Optimization project Ferrero Pizzolato

Ferrero Elisa, Pizzolato Andrea

12^{th} July 2023

1 Introduction

In this paper, we want to analyze how the Frank-Wolfe method (FW) for constrained optimization can be utilized for a machine learning task like the Multi-Layer Clustering (MLC), i.e. a graph consisting of many layers all with the same nodes that need to be clustered. Multi-Layer graphs can be used for a lot of real world applications like transport (where we need to consider all the possible vehicles), social relationships and working relationships, such that there are qualitatively different roles.
In order to do that, we will firstly analyze the FW method and its weaknesses, then introduce the Away-Step FW and analyze their convergence rates, since the classic FW has a sub-linear convergence rate for certain problems while its variants all have a linear convergence rate.
Later on we will explain in detail the MLC problem and how to deal with noise layers (using them for improving the algorithm performances for instance), the introduce the datasets we will use for training and testing of our model; then we will report some results regarding the loss functions and the running time of the MLC problem.

2 Frank-Wolfe method

The Frank-Wolfe method is a popular algorithm for constrained convex optimization. We consider as the set of feasible points the set $\mathcal{M} = \text{conv}(\mathcal{A})$, where $\mathcal{A} \subseteq \mathbb{R}^d$ is a *finite* set of atoms and we look for

$$\min_{x \in \mathcal{M}} f(x) \tag{1}$$

with $f : \mathbb{R}^d \to \mathbb{R}$ a convex function with Lipschitz Continuous Gradient (LCG) with constant L.
It only requires a Linear Minimization Oracle (LMO) in order to find the minimizer along the possible directions (which are the atoms).
To do that, we could use classic gradient descent to update the candidate solution at each iteration, but this would be too costly in a setting like multi-layer graphs. Instead, we will estimate the gradient using the incremental ratio as

estimate, in order to find the direction for minimization along the set of feasible atoms.This decision for the algorithm makes this method a Zero-th Order method, thus the name ZOFW (it is zero-th order because it does not compute the gradient for the objective function).
Another important concept is the *FW gap*

$$g_t^{FW} = \langle -\nabla f(x), d_t \rangle$$

that is an upper bound for the error due to the convexity of the problem [see LJ15, page 4], and this holds for all FW methods.
The key point for FW is the concept of *active set* $\mathcal{S}^{(t)}$, which is the set of atoms explored at iterations previous than t with positive weight at iteration t. At iteration t, the candidate can be written as a convex combination of at most $t+1$ active atoms, belonging to $\mathcal{S}^{(t)}$, in a form like

$$x^{(t)} = \sum_{v \in \mathcal{S}^{(t)}} \alpha_v^{(t)} v$$

Then the algorithm looks for the vector $s_t = \text{argmin}_{v \in \mathcal{A}} \langle \nabla f(x), v \rangle$, then defines the *FW direction* $d_t^{FW} = s_t - x^{(t)}$, then updates the candidate doing a linear search for the appropriate stepsize and calculates the new weight for the complex combination. It is not necessary to keep track of the active set at each iteration for the classic FW, but its variants need it necessarily.

FW presents a huge problem in the case the optimal solution x^* lies near the border of the domain $\mathcal{M}$: when the candidate gets near the borders, it occurs a zig-zagging phenomenon, and the solution oscillates a lot before getting to x^*, as we have seen during the course. This phenomenon slows down heavily the classical FW to a sub-linear convergence rate [see LJ15, page 3].
To deal with this huge problem, some variants of Frank-Wolfe have been proposed

2.1 Away-Step FW method

In this paper we will analyze in depth only the ***Away-Step FW***, a method that tries to solve the zig-zagging problem of FW by considering the so-called *away step*, which is an alternative direction to the FW descent direction, and at each iteration it needs to be chosen one of the two for that iteration's upgrade. This modification is sufficient to make the algorithm linearly convergent for strongly convex functions.
This is done by computing the quantity $v_t \in \text{argmax}_{v \in \mathcal{S}^{(t)}} \langle -\nabla f(x), v \rangle$(this is the atom in the active set that maximizes the potential gradient descent) and the defining the *away direction* $d_t^A = x^{(t)} - v_t$, then the algorithm decides which direction to take with the following rule: it takes d_t^{FW} if

$$\langle -\nabla f(x), d_t^{FW} \rangle \geq \langle -\nabla f(x), d_t^A \rangle$$

and the line search for the exact step size is done in the interval $[0, \gamma_{max}]$ with $\gamma_{max} = 1$. Otherwise the algorithm chooses d_t^A with $\gamma_{max} = \frac{\alpha_{v_t}}{1-\alpha_{v_t}}$. This variation in the line search interval is due to the necessity of the point belonging to the domain [LJ15, page 3]. In case the minimizer $\gamma_t = \text{argmin}_{\gamma \in [0,\gamma_{max}]} f(x^{(t)} + \gamma d_t)$ is equal to $\gamma_t = \gamma_{max}$ the step is called a *drop step*, since one atom is dropped from the active set and gets its weight set to 0.

For a FW step, we set $\mathcal{S}^{(t+1)} = s_t$ if $\gamma_t = 1$, otherwise $\mathcal{S}^{(t+1)} = \mathcal{S}^{(t)} \cup s_t$; the weights update is $\alpha_{s_t}^{(t+1)} = (1 - \gamma_t)\alpha_{s_t}^{(t)} + \gamma_t$ and $\alpha_v^{(t+1)} = (1 - \gamma_t)\alpha_v^{(t)}$ for $v \in \mathcal{S}^{(t)} \setminus s_t$.
For an away step we have that $\mathcal{S}^{(t+1)} = \mathcal{S}^{(t)} \setminus v_t$ if it is a drop step, otherwise $\mathcal{S}^{(t+1)} = \mathcal{S}^{(t)}$ and the weights update is $\alpha_{v_t}^{(t+1)} = (1 + \gamma_t)\alpha_{v_t}^{(t)} - \gamma_t$ and $\alpha_v^{(t+1)} = (1 + \gamma_t)\alpha_v^{(t)}$.

2.2 Other FW variants

There are other FW variants that we are not going to use in these projects but that are interesting in the way they deal with the sub-linearity problem. We will just explain the main idea for each of them [LJ15, pages 3 and following].

Pairwise FW The Pairwise FW method is an evolution of the Away Step FW that at each step only changes the weights between the away step atom v_t and the FW atom s_t at each iteration. It is based on moving a certain mass from the away atom to the FW atom up to when $\gamma = \alpha_{v_t}$, in which case we talk about a *away step*. The algorithm requires all active atoms to be ordered w.r.t. the scalar product with $\nabla f(x^{(t)})$ (so that the away atom is the first to be considered for the swap) and the mass exchange is done following this ordinament of the active atoms.
In order to have an adaptive algorithm the algorithm applies a line search in the *Pairwise FW direction* $s_t - v_t$ and this correction is enough to ensure a global linear convergence rate.
Although a theoretical analysis is difficult due to the unpredictability of the presence of swap steps, this method works really well in practice, often outperforming Away Step FW: due to the reduced number of updates of the weights in away steps it reduces wasted resources, instead of updating uniformly all weights (that may need corrective updates in a second moment) it updates only the "good" FW direction.

Fully corrective FW This algorithm is used when the call to the linear oracle becomes too costly, and is based on avoiding to do an exact line search at each iteration, substituting it with a re-optimization of the function on a set $conv(\mathcal{A}^{(t)})$, where $\mathcal{A}^{(t)} \supseteq \mathcal{S}^{(t)}$ is a set containing the active atoms and other more. At each iteration the candidate solution is chosen as an element in $conv(\mathcal{A}^{(t)})$ that improves at least as much as the FW iteration and such that the gap

$g^A_{t+1} = \max_{v \in \mathcal{S}^{(t+1)}} \langle -\nabla f(x^{(t+1)}), x^{(t+1)} - v \rangle \leq \varepsilon$
is small enough for the parameter chosen as stopping criterion.
For this algorithm it is critical the choice of the correction set $\mathcal{A}^{(t)}$ that must have enough directions to find a suitable solution. It can be shown [see LJ15, page 5] that this method has a linear convergence rate that does not require an exact solution for the correction and can be proved just with weaker estimates.

Wolfe's min-norm point algorithm This method is a variation of the previous FCFW based on a sequence of affine projections that potentially yield multiple updates but in practice performs really well on a vaste class of problems [see LJ15, page 4]

3 Convergence of FW methods

It is of crucial importance for our analysis the study of the convergence rate of the FW method and of its variants. The convergence rate for the classic FW is linear for optimal solutions x^* distant from the borders , whilst the rate becomes sub-linear due to the zig-zagging of the solutions. It has been shown that the variants introduced before have all a linear convergence rate, thus explaining why they are really interesting for lots of research fields, especially for our problem: this is due to the resolution of the zig-zagging, that all variants face trying either to avoid useless steps or by using tricks like away steps or drop steps.
We will understand now what is the cause of the sublinearity of classical FW from a mathematical point of view: we assume our objective function f to be μ-strongly convex and to have Lipschitz continuous gradient with constant L over a compact set $\mathcal{M}$ such that $M = \text{diam}(\mathcal{M})$. Given a vector v we write $\hat{v}$ to call the vector normalized. We call $h_t = f(x^{(t)}) - f(x^*)$ the suboptimality error and $r_t = -\nabla f(x^{(t)})$.
We consider the direction given by the FW algorithm d_t, by the standard descent lemma we have

$$f(x^{(t+1)}) \leq f(x^{(t)} + \gamma d_t) \leq f(x^{(t)} + \gamma \langle -r_t, d_t \rangle + \frac{\gamma^2}{2} L ||d_t||^2 \tag{2}$$

for each $\gamma \in [0, \gamma_{max}]$. By taking $\gamma = \frac{\langle r_t, d_t \rangle}{L||d_t||^2}$ and rearranging terms we get

$$h_t - h_{t+1} \geq \frac{1}{2L} \langle r_t, \hat{d}_t \rangle^2$$

We also define the error on the domain $e_t = x^{(t)} - x^*$ and by strong convexity

$$f(x^{(t)} + \gamma e_t) \geq f(x^{(t)} + \gamma \langle \nabla f(x^{(t)}, e_t \rangle + \frac{\gamma^2}{2} \mu ||e_t||^2 \tag{3}$$

for each $\gamma \in [0,1]$. We now take the minimum in the RHS ($\gamma = \frac{\langle r_t, e_t \rangle}{\mu ||e_t||^2}$) and $\gamma = 1$ in the LHS thus getting

$$h_t \leq \frac{\langle r_t, e_t \rangle^2}{2\mu}$$

and combining with the previous result we get

$$h_t - h_{t+1} \leq \frac{\mu}{L} \frac{\langle r_t, \hat{d}_t \rangle^2}{\langle r_t, \hat{e}_t \rangle^2} h_t$$

If the optimal solution lies at a distance $\delta > 0$ from the borders the numerator of the coefficient on the RHS can be bound as $\langle r_t, \hat{d}_t \rangle \leq \delta ||d_t||$, then remembering that M is the diameter of the domain we get a linear rate of convergence with constant $1 - \frac{\mu}{L}(\frac{\delta}{M})^2$. But when the optimal solution is on the border the scalar product above cannot be bounded well and the rate becomes sub-linear.
In the case of the away-step FW, in order to prove the linear convergence it is fundamental to consider the *pairwise FW direction*, defined as $d_t^{PFW} = s_t - v_t$. Using this we can write

$$2\langle r_t, d_t \rangle \geq \langle r_t, d_t^A \rangle + \langle r_t, d_t^{FW} \rangle \geq \langle r_t, d_t^{PFW} \rangle \Longrightarrow \langle r_t, d_t \rangle \geq \frac{\langle r_t, d_t^{PFW} \rangle}{2}$$

The key point of the proof for the Away Step FW is that the quantity $\langle \hat{r}_t, d_t^{PFW} \rangle$ can be lower bounded just by using a quantity depending on the geometry of the domain $\mathcal{M}$, the *pyramidal width of* $\mathcal{A}$ [LJ15, Appendix B]. This implies that when γ_{max} is too small the algorithm performs a drop step and here we do not know how big is the improvement, but these steps cannot happen more than half the time.
We can now state the theorem that describes how the ASFW has linear convergence rate.

Theorem Suppose that f has L-continuous gradient and is μ-strongly convex over $\mathcal{M} = conv(\mathcal{A})$, with $M = \text{diam}(\mathcal{M})$ and δ is the pyramidal width of $\mathcal{A}$. Then for every step that is not a drop step($\gamma_t < \gamma_{max}$), the ASFW method is such that

$$h_{t+1} \leq (1 - \rho) h_t$$

with $\rho = \frac{\mu}{4L}(\frac{\delta}{M})^2$. If we call $k(t)$ the number of steps that are not drop steps, we know that $k(t) \geq t/2$ and we can state that

$$h_t \leq h_0 exp(-\rho k(t))$$

thus we have linear convergence.
Beyond that, the FW gap can be bounded with the error

$$g_t^{FW} \leq h_t + \frac{LM^2}{2} \qquad \text{when } h_t > LM^2/2$$
$$g_t^{FW} \leq M\sqrt{2h_t L} \qquad \text{otherwise}$$

The proof of this theorem descends from the geometric results and from calculus manipolations on all the quantities and it's quite straightforward.[see LJ15, page 13]

Pyramidal width It is interesting to highlight the main concepts used to define what is the pyramidal width of the convex envelope of a (finite) set. Since $\langle r_t, d_t^{PFW} \rangle = \langle r_t, s_t - v_t \rangle = \max_{s \in \mathcal{M}, v \in \mathcal{S}^{(t)}} \langle r_t, s - v \rangle$, this quantity looks like the directional width of a pyramid with base $\mathcal{S}^{(t)}$ and summit s_t. Working on this intuition we can define the following elements [LJ15, page 7].

- the *directional width* of a set $\mathcal{A}$ w.r.t. a direction r is defined as $dirW(\mathcal{A}, r) = \max_{s,v \in \mathcal{A}} \langle \frac{r}{||r||}, s - v \rangle$, and its *width* can be defined as the minimum directional width over directions in the affine hull of the set
- we define the *pyramidal directional width* of a set w.r.t. a direction r and a point $x \in \mathcal{M}$ as $PdirW(\mathcal{A}, r, x) = \min_{s \in \mathcal{S}_x} dirW(\mathcal{S} \cup \{s(\mathcal{A}, r)\}, r)$ where $s(\mathcal{A}, r) = \text{argmax}_{v \in \mathcal{A}} \langle r, v \rangle$ and $\mathcal{S}_x$ is the set of all possible subsets of $\mathcal{A}$ that have x as a proper combination of their elements.
- The *pyramidal width* is the minimum over the cone of possible feasible direction of the DPW, i.e. directions pointing inward towards $\mathcal{A}$ from x

$$PwidthA = \min_{\mathcal{K}, x, r} PdirW(\mathcal{K} \cap \mathcal{A}, r, x) \tag{4}$$

  with $\mathcal{K}$ belonging to the set of faces of $conv(\mathcal{A})$, $x \in \mathcal{K}$, r in the cone created by $\mathcal{K}$ and x excluding the null vector.

With these definitions and using the notations defined above we can state the following theorem

Theorem Given all the previous definitions, it holds that

$$\frac{\langle r_t, \hat{d}_t \rangle}{\langle r_t, \hat{e}_t \rangle} \geq PwidthA$$

The more interesting aspect of this theorem is that it helps understanding the constant appearing in the convergence rate theorem.

$$\frac{1}{\rho} = 4 \frac{L}{\mu} (\frac{M}{\delta})^2$$

The first term is the condition number of the function f, while the second term is a sort of condition number for the domain called also *eccentricity* of the domain, that is smaller when the pyramidal width is similar to the diameter of the domain. This means that the convergence is faster when this eccentricity is smallest, which is when the set is the regular simplex, which has eccentricity $d/2$ (d is the dimension where the domain lives in), while ASFW becomes slower

when the diameter is large compared to the pyramidal width.

After this theoretical analysis on the FW algorithm and its variants we are going to analyze the MLC problem and see how to use these algorithms to solve it.

4 Definition of the problem: Multi-Layer Clustering (MLC)

We now introduce the main problem of this paper, the Multi-Layer Clustering Problem. It is a semi-supervised learning problem on graph. We consider a graph with more than one layer, i.e. a structure that has the same nodes at each layer but with different set of edges at each layer. The key goal we have is to understand how to use the multiple information sets given by the edges in order to correctly cluster the nodes, using as data a small fraction of labelled nodes and trying to predict the correct labels for the remaining nodes.
One of the main challenges with multilayer graphs is that it is not obvious which layers possess relevant information for the clustering, which are useless and which layers can be interpreted as noise layers, so it is imperative to find a good model that is able to discern the information it gets from the various layers.
The idea that has been used in [Ven+23] is to build an algorithm that aggregates the layers with a non-linear generalized mean function with parameters learned through training by an inexact bilevel gradient optimization scheme.
After that, a standard clustering technique like spectral clustering can be used to cluster the aggregated layers.

Formulation We define a multilayer graph as $\mathbf{G} = (G^1, G^2, ..., G^k)$ where each $G^i = (V, E^i, w_i)$ with $V = \{1, 2, ..., N\}$, $E^i \subseteq V \times V$ and $w_i : E^i \to \mathbb{R}^+$ is the weight function for layer i, that can be represented also as an adjacency matrix A^i. We assume that $\mathbf{G}$ is formed by a set of communities $C = \{C_1, C_2, ..., C_m\}$ such that each node belongs to a community, all communities are disjoint and for every layer the node belongs to the same community. Further we assume that for each $C_j \in C$ we know a set of labelled nodes $O_j \subseteq V$ which is supposedly a small fraction of the nodes and such that it is one-hot encoded into a matrix $Y \in \mathbb{R}^{n \times m}$ such that $Y_{ij} = 1$ if node i belongs to community C_j, $Y_{ij} = 0$ otherwise.
The goal is to minimize a standard regularized laplacian-based loss function, which is a generalization of the one-layer loss

$$\varphi(X) = ||X - Y||_F^2 + \frac{\lambda}{2} Tr(X^T L X)$$

where $X \in \mathbb{R}^{n \times m}$ is the argument of the minimization, $L = D - A$ is the graph laplacian and λ is an hyperparameter that needs to be learned. In order to build the correct model of loss function we need to define the *generalized mean*

adjacency matrix which is

$$A(\alpha,\beta)_{ij} = \left(\sum_{k=1}^{K} \beta_k (A_{ij}^k)^\alpha\right)^{1/\alpha}$$

where $\sum_k \beta_k = 1,\ \beta_k > 0$ and $\alpha \in \mathbb{R}$. This generalized mean is a really powerful tool since it can model both the weights for each layer (the vector $\beta = \{\beta_1, ..., \beta_k\}$) and by tuning the parameter α we can model the functional behaviour of the mean.
It is clear from the definition that this generalized mean comprehends some other aggregation functions, like the simple mean ($\alpha = 1, \beta_k = 1/K$), the maximum ($\alpha \to +\infty$) and the minimum ($\alpha \to -\infty$), and so it is clear that choosing this model has a better potential to correctly fit the data.
With this new function we can define also a matrix

$$D(\alpha,\beta) = \text{diag}(A(\alpha,\beta)\mathbb{I}_n)$$

and also a generalized graph laplacian

$$L(\alpha,\beta) = D(\alpha,\beta) - A(\alpha,\beta)$$

With this definitions we can define our loss function

$$\varphi(X;\alpha,\beta,\lambda) = ||X - Y||_F^2 + \frac{\lambda}{2}(X^T L(\alpha,\beta)X) \tag{5}$$

The problem could also be analyzed layer-wise by considering the loss function as a sum of K sub functions defined on each layer

$$\varphi_k(x,y,\alpha,\beta,\lambda) = \textstyle\sum_i |x_i - y_i^k|^2 + \frac{\lambda}{2}\sum_{i,j} A_{ij}(\alpha,\beta)(x_i - x_j)^2$$

and since they are all independent they can be optimized independently from the others [Ven+23, page 3].
In order to learn the set of parameters $\theta{:=}(\alpha,\beta,\lambda)$ we split the available labels into training set and test set, with associated matrices Y^{tr}, Y^{te} and consider the bilevel optimization model

$$\begin{array}{ll} \min & H(Y^{te}, X_{Y^{tr};\theta}) \\ s.t. & X_{Y^{tr};\theta} = \text{argmin}_X \varphi(X,Y,\theta) \\ & \alpha \in \mathbb{R}, \beta \geq 0, \sum_k \beta_k = 1,\ \lambda \in \mathbb{R} \end{array}$$

where H i the multiclass cross entropy loss function

$$H(Y,X) = -\frac{1}{N}\sum_{i=1}^{N}\sum_{j=1}^{N} Y_{ij} \log\left(\frac{X_{ij}}{\sum_{i=1}^{N} X_{ij}}\right)$$

After all these operations, the final embeddings of the nodes have been aggregated in a single layer that can be clustered using traditional techniques such as

spectral clustering, that is based on the graph laplacian of the obtained graph. Firstly, let's look the fact that when fixing the parameters' set θ the solution to the argmin problem above can be solved exactly and gives the result

$$\text{argmin}_X \varphi(X, Y; \theta) = (\mathbb{I}_n + \lambda L(\alpha, \beta))^{-1} Y$$

If we put this result together with the minimization problem as exposed in 4, we can rewrite it as

$$\begin{array}{ll} \min_{\theta \in S} & f(\theta) \\ f(\theta) = & H(Y^t e, (\mathbb{I}_n + \lambda L(\alpha, \beta))^{-1} Y^{tr}) \end{array}$$

so the problem now is to determine the best hyper parameters.

5 Use of FW methods for MLC

We need to find the best choice of hyper parameters and we can use a FW method to optimize their choice into the domain space. The parameters live in a space which has dimension $K + 2$, and in order to find them we need to determine what values we search.
The domain is

$$S = \{(\alpha, \beta, \lambda) \in \mathbb{R}^{K+2} \mid \alpha \in [-a, a],\ \sum_k \beta_k = 1,\ \beta_k > 0,\ \lambda \in [l_0, l_1]\}$$

- α belongs to an interval $[-a, a], a \in \mathbb{R}$ since the behaviour of the generalized mean adjacency matrix as a function of α quickly approaches the maximum (minimum) which is the limit on the value of α going to $+\infty$ $(-\infty)$, thus making it useless to look on a very wide interval
- λ belongs to an interval $[l_0, l_1]$ with $l_0, l_1 \in \mathbb{R}$ because it is a penalty term and thus it cannot grow too big
- $\beta = \{\beta_1, ..., \beta_k\}$ is a set of weights, so that all these parameters belong to $(0, 1)$ and sum to 1.

This choice means that the domain of the function f as defined above is a box-plus-simplex, a particularly simple structure for optimization purposes (it's clearly convex and compact), and so the problem can be solved separately in the three types of parameters.
Instead of computing the exact gradient that could become very costly due to the high number of terms to sum, we can use an approximate gradient

$$\tilde{\nabla} f(\theta_n) = \sum_{i=1}^{K+2} \frac{f(\theta_n + h_n e_i) - f(\theta_n)}{h_n} e_i$$

by using a finite difference scheme, with h_n a small enough value. This is the pseudocode of the algorithm

***Algorithm 1**: FW method for the choice of best parameters for aggregation*

1. Given $\theta_0 \in S$
2. **For** $n = 0, 1, ...$:
3. Compute $\tilde{\nabla} f(\theta_n)$ as an estimate of the real gradient
4. Compute $\hat{\theta}_n \in \text{argmin}_{\theta \in S} \tilde{\nabla} f(\theta)^T (\theta - \theta_n)$
5. Set $d_n = \hat{\theta}_n - \theta$
6. Compute a stepsize $\eta_n \in (0, 1]$ by a line search
7. Set $\theta_{n+1} = \theta_n + \eta_n d_n$
8. **End For**

At this point the structure of the domain helps us, because w.r.t. α we are minimizing a linear function over an interval in the real numbers, so we have that $\hat{\alpha}_n = a$ if $\tilde{\nabla}_\alpha f(\theta_n) < 0$, otherwise $\hat{\alpha}_n = -a$. In the same way we are minimizing a linear function over λ so the best values are on the borders of the interval at each iteration, thus $\hat{\lambda}_n = l_0$ if $\tilde{\nabla}_\lambda f(\theta_n) > 0$. For what concerns the simplex domain of β, it is such that the candidate at each iteration is $\beta_n = e_{\hat{j}}$ with $\hat{j} = \text{argmin}_j [\tilde{\nabla}_\beta f(\theta_n)]_j$ [see Ven+23, pages 4, 5]

5.1 Convergence analysis

In order to analyze the convergence of this algorithm we need some notations to describe the various FW gaps we use to estimate the error

- $g_n = -\nabla f(\theta_n)^T d_n$ is the exact gap
- $\tilde{g}_n = -\tilde{\nabla} f(\theta_n)^T d_n$ is the gap computed with the inexact gradient
- $g_n^{FW} = -\nabla f(\theta_n)^T d_n^{FW}$ where $d_n^{FW} \in \text{argmin}_{\theta \in S}(-\nabla f(\theta_n)^T(\theta - \theta_n)) - \theta_n$ is the direction given by the FW algorithm with exact gradient.

We do an assumption [see Ven+23, page 5, Assumption IV.1] on the error committed by the inexact gradient approximation

Assumption For every n, there exists $\varepsilon_n > 0$ such that

$$|(\nabla f(\theta_n) - \tilde{\nabla} f(\theta_n))^T (\theta - \theta_n)| \leq \varepsilon_n \qquad \forall \theta \in S$$

Since S is a convex set a point a point θ^* in it is *stationary* if

$$\nabla f(\theta^*)^T (\theta - \theta_n) \geq 0 \quad \forall \theta \in S$$

. It is clear then that the FW gap g_n^{FW} is an optimality measure for the function f. We can state a sub-linear convergence rate theorem for the simple FW method applied in this case.

Theorem Let f be a Lipschitz continuous gradient function with constant M, and let S be a compact set with finite diameter Δ, and let $\{\theta_n\}$ be a sequence generated by *Algorithm 1* with the inexact gradient satisfying the previous assumption with

$$\varepsilon_n \leq \frac{\sigma}{1+\sigma}\tilde{g}_n \qquad\qquad 0 < \sigma < \frac{1}{3}$$

Also let the stepsize be bounded in the following way

$$\eta_n \geq \bar{\eta}_n = \min(1, \frac{\tilde{g}_n}{M||d_n||^2})$$
$$f(\theta_n) - f(\theta_n + \eta_n d_n) \geq \rho\bar{\eta}_n\tilde{g}_n$$

for some $\rho > 0$. Under these hypotheses, defining f^* as the minimum of f in S and g_n^* the minimum FW gap up to iteration n, the following inequality holds

$$g_n^* \leq \max\left(\sqrt{\frac{(\Delta^2 M)(f(\theta_0) - f^*)}{n\rho(1-\sigma)^2}}, \frac{2(f(\theta_0) - f(\theta^*)}{n(1-3\sigma)}\right) \tag{6}$$

This theorem gives us a bound on the FW gap relating it to the initial error and decreasing on the number of iterations (going either as the inverse of n or of its square root). Its proof is based on a series of inequalities and on the standard descent lemma, combined with the division in various cases (that results in the max and min conditions in the text). It can be proved that using an ASFW method on this algorithm results in a linear convergence rate as a result.

For what concerns an appropriate stepsize we can use the following lemma

Lemma Given the assumption 5.1 above with the condition $0 < \sigma < 1/2$, if we consider an Armijo line search with

$$\eta_n = \delta^j$$

with j the minimum integer such that

$$f(\theta_n) - f(\theta_n + \eta_n d_n) \geq \gamma\eta_n\tilde{g}_n$$

with $\gamma \in (0, 1/2)$ and $\delta \in (0, 1)$ two parameters fixed at the beginning: then we have that

$$\eta_n \geq \min(1, 2\delta(1-\gamma-\sigma))\bar{\eta}_n \tag{7}$$

5.2 Implementation details

As described above, the gradient of f has been approximated with a finite difference, depending on the parameter h_n: in order to have a higher precision

estimation of the gradient in the later steps (where it is more crucial to be accurate in choosing the direction), we have set $h_0 = 10^{-4}$ and $h_n = \frac{h_{n-1}}{2}$ so that we use smaller steps thus approximating the gradient better. The stopping criterion for the algorithm is when the approximated FW gap gets smaller than a tolerance $\tilde{g}_n \leq \tau = 10^{-4}$.
The iterative parametric label propagation we used is based on the following recursive rule

$$X^{(r+1)} = \lambda A(\alpha, \beta)(\mathbb{I}_n + \lambda D(\alpha, \beta))^{-1} X^{(r)} + (\mathbb{I}_n + \lambda D(\alpha, \beta))^{-1} Y \qquad (8)$$

that converges to the solution of the argmin problem 4. because the spectral radius of $A(\alpha, \beta)(\mathbb{I}_n + \lambda D(\alpha, \beta))^{-1}$ is smaller than 1.
We have set the intervals for the parameters putting $a = 20$, $l_0 = 0.1$ and $l_1 = 10$, which are values that are broad enough for a good learning for the model. Another important feature we need to consider is that we apply the multistart FW version, with a fixed number of starting points (we chose 3 for computational reasons: it was a good compromise between a higher number of starting points (which gives a better final f and the computational time), so that the algorithm can search through different sets of parameters (this is to avoid that the algorithm gets stuck on local minima). Between the three starting points we put the arithmetic mean parameters and the harmonic mean parameters beyond a random initialization, to see if predetermined choices for the aggregation function could be feasible solutions. As it turns out, the main problem with all types of predetermined choices for the aggregation function is that they may perform well on specific datasets but do not generalize well at all and their performance decreases much when dealing with noise layers (this is due to the equi-weighting of the layers in the initialization which is difficult to wipe away with a reasonable amount of learning iterations).
The parameters for the line search for the stepsize have been set to $\delta = 0.75$ and $\gamma = 0.1$ after some tuning on a single dataset to see which parameters performed well.

Noise layers How does the model we built solve the issue of noisy layers (i.e., layers that are not informative with edges relating nodes in different clusters without any precise pattern)? The aggregation parameters learned through FW method are able to discern them and since each of the β_k is the convex coefficient of a layer, it is reduced of importance when it is a noise layer, because it does not improve the performance of the algorithm and is thus penalized in the learning phase.

5.3 Datasets

We tested our model on two real-world datasets:

- *bbcnews*: BBC news articles (685 nodes, 5 communities, 4 layers)
- *cora*: a citations dataset (2708 nodes, 7 communities, 2 layers)

We tried two datasets with different sizes, in order to confirm that the model can work independently from the size of the datasets.
The MLC problem is really interesting when there is only a small fraction of labelled data, since in real world applications it is often unfeasible to possess the labels of a lots of nodes: in order to accomplish this, we tried various percentages of labelled data:
For both the *bbcnews* dataset and the *cora* dataset:

- 1% of labelled data
- 5% of labelled data
- 10% of labelled data
- 15% of labelled data

Since they are real datasets, we also tried to learn the parameters adding some layers of noise, specifically we have run all the tests on both datasets with 0,1 or 2 noisy layers, where each noisy layer is generated following a Bernoulli distribution on the edges (which is that each edge is added with a 50% probability).

6 Analysis of the results

In this section we want to report the numerical analysis of the results of our algorithm. The results for all the combination of noise layers and input labels percentage are in tabular forms, while the graphics refer all to the *cora* dataset, with a 10% of labelled inputs and 2 layers of noise.

6.1 Accuracy

We will firstly report the accuracy we reached for each dataset for all possible combinations of labelled data and number of noise layers. We will also report the set of best parameters for each possible setting (i.e. a combination of input percentage and number of noisy layers).

bbcnews

$ACCURACY$	1%	5%	10%	15%
0 $n.l.$	0.716	0.832	0.841	0.854
1 $n.l.$	0.747	0.731	0.844	0.887
2 $n.l.$	0.674	0.758	0.830	0.861

bbcnews: ASFW method

$ACCURACY$	1%	5%	10%	15%
0 $n.l.$	0.661	0.778	0.796	0.767
1 $n.l.$	0.661	0.758	0.796	0.816
2 $n.l.$	0.767	0.778	0.766	0.816

cora

ACCURACY	1%	5%	10%	15%
0 *n.l.*	0.690	0.759	0.762	0.799
1 *n.l.*	0.538	0.707	0.773	0.797
2 *n.l.*	0.524	0.733	0.755	0.792

cora: ASFW method

ACCURACY	1%	5%	10%	15%
0 *n.l.*	0.611	0.721	0.742	0.788
1 *n.l.*	0.615	0.688	0.773	0.805
2 *n.l.*	0.731	0.747	0.755	0.792

These results show us similar patterns: firstly, when there are very few labelled data (1%) adding noise not only is not a problem, but can also help the model to learn better (as can be seen in the ASFW method applied to the *bbcnews* dataset), since between the random connections there may also be some useful, informative ones: in general the results are better in terms of accuracy for the *bbcnews* dataset, where accuracy reaches almost 90%. The performance is not as good with the ASFW as with the classic FW, especially for the *bbcnews*, while for the other one the performance is comparable between the two methods.
The algorithm performance increases significantly when the label percentage passes from 1% to 5% then the improvements become more moderate. Although the 2 graphs have quite a different structure, the algorithm has roughly similar performances on them both, proving that the model we are using is proficient both on wider graphs like *cora* and on multi-layer graphs with a deeper structure like *bbcnews*, where it deals well with finding the informative layers (see below).
The model is also able to detect well communities when given layers of noise, since the drop in the performance when adding noise is very small, and in some cases there is also an improvement of the performance.

6.2 Runtimes

We will report a similar kind of tabular for the running times of the algorithms. All times are in seconds, and each value represents the average of the runtimes with the three different initializations of the parameters.

bbcnews

RUNTIME	1%	5%	10%	15%
0 *n.l.*	3, 807	0, 179	0, 677	0, 624
1 *n.l.*	0, 413	1, 511	15, 84	3, 501
2 *n.l.*	65, 54	0, 981	00, 285	0, 917

bbcnews: ASFW method

$RUNTIME$	1%	5%	10%	15%
0 $n.l.$	$15,77$	$0,179$	0888	$0,849$
1 $n.l.$	$0,435$	$1,826$	$1,167$	$3,035$
2 $n.l.$	$32,43$	0972	$0,292$	$2,261$

cora

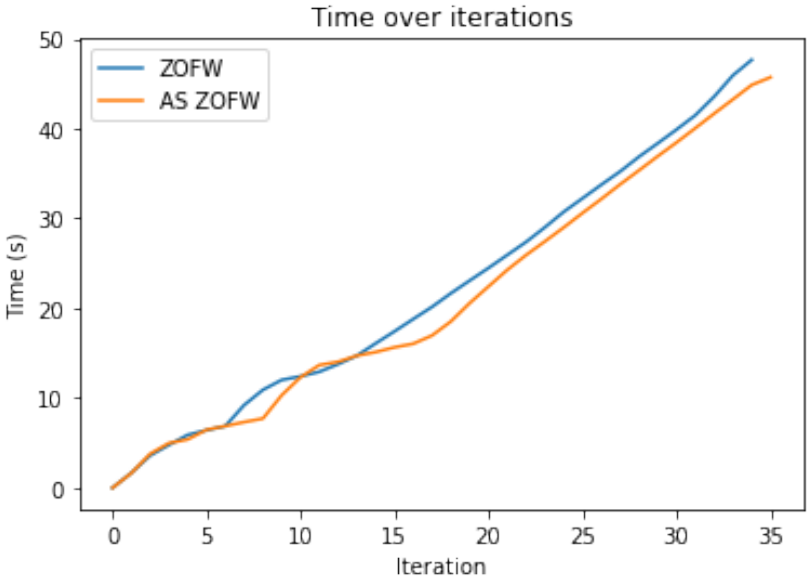
$RUNTIME$	1%	5%	10%	15%
0 $n.l.$	$2,142$	$30,65$	$44,85$	$589,52$
1 $n.l.$	$48,00$	$5,517$	$61,10$	$85,76$
2 $n.l.$	$9,984$	$48,24$	$80,69$	$133,01$

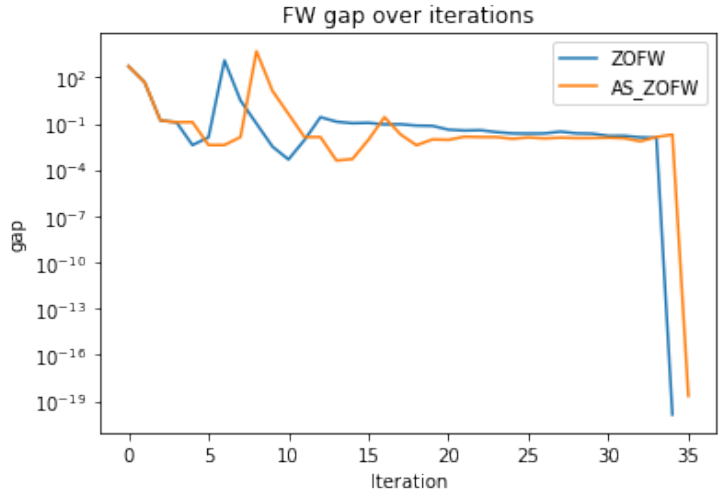
cora: ASFW method

$RUNTIME$	1%	5%	10%	15%
0 $n.l.$	$2,146$	$9,431$	$14,99$	$245,49$
1 $n.l.$	$4,817$	$6,272$	$52,76$	$53,80$
2 $n.l.$	$6,502$	$19,69$	$53,43$	$36,23$

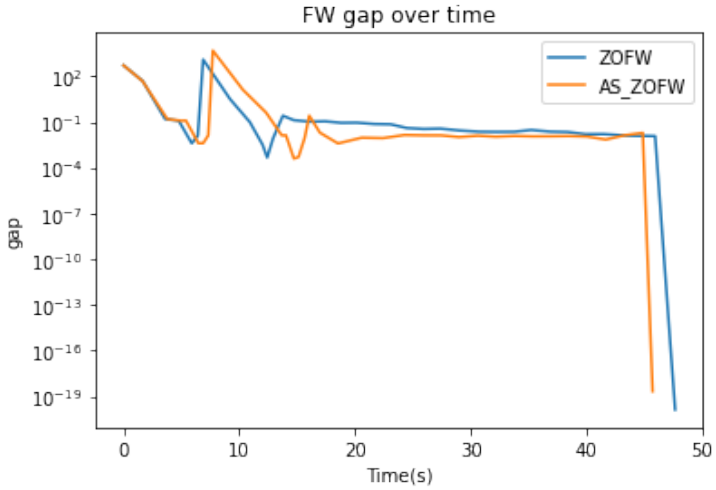
Running time was tested on a single run, starting from a random atom with $\alpha = 20$, $\lambda = 0.1$, and all the weight on a random layer.
There is no specific rule relating the runtime with the specific set of parameters, but this is clearly due to the fact that the initialization of the parameters may be already good enough so that the FW gap is below the tolerance, and there is no need for iterating the algorithm. This is what happens when the time is very small, near 0 seconds, in the results: it means that the initial set of parameters was such that the FW gap was small already and the algorithm stopped immediately.
The fact that the *bbcnews* has very small runtime compared to the *cora* one is due mainly to the smaller number of nodes in each layer of the graph which vastly reduces the total amount of operations needed.
The big time increment when increasing the number of labelled inputs is related to the increment in the dimension of the matrices used by the algorithm related to it (like Y) because the number of operations is related to the product of the number of labelled elements with the number of the remaining ones, and when dealing with a big dataset like *cora* the number of unlabelled elements changes little in a relative way but the number of labelled ones increases by a factor of 5, so the total number of operations increases by about 5 times. The main discriminant remains the initialization however, since for example with 15% of labelled inputs and 2 noisy layers the runtime for the *bbcnews* dataset was only of 2,261 seconds.
We also present a plot of the time required by the algorithm. It is important to note how the iterations of the AS version aren't particularly slower, requiring the same amount of calls to the ZO oracle as the standard FW.

We will also show some plots relative to the FW gap. The gap is plot against iterations firstly and against time lately, with a semilogarithmic scale.





In both cases, the behaviour of the gap is similar, with the algorithm that gets stuck slightly above the tolerance τ and then dropping practically to zero when the final solution is reached. The fact that the behaviour is similar may be due to the fact that for the setting we are considering (*cora* dataset, 10% labelled input, 2 noisy layers) the optimal solution might be internal to the domain (see 3) and we have already shown in 3 that in this case the convergence rate is linear for classic FW method also.The constant can be computed with the formula we have seen at 3.



6.3 Parameters

In this section we will report the optimal values for the set of parameters θ in order to see what is the tendency for their behaviour. In particular, this can be useful to see if the model deals well with layers of noise, since by looking at the weights of the layers if there are sensible variations when adding noise layers then it means that the algorithm has not been able to infer correctly the nature of the layer and ignore it.

	α			β			λ		
	0 n.l.	1 n.l.	2 n.l.	0 n.l.	1 n.l.	2 n.l.	0 n.l.	1 n.l.	2 n.l.
1%	-20	-20	0,004	[0;1;0;0]	[0,433;0;0;0,567;0]	[0,227;0,155;0,155;0,155;0,154;0,154]	0,1	0,1	0,935
5%	0,146	-20	-20	[0,227;0,226;0,281;0,266]	[0;0;1;0;0]	[1;0;0;0;0;0]	0,99	0,1	0,1
10%	0,105	0,164	0,205	[0,256;0,234;0;234;0,276]	[0,185;0,190;0,214;0,226;0,185]	[0,150;0,150;0,209;0,191;0,150;0,150]	1,144	1,218	0,912
15%	0,121	0,219	0,116	[0,275;0,258;0,234;0,233]	[0,235;0,187;0,205;0,187;0,186]	[0,190;0,187;0,170;0,152;0,151;0,150	0,941	1,012	1,443

Figure 1: Parameters for *bbcnews* with FW

In this case, the algorithm keeps similar set weights only for a consistent number of labelled input. This could be due to the fact that with small percentages of labels the algorithm is not able to infer correctly the weight of the layers and the quantity of true information they carry (the 1% of ∼700 nodes is 7 nodes, which are very few to be used to find noisy layers).

The ASFW method applied on the *bbcnews* dataset is really good at avoiding noise layers, since the weights of each layer stay practically the same also when adding noisy layers. It is also interesting to note that the value of α is almost everywhere equal to the minimum possible value, leading us to think that a possible good aggregation function for these problems could be the minimum.

	α			β			λ		
	0 n.l.	1 n.l.	2 n.l.	0 n.l.	1 n.l.	2 n.l.	0 n.l.	1 n.l.	2 n.l.
1%	-20	20	1,997	[1;0;0;0]	[1;0;0;0;0]	[0,999;0,001;0;0;0;0]	0,1	0,1	0,107
5%	-20	-20	-20	[0;0;0;1]	[1;0;0;0;0]	[0;0;0;1;0;0]	0,1	0,1	0,1
10%	-20	-20	-20	[1;0;0;0]	[1;0;0;0;0]	[0;1;0;0;0;0]	0,1	0,1	0,1
15%	-20	-20	-20	[1;0;0;0]	[1;0;0;0;0]	[1;0;0;0;0;0]	0,1	0,1	0,1

Figure 2: Parameters for *bbcnews* with ASFW

Also, the value of λ being almost everywhere equal to the minimum possible value leads to the idea that there is not too much need of regularization in this specific setting.

	α			β			λ		
	0 n.l.	1 n.l.	2 n.l.	0 n.l.	1 n.l.	2 n.l.	0 n.l.	1 n.l.	2 n.l.
1%	-20	-20	0,004	[1;0]	[0;1;0]	[0,352;0,216;0,216;0,216]	0,1	0,1	0,877
5%	-20	0,145	0,587	[1;0]	[0,712;0,288;0]	[0,874;0,122;0,0037;0,0003]	4,85	0,451	0,366
10%	0,143	0,33	0,09	[0,884;0,116]	[0,914;0,085;0,001]	[0,883;0,117;0;0]	0,667	1,546	0,597
15%	0,343	0,142	0,204	[0,867;0,133]	[0,893;0,107;0]	[0,882;0,112;0,006;0]	0,615	1,072	0,541

Figure 3: Parameters for *cora* with FW

In this case, the algorithm does an excellent work at discovering noisy layers, since for 5%, 10% and 15% of labelled inputs there is very small change in the weights set β when adding noisy layers, proving that for a sufficient number of inputs the denoising task can be solved in a very precise way. For what concerns the other parameters, they tend to have small values: it means that the model does not require too high regularization and that from a functional point of view its behaviour is between the geometric mean ($\alpha = 0$) and the arithmetic mean ($\alpha = 1$) in most of the settings.

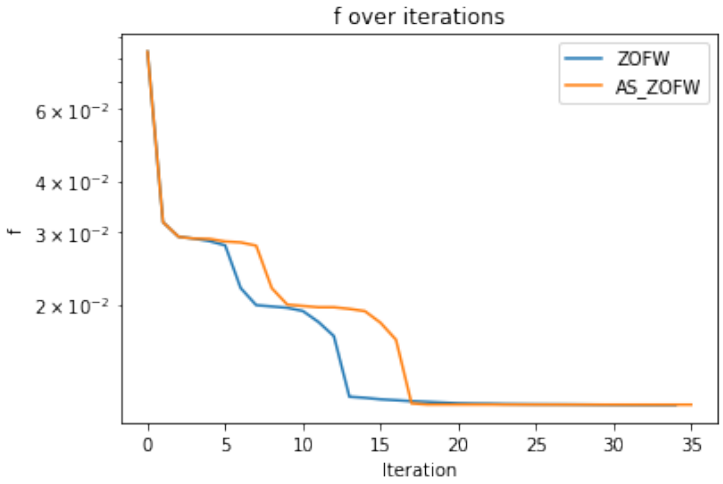
	α			β			λ		
	0 n.l.	1 n.l.	2 n.l.	0 n.l.	1 n.l.	2 n.l.	0 n.l.	1 n.l.	2 n.l.
1%	20	20	20	[1;0]	[1;0;0]	[1;0;0;0]	0,1	0,1	10
5%	0,062	1,131	-17,5	[0,954;0,045]	[1;0;0]	[1;0;0;0]	0,668	0,518	0,719
10%	11,63	0,0088	7,038	[1;0]	[0,985;0,014;0,001]	[1;0;0;0]	0,109	3,169	0,265
15%	0,476	0,01	0,259	[0,957;0,043]	[0,981;0,018;0,001]	[0,906;0,049;0,001;0,044]	0,347	3,129	0,77

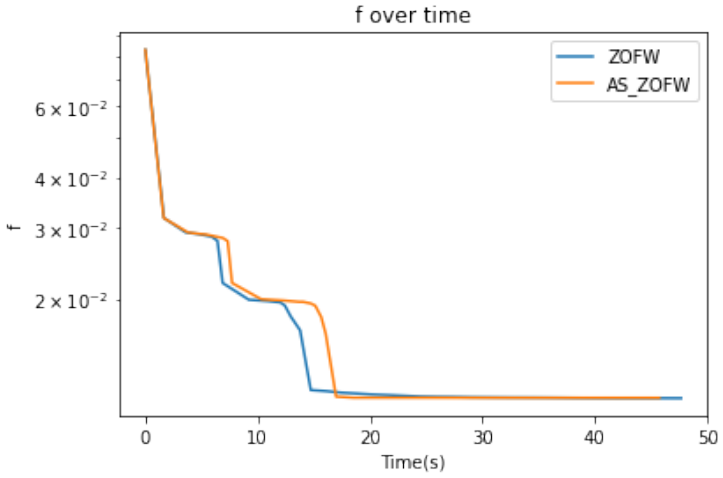
Figure 4: Parameters for *cora* with ASFW

In terms of layers' weights, the algorithm is very efficient in finding noisy layers also in this case, which further proves that the model works very well in finding noisy layers when given enough starting information, i.e. enough labelled inputs.

6.4 Loss function

In this section we will show the plots of the loss function against iterations and then against time.





The loss decreases irregularly, alternating between periods of little to no improvement and large steps in very few iterations.

7 Conclusions

In this paper we have used the works [Ven+23] and [LJ15] to build a model for the Multi-Layer Clustering problem based on optimizing a suitably chosen loss

function through the use of the Frank-Wolfe method and of its variants, able to deal with a small percentage of labelled inputs and with some noise layers, i.e. layers with no useful informations on the clusters. We provided a theoretical explanation on how the model was supposed to use the different tools we have introduced, with a clear description of the structure of the optimization methods and the tools used for building the aggregation function.
We have built a model based on a generalized mean aggregation function approach, building a proper convex combination of the various layers over which apply a suitable non-linear mean, and train a model based on this aggregation function with a certain amount of regularization. After the choice of the cross-entropy loss function we wrote an appropriate minimization problem and found the parameters over which minimizing it.
We have then trained this model on two different datasets, varying for number of nodes and number of layers and communities: we found out that in terms of accuracy the algorithm is able to get good results also when the dataset increases in size, and under some hypotheses it deals very well also with noisy layers. We have then reviewed what parameters the algorithm has returned and what possible meaning these could have, besides an analysis of the loss and function error plots.
What we have found is that the model works quite robustly in terms of ignoring noisy layers, although in terms of accuracy we need at least 5% labelled inputs in order to obtain satisfactory results.

References

[LJ15] Simon Lacoste-Julien and Martin Jaggi. "On the global linear convergence of Frank-Wolfe optimization variants". In: *Advances in neural information processing systems* 28 (2015).

[Ven+23] Sara Venturini et al. "Learning the Right Layers: a Data-Driven Layer-Aggregation Strategy for Semi-Supervised Learning on Multilayer Graphs". In: *arXiv preprint arXiv:2306.00152* (2023).