \frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 + C \sum_{i \in [m]} \xi_i \\ & + \sum_{i \in [m]} \alpha_i \left(1 - \xi_i - y_i \left(\langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} + b \right) \right) + \sum_{i \in [m]} \lambda_i (-\xi_i) \end{aligned}$$

with dual variable constraints $\alpha_i \geq 0$ and $\lambda_i \geq 0$, $\forall i \in [m]$. Our dual objective is given by

$$\begin{aligned} g(\boldsymbol{\alpha}, \boldsymbol{\lambda}) &= \inf_{\mathbf{w}, b, \boldsymbol{\xi}} L(\mathbf{w}, b, \boldsymbol{\xi}, \boldsymbol{\alpha}, \boldsymbol{\lambda}) \\ &= \inf_{\mathbf{w}} \left(\frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 - \sum_{i \in [m]} \alpha_i y_i \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} \right) - \inf_{b} \left(\sum_{i \in [m]} \alpha_i y_i \right) b + \inf_{\boldsymbol{\xi}} \sum_{i \in [m]} \xi_i (C - \alpha_i - \lambda_i) \\ &\quad + \sum_{i \in [m]} \alpha_i \end{aligned}$$

The first term is the unique minimum of a 1-strongly convex function, which we can derive by setting the gradient with respect to $\mathbf{w}$ to 0:

$$\frac{\partial}{\partial \mathbf{w}} \left(\frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 - \sum_{i \in [m]} \alpha_i y_i \langle \Phi(\mathbf{x}_i), \mathbf{w} \rangle_{\mathcal{H}} \right) = \mathbf{w} - \sum_{i \in [m]} \alpha_i y_i \Phi(\mathbf{x}_i) = 0 \Longrightarrow \mathbf{w} = \sum_{i \in [m]} \alpha_i y_i \Phi(\mathbf{x}_i)$$

which gives

$$\inf_{\mathbf{w}} \left(\frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 - \sum_{i\in[m]} \alpha_i y_i \left\langle \Phi\left(\mathbf{x}_i\right), \mathbf{w} \right\rangle_{\mathcal{H}} \right) = -\frac{1}{2} \left\| \sum_{i\in[m]} \alpha_i y_i \Phi\left(\mathbf{x}_i\right) \right\|^2$$

As for the other two terms, they involve an infimum over affine functions of b and $\boldsymbol{\xi}$. Hence, they are equal to $-\infty$ if any of the coefficients are non-zero. In summary,

$$g\left(\boldsymbol{\alpha}, \boldsymbol{\lambda}\right) = \begin{cases} \sum_{i\in[m]} \alpha_i - \frac{1}{2} \left\| \sum_{i\in[m]} \alpha_i y_i \Phi\left(\mathbf{x}_i\right) \right\|^2, & \sum_{i\in[m]} \alpha_i y_i = 0 \text{ and } \alpha_i = C - \lambda_i, \ \forall i \in [m] \\ -\infty, & \text{otherwise} \end{cases}$$

Finally, we eliminate the dependence on $\boldsymbol{\lambda}$ by noting that $\alpha_i = C - \lambda_i \implies \alpha_i \leq C$. This gives us our dual problem,

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i\in[m]} \alpha_i - \frac{1}{2} \left\| \sum_{i\in[m]} \alpha_i y_i \Phi\left(\mathbf{x}_i\right) \right\|^2 \\ \text{subject to} \quad & \sum_{i\in[m]} \alpha_i y_i = 0 \text{ and } 0 \leq \alpha_i \leq C, \ \forall i \in [m] \end{aligned} \tag{37}$$

Since Slater's conditions are satisfied, the KKT conditions are necessary and sufficient for optimality. Setting the derivative of the Lagrangian with respect to $\mathbf{w}$ to 0 gives us

$$\mathbf{w}^* = \sum_{i\in[m]} \alpha_i^* y_i \Phi\left(\mathbf{x}_i\right) \tag{38}$$

Hence, we can solve the dual problem to find $\boldsymbol{\alpha}^*$, and then use Equation 38 to recover the primal optimum, $\mathbf{w}^*$.

A.4. Kernel SVM

(37) involves inner products between the feature vectors, suggesting that we can kernelize the problem, thereby doing away with explicit feature mappings which are harder to specify and messier to work with than the induced kernel.

A.4.1. Primal Problem

Let $\Phi(x) = k(x, \cdot) \in \mathcal{H}$ induce a positive definite kernel, $k(\cdot,\cdot) : \mathcal{X} \times \mathcal{X} \to \mathbb{R}$, with $\mathcal{H}$ its Reproducing Kernel Hilbert Space (RKHS) of functions on $\mathcal{X}$. Then we can rewrite (35) as

$$\min_{\mathbf{w},b} \left(\frac{1}{2} \|\mathbf{w}\|_{\mathcal{H}}^2 + C \sum_{i\in[m]} \ell\left(y_i \left(\left\langle k\left(\mathbf{x}_i, \cdot\right), \mathbf{w}\right\rangle_{\mathcal{H}} + b\right)\right) \right) \tag{39}$$

A.4.2. Dual Problem

Following (37), we can rewrite the dual problem of (39) in a kernelized form

$$\begin{aligned} \max_{\boldsymbol{\alpha}} \quad & \sum_{i\in[m]} \alpha_i - \frac{1}{2} \sum_{i\in[m]} \sum_{i\in[j]} \alpha_i \alpha_j y_i y_j k\left(\mathbf{x}_i, \mathbf{x}_j\right) \\ \text{subject to} \quad & \sum_{i\in[m]} \alpha_i y_i = 0 \quad \text{and} \quad 0 \leq \alpha_i \leq C,\ \forall i \in [m] \end{aligned} \tag{40}$$

The solution to the primal problem, $\mathbf{w}^*$, can be recovered from the solution to the dual problem, $\boldsymbol{\alpha}^*$, by setting

$$\mathbf{w}^*\left(\cdot\right) = \sum_{i\in[m]} \alpha_i^* y_i k\left(\mathbf{x}_i, \cdot\right) \tag{41}$$

Appendix B. Pseudocode and Implementation

Algorithm 1 Modified Frank-Wolfe Algorithm for the Minimum Enclosing Ball Problem

```
Require: $S$, $\epsilon$.
initialization: compute $\mathcal{I}_0$ and $\boldsymbol{\alpha}_0$;
$\tilde{\Delta}^2 \longleftarrow \tilde{k}(\tilde{\mathbf{x}}_1, \tilde{\mathbf{x}}_1)$;    ▷ Equation 14
$R_0 \longleftarrow \sum_{i,j\in\mathcal{I}_0} (\boldsymbol{\alpha}_0)_i (\boldsymbol{\alpha}_0)_j \tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$;
$r_0^2 \longleftarrow \tilde{\Delta}^2 - R_0$;    ▷ Equation 12
$i^* \longleftarrow \arg\max_{i\in[m]} \gamma^2(\boldsymbol{\alpha}_0; i) := \tilde{\Delta}^2 + R_0 - 2\sum_{j\in\mathcal{I}_0} (\boldsymbol{\alpha}_0)_j \tilde{k}(\tilde{\mathbf{x}}_j, \tilde{\mathbf{x}}_i)$;    ▷ Equation 25
$j^* \longleftarrow \arg\min_{j\in\mathcal{I}_0} \gamma^2(\boldsymbol{\alpha}_0; j)$;    ▷ Equation 26
$\delta_0^+ \longleftarrow \frac{\gamma^2(\boldsymbol{\alpha}_0;i^*)}{r_k^2} - 1$;    ▷ Equation 32
$\delta_0^- \longleftarrow 1 - \frac{\gamma^2(\boldsymbol{\alpha}_0;j^*)}{r_k^2}$;    ▷ Equation 33
$k \longleftarrow 0$;
while $\delta_k^+ > (1+\epsilon)^2 - 1$ do    ▷ Section 3.3.4
    if $\delta_k^+ \geq \delta_k^-$ then    ▷ Lemma 1
        $\lambda_k \longleftarrow \frac{1}{2}\left(1 - \frac{r_k^2}{\gamma^2(\boldsymbol{\alpha}_k;i^*)}\right)$;    ▷ Equation 30
        $k \longleftarrow k+1$;
        $\boldsymbol{\alpha}_k \longleftarrow (1-\lambda_{k-1})\boldsymbol{\alpha}_{k-1} + \lambda_{k-1}\mathbf{e}_{i^*}$;    ▷ Equation 20
        $r_k^2 \longleftarrow r_{k-1}^2 \left(1 + \frac{\left(\delta_{k-1}^+\right)^2}{4(1+\delta_{k-1}^+)}\right)$;    ▷ (Yildirim, 2008, Lemma 3.2)
    else
        $\lambda_k \longleftarrow \min\left\{\frac{\delta_k^-}{2(1-\delta_k^-)}, \frac{(\boldsymbol{\alpha}_k)_{j^*}}{1-(\boldsymbol{\alpha}_k)_{j^*}}\right\}$;    ▷ Equation 31
        $k \longleftarrow k+1$;
        $\boldsymbol{\alpha}_k \longleftarrow (1+\lambda_{k-1})\boldsymbol{\alpha}_{k-1} - \lambda_{k-1}\mathbf{e}_{j^*}$;    ▷ Equation 22
        $r_k^2 \longleftarrow (1+\lambda_{k-1})r_{k-1}^2 - \lambda_{k-1}(1+\lambda_{k-1})(\delta_{k-1}^- - 1)r_{k-1}^2$;
    end if
    $\mathcal{I}_k \longleftarrow \{i \in \mathcal{I} : (\boldsymbol{\alpha}_k)_i > 0\}$;    ▷ Section 3.3.3
    $R_k \longleftarrow \sum_{i,j\in\mathcal{I}_k} (\boldsymbol{\alpha}_k)_i (\boldsymbol{\alpha}_k)_j \tilde{k}(\tilde{\mathbf{x}}_i, \tilde{\mathbf{x}}_j)$;
    $i^* \longleftarrow \arg\max_{i\in\mathcal{I}} \gamma^2(\boldsymbol{\alpha}_k; i) := \tilde{\Delta}^2 + R_k - 2\sum_{j\in\mathcal{I}_k} (\boldsymbol{\alpha}_k)_j \tilde{k}(\tilde{\mathbf{x}}_j, \tilde{\mathbf{x}}_i)$;    ▷ Equation 25
    $j^* \longleftarrow \arg\min_{j\in\mathcal{I}_k} \gamma^2(\boldsymbol{\alpha}_k; j)$;    ▷ Equation 26
    $\delta_k^+ \longleftarrow \frac{\gamma^2(\boldsymbol{\alpha}_k;i^*)}{r_k^2} - 1$;    ▷ Equation 32
    $\delta_k^- \longleftarrow 1 - \frac{\gamma^2(\boldsymbol{\alpha}_k;j^*)}{r_k^2}$;    ▷ Equation 33
end while
return $\mathcal{I}_S = \mathcal{I}_k$, $\boldsymbol{\alpha} = \boldsymbol{\alpha}_k$.
```

```python
import numpy as np

class GaussianKernel:

    def __init__(self, gamma):
        self.radius_sq = 1
        self.gamma_sq = gamma**2

    def __call__(self, *args):
        return self.compute(*args)

    def compute(self, x_rows, x_cols):
        distances = np.square(
            np.linalg.norm(x_rows, axis=1, keepdims=True)
        ) + np.square(
            np.linalg.norm(x_cols, axis=1, keepdims=True)
        ).T - 2 * x_rows @ x_cols.T
        distances = np.exp(-self.gamma_sq*distances)
        return distances
```

Listing 1: SVM Kernel Implementation (`svm_kernel.py`)

```
import numpy as np
from utils import expand_dims

def kronecker_delta(idx_rows, idx_cols):
    idx_rows, idx_cols = expand_dims(idx_rows, idx_cols)
    kd_matrix = (idx_rows.T == idx_cols).astype(int)
    return kd_matrix

class MEBKernel:

    def __init__(self, svm_kernel, C):
        self.svm_kernel = svm_kernel
        self.C = C
        self.radius_sq = svm_kernel.radius_sq + 1 + 1/C

    def __call__(self, *args):
        return self.compute(*args)

    def compute(self, idx_rows, idx_cols):
        y_rows, y_cols = expand_dims(S[idx_rows, -1], S[idx_cols, -1])
        kernel_matrix = (y_rows.T*y_cols) \
            * (self.compute_svm_kernel(idx_rows, idx_cols) + 1) \
            + kronecker_delta(idx_rows, idx_cols) / self.C
        return kernel_matrix

    def compute_svm_kernel(self, idx_rows, idx_cols):
        x_rows, x_cols = expand_dims(S[idx_rows, :-1], S[idx_cols, :-1])
        return self.svm_kernel(x_rows, x_cols)

    def init_coreset(self, indices):
        self.coreset_precomputations = self.compute(
            np.arange(len(S)), indices
        )

    def add_precomputations(self, index):
        coreset_size = self.coreset_precomputations.shape[1]
        self.coreset_precomputations = np.hstack((
            self.coreset_precomputations,
            self.compute(np.arange(len(S)), [index])
        ))

    def remove_precomputations(self, index):
        coreset_size = self.coreset_precomputations.shape[1]
        self.coreset_precomputations = \
            np.delete(self.coreset_precomputations, [index], 1)
```

Listing 2: MEB Kernel Implementation (`meb_kernel.py`)

```python
import numpy as np
from utils import expand_dims, evaluate

class ModifiedFW:

    def __init__(self, meb_kernel):
        self.meb_kernel = meb_kernel

    def initialize(self):
        pos_indices, = np.where(S[:, -1] == +1.)
        neg_indices, = np.where(S[:, -1] == -1.)
        distance_matrix = self.meb_kernel.compute_svm_kernel(
            pos_indices, neg_indices
        )
        _j, _k = np.unravel_index(
            np.argmax(distance_matrix), distance_matrix.shape
        )
        j, k = int(pos_indices[_j]), int(neg_indices[_k])
        coreset = [j, k]; self.meb_kernel.init_coreset(coreset)
        alpha = np.zeros(len(S)); alpha[j] = alpha[k] = 0.5
        return coreset, alpha

    def optimize(self, eps):

        def compute_R():
            non_zero_duals, = expand_dims(alpha_k[coreset_k])
            assert non_zero_duals.shape == (1, len(coreset_k))
            R = non_zero_duals \
                @ self.meb_kernel.coreset_precomputations[coreset_k] \
                @ non_zero_duals.T
            return R.squeeze()

        def search_directions():
            non_zero_duals, = expand_dims(alpha_k[coreset_k])
            gamma_sq = (self.meb_kernel.coreset_precomputations \
                @ non_zero_duals.T).squeeze()
            i_star = np.argmin(gamma_sq)
            gamma_sq_i_star = delta_sq+R_k-2*gamma_sq[i_star]
            delta_plus = gamma_sq_i_star/r_k_sq - 1.
            j_star = coreset_k[np.argmax(gamma_sq[coreset_k])]
            gamma_sq_j_star = delta_sq+R_k-2*gamma_sq[j_star]
            delta_minus = 1. - gamma_sq_j_star/r_k_sq
            return i_star, delta_plus, j_star, delta_minus

        k = 0
        coreset_k, alpha_k = self.initialize()
        delta_sq = self.meb_kernel.radius_sq
        R_k = compute_R()
        r_k_sq = delta_sq - R_k
        assert r_k_sq > 0.
        i_star, delta_plus, j_star, delta_minus = search_directions()

        dual_iterates, dual_evaluations = [alpha_k], [r_k_sq]
        coreset_sizes, step_sizes = [2], list()

        while 1+delta_plus > (1+eps)**2:

            coreset_sizes.append(coreset_sizes[-1])

            if delta_plus >= delta_minus:
                if i_star not in coreset_k:
                    coreset_k.append(i_star)
                    self.meb_kernel.add_precomputations(i_star)
                    coreset_sizes[-1] += 1
                lambda_k = delta_plus/(2*(1+delta_plus))
                step_sizes.append(lambda_k)
                k += 1
                e_star = np.zeros_like(alpha_k); e_star[i_star] = 1.
                alpha_k = (1-lambda_k)*alpha_k + lambda_k*e_star
                r_k_sq = r_k_sq * (1 + (delta_plus**2)/(4*(1+delta_plus)))

            else:
                best_step = lambda_k = delta_minus / (2*(1-delta_minus))
                max_feasible = alpha_k[j_star] / (1-alpha_k[j_star])
                if lambda_k > max_feasible:
                    lambda_k = max_feasible
                    to_remove = coreset_k.index(j_star)
                    coreset_k.pop(to_remove)
                    self.meb_kernel.remove_precomputations(to_remove)
                    coreset_sizes[-1] -= 1
                step_sizes.append(-lambda_k)
                k += 1
                e_star = np.zeros_like(alpha_k); e_star[j_star] = 1.
                alpha_k = (1+lambda_k)*alpha_k - lambda_k*e_star
                r_k_sq = (1+lambda_k) * r_k_sq \
                    * (1-lambda_k*(delta_minus-1))

            dual_iterates.append(alpha_k)
            dual_evaluations.append(r_k_sq)

            R_k = compute_R()
            i_star, delta_plus, j_star, delta_minus = search_directions()

        return dual_iterates, dual_evaluations, coreset_sizes, coreset_k, step_sizes
```

Listing 3: Modified Frank Wolfe for MEB Problem Implementation (`modified_fw.py`)