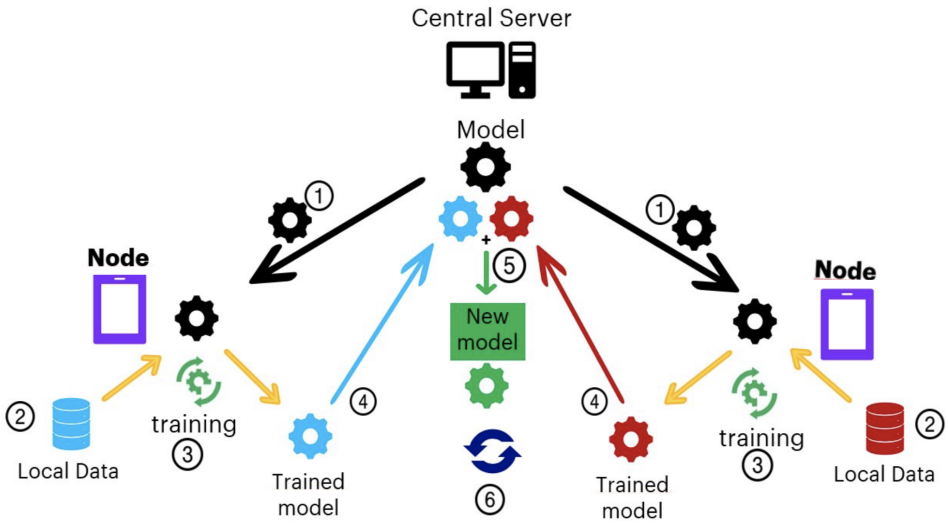
Federated Multi-Task Learning on Acoustic Signals for Predictive Maintenance



Master's Thesis, MATTEK - f21matteksPEC_4

Kristian Juul Tilsted

Mathematical Engineering

Aalborg University

AALBORG UNIVERSITY
STUDENT REPORT

Mathematical Sciences - Aalborg University
Skjernvej 4A, 9220 Aalborg Øst
http://www.aau.dk

Title:

Federated Multi-Task Learning on Acoustic Signals for Predictive Maintenance

Theme:

Master's Thesis

Project period:

Autumn semester 2020 and spring semester 2021

Project group:

MATTEK 9-10 - f21mattekspec_4

Participants:

Kristian Juul Tilsted

Supervisors:

Zheng-Hua Tan
Petar Popovski
Jesper Møller

Number of pages: 95
Date of Completion: 4th of june 2021

Abstract:

This Master's thesis investigates the possibility of developing a federated multi-task learning algorithm with focus on minimal computational complexity and low communications requirements, for use in a distributed predictive maintenance setting. This Master's thesis is a collaboration with Grundfos A/S on predictive maintenance on acoustic data, from their pumping systems. The proposed solution - developed in this work - is compared to the current state-of-the-art federated multi-task algorithm, the MOCHA algorithm. We compare the two solution using ROC curves, which yielded an average decreased performance, in terms of AUC, for 100 runs of each algorithms, across 9 tasks, of 0.095. Though the proposed solution performs worse, the computational complexity is about 150 times less, in terms of FLOPs, compared to that of the MOCHA algorithm.

Preface

This Master Thesis (60 ECTS) is written by Kristian Juul Tilsted of the Master's program: Mathematical Engineering at Aalborg University, Department of Mathematical Sciences and has been completed in the period of September 2020 to June 2021. The project is a collaboration between the Department of Mathematical Sciences at Aalborg University, the Department of Electronic Systems at Aalborg University and Grundfos A/S.

Citations appear as numbers encased by brackets - e.g. [6] - with an optional location specifying, among others, the chapter, section, theorem or definition in the source. The bibliography is ordered numerically after appearance in the thesis.

The author is the creator of all figures presented in this thesis, except for the figure on the front page, which is from [1].

Referencing of theorems, definitions, figures, tables etc. have individual counters, e.g. Theorem 1.1 could follow Definition 1.5.

The author would like to thank the supervisors Zheng-Hua Tan (Department of Electronic Systems), Petar Popovski (Department of Electronic Systems), Jesper Møller (Department of Mathematical Sciences) and Rasmus Engholm (Grundfos A/S) for their guidance during the project.

Aalborg University, June 4, 2021

Resumé på dansk

Søgen efter øget effektivitet og optimering af ressourcebrug og mandetimer, samtidigt med at processeseringsenheder bliver mindre og mere kraftfulde, har givet mulighed for decentraliseret overvågning af og fejlfinding på diverse maskineri. Konstant overvågning af maskineriet, kan give et mere nutidigt og detaljeret indblik i maskineriets tilstand. Derved, gøres det muligt at tilpasse vedligeholdelsen på individuelle maskiner, uafhængigt at arbejdspres og miljø, som den individuelle maskine befinder sig i. Vi kalder dette for prædiktiv vedligeholdelse, altså at forudsige hvornår en maskine behøver vedligeholdelse.

I dette kandidat speciale arbejder vi særskilt på en case fra Grundfos A/S, som omhandler prædiktiv vedligeholdelse på nogle af deres pumpesystemer. Vi ønsker at undersøge hvorledes det er muligt at konstruere en model ved hjælp af machine learning, som kan tilpasse sig de individuelle pumpesystemer og deres miljø således vi kan give en detaljeret beskrivelse af pumpesystemmets tilstand. Vi forestiller os at pumpesystemerne allerede er blevet installeret hos kunderne. Et hvert system er udstyret med en processeserings- og en kommonikationsenhed, som muliggøre delvise computer operationer på selve systemerne, og tillader kommunikation med en central server, placeret hos Grundfos.

Da, vi ingen antagelser gør os om de miljøer, hvor i pumpesystemerne placeres, antager vi nødvendigheden af en unik model til ethvert pumpesystem. Samtidigt er der en forventning om at pumpesystemer af samme type, vil have mange ligheder i deres tilhørende modeller, og den største forskel vil være baggrundsstøjen fra miljøet hvori systemerne er placeret i. Derved, skal der bygges et system som tillader samarbejde blandt pumpesystemerne. Vi har en forventning at dette vil øge indlæringshastigheden og kvaliteten af de modeller som i sidste ende bliver konstrueret.

Det findes allerede algoritmer som kan løse disse typer af problemer. Nogle af dem er [2, 3], som dog begge kræver kommunikation af selve det data, som opsamles ved de individuelle pumpesystemer. Dette kræver øget kommunikation mellem pumpesystemer og den centrale server hos Grundfos, som vi, i dette speciale, ikke kan tillade. Et andet forslag til en løsning er [4], som på nuværende tidspunkt er en state-of-the-art løsning. Vi sammenligne de resultater, som opnåes i dette kandidat speciale med [4].

I dette speciale udvikler vi en federated multi-task leanring løsning som opfylder de oven-

stående krav, med særligt stort fokus på at løsningen har mindre krav til processeserings- og kommunikationsenheden i pumpesystemerne. Dog, som forventet finder vi, at med mindre processeseringskraft følger ringere evne til, at detaljeret give indblik i pumpesystemerne tilstand.

Det viser sig, at den udviklede løsning er i stand til, i nogle tilfælde ligeværdig og andre ringere end [4], at forudsige pumpesystemernes tilstand. Dette er dog forventeligt, da antallet af brugte computeroperationer (FLOPs) er ca. 150 gange færre ved den udviklede løsning end hos [4]. Vi må derfor konkludere, at den udviklede løsning ikke bliver en konkurrent til allerede eksisterende metoder.

Contents

1 Introduction

The everlasting industrial search for increased efficiency and reduction in cost of both power and man hours, has laid the groundwork for increased monitoring of production machinery and product tests. A more detailed evaluation of the status of the machinery gives the opportunity of planning repairs or complete swaps. This will in turn increases efficiency by avoiding repairs or swaps in the middle of a batch of products and diminishes the possibility of unforeseen emergencies. Planning repairs and avoiding these unforeseen emergencies could potentially reduce the maintenance cost of running a production, which in many cases range from 15% to 60 % of the entire production cost [5]. Besides the increase in efficiency, continued monitoring of products after them being sold to the customers has the potential of becoming a secondary income source or an advantage that will favour a company's products over its competitors'.

This project is done in collaboration with Grundfos A/S on machine learning and acoustic sensing for predictive maintenance. Grundfos is a company that manufactures a wide range of water solutions for both private and commercial use [6]. In this collaboration with Grundfos, we pretend that the leadership at Grundfos see a potential business opportunity in offering a monitoring solution to customers, to inform them of the status of their pumps. We shall in this project investigate if and how such a solution could become real.

Due to the both private and commercial sales, there is no minimum limit on how many units a particular customer buys. Thus, there is no control over amount of pumps being installed at a given location. To avoid the high costs and inconveniences associated with having a technician visit every single location of a sold pump, to perform maintenance service, the desired solution has to be automatic. This naturally leads to a machine learning approach which, once the model is trained, is a fully automated process of predicting the status of the pump. A solution where a computation unit is placed on the pumps with memory, computational power and transmission ability is proposed. As every single pump would have to be equipped with such a device, limiting the cost of the device is essential. A vital part of this project should concern the required processing power. We, therefore, make an assumption of having limited processing power on each of the pumps. This means that a fully local machine learning scheme might not be feasible, as would be the traditional approach.

Even within the category of pumps there exists a multitude of sizes and shapes based on the needs of the customer and the application. Thus, is it unlikely to assume that we can build one

model that would fit all types of pumps. Another problem is the environment that the pumps are placed in. These can be vastly different. That is, some pump might operate in noisy environments, whilst others in near silent acoustic environments. This means, that one model for each type of pump might not even be specific enough to determine the status of a pump.

Lately, there has been a lot of research in the field of federated (or decentralised) learning, the act of learning a model in a setting where the data is located in multiple location with no realistic means of moving the data. However, in general, federated learning is employed to build a single model from the averaging of multiple models learned where the data is stored. [7, 8]

The idea for a solution is to learn multiple models at the same time, given limited processing power, memory and transmission power. In a multi-task machine learning scheme, multiple tasks (models) are learned at the same time. In such a scheme, the goal is to take advantage of potential similarities between the tasks, e.g. two pumps could be of the same type. However, multi-task learning - in it self - does not imply friendliness or fitness to be employed in a federated settings, as many of them utilise a kernel matrix consisting of some transformation of the inner products between all the data points regardless of task association [9, 10].

Recently, a new field has emerged. The field of federated multi-task learning, which is the act of learning multiple related models (or tasks) simultaneously in a federated setting. A federated multi-task solution could be a perfect solution. There already exists algorithms which could solve the problem at hand. However, common for both [3] and [11] is the need - in some capacity - to share or move the data, as both of them rely on a kernel matrix. This kernel matrix is a strong tool as it expresses relations between the data points, even across tasks. However, the goal of this project is to develop a solution which rely on only very limited communications. There exists other solutions, one of them is [2], however, it focuses on deep neural networks (abbr. DNN). A DNN requires a large amount of computations, but has the opportunity to generate an outstanding prediction model. Thought they have great upsides, the requirement on the amount of computations makes it infeasible for this problem. Lastly, possible the best solution that currently exists is the MOCHA algorithm from [4]. The MOCHA algorithm does not - like the others mentioned - require any sharing of the data.

Common for all federated multi-task learning schemes - mentioned here - as well as federated learning schemes, is the need for a central server for the pumps to communicate with [3, 4, 11]. With the limited processing power on the pumps, the aim should be for most of the computations to be handled by the central server. With the above introduction in mind we investigate the potential for a solution to the described problem.

1.1 Problem Statement

Based upon the above introduction we formulate the problem statement:

In what capacity is it possible to build an algorithm that conducts meaningful predictive maintenance on machinery by the use of federated multi-task learning focusing on low computational complexity and without any sharing of data points?

Sub-Questions

- How does machine learning function in a federated setting?
- How does one learn multiple machine learning tasks simultaneously, by letting the tasks cooperate to improve learning?
- In what way can these two concepts be combined?
- Is possible to construct an algorithm that detects faulty machinery using federated multi-task learning?
- How does such an algorithm stack up against traditional methods?

Delimitations

The goal of the project is to find a solution to the federated predictive maintenance problem described above. While a sub-goal is to minimise the number of computations at the pumps, we set no upper bound on this, we simply aim to minimise the number of computations. We make the assumption that the central server has a large amount of processing power. We solely comment on this as there is no consideration in regards to the memory nor processing power requirements on the central server. We shall not consider routing protocols, schedulers nor channel models, as we simply assume that these function with lossless channels. We shall, however, consider the amount of floating point values transmitted to compare this to moving the entirety of the data sets. Likewise, we are going to compare the number of floating point operations required, but will do so in a non-detailed oriented way, as computer science is outside the scope of the project.

Outline

The thesis is structured as follow: An introduction to preliminary optimisation results, aiming to ease the derivations later on in the thesis, is found in chapter 2. In chapter 3, we introduce the necessary machine learning schemes and techniques, for us to be able to develop a new federated multi-task learning algorithm. Chapter 4 introduced the proposed solution of this thesis in great theoretical detail, and lastly presents the associated algorithm. Next, in chapter 5, we experiment with the proposed solution, to determine its capabilities, and present the results of said experiments. Lastly, in chapters 6 and 7, we question the methods and results obtained in the thesis, and give our final verdict of the usefulness of the proposed solution, respectively.

2 Optimisation Theory

We shall in this chapter discuss introductory optimisation theory. The goal of this chapter is to introduce concepts and theorems, which shall lay the groundwork for later discussions. Unless otherwise stated, this chapter is based on [12]

2.1 General Properties of Optimisation Problems

We shall in this section briefly cover some of the important concepts of optimisation theory. We begin by examining a general formulation of an minimisation problem.

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f_0(\boldsymbol{x}) \\ \text{s.t.} \quad & f_i(\boldsymbol{x}) \leq 0 \quad \text{for } i = 1,\dots N \\ & h_i(\boldsymbol{x}) = 0 \quad \text{for } i = 1,\dots M, \end{aligned} \tag{2.1}$$

where $f_0 : \mathbb{R}^n \to \mathbb{R}$ is called the objective function, $f_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1,\dots N$ are called the inequality constraints and $h_i : \mathbb{R}^n \to \mathbb{R}$ for $i = 1,\dots M$ are called the equality constraints. The domain $\mathcal{D}$ of (2.1) is defined as follows:

$$\mathcal{D} = \bigcap_{i=0}^{N} \operatorname{dom} f_i \cap \bigcap_{i=1}^{M} \operatorname{dom} h_i,$$

where $\operatorname{dom} f_i$ is the domain of f_i. The subset $\mathcal{F} \subset \mathcal{D}$ given as

$$\mathcal{F} = \left\{ \boldsymbol{x} \in \mathcal{D} \mid f_i(\boldsymbol{x}) \leq 0, i \in [0, N] \wedge h_i(\boldsymbol{x}) = 0, i \in [1, M] \right\},$$

is known as the *feasible set*, the set of all points $\boldsymbol{x}$ between which we can find the minimum. Thus, $\boldsymbol{x} \in \mathcal{F}$ is a *feasible point* and a point $\tilde{\boldsymbol{x}} \in \mathcal{F}$ is said to be *strictly feasible* if $f_i(\tilde{\boldsymbol{x}}) < 0$ for $i = 1,\dots N$. We then say that the *optimal value* $p^\star$ for the minimisation problem in (2.1) is defined as

$$p^\star = \inf \left\{ f_0(\boldsymbol{x}) | \boldsymbol{x} \in \mathcal{F} \right\},$$

likewise, $\boldsymbol{x}^\star$ is the point such $f_0(\boldsymbol{x}^\star) = p^\star$, called the *optimal point*.

An interesting subclass of optimisation problems is convex optimisation problems, which we defined next.

Definition 2.1. Convex Function

A function f is convex if dom f is convex and if for all $\boldsymbol{x}, \boldsymbol{y} \in$dom f and $0 \leq \theta \leq 1$, we have

$$f(\theta \boldsymbol{x} + (1-\theta)\boldsymbol{y}) \leq \theta f(\boldsymbol{x}) + (1-\theta) f(\boldsymbol{y}). \tag{2.2}$$

f is said to be strictly convex if the inequality in (2.2) is sharp.

Definition 2.2. Convex Optimisation Problem

An optimisation problem

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f_0(\boldsymbol{x}) \\ \text{s.t.} \quad & f_i(\boldsymbol{x}) \leq 0 \quad \text{for } i = 1, \dots N \\ & h_i(\boldsymbol{x}) = 0 \quad \text{for } i = 1, \dots M, \end{aligned}$$

is convex if the objective function and the constraints are convex.

The optimisation problem of the form in (2.1) is known as the *primal problem*, however, there is also a formulation known as the *dual problem* which solution gives a lower bound for $p^\star$ of (2.1). To formulate the dual problem, one needs the *Lagrangian* $\mathrm{L} : \mathbb{R}^n \times \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}$, which is defines as

$$\mathrm{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f_0(\boldsymbol{x}) + \sum_{i=1}^{N} \lambda_i f_i(\boldsymbol{x}) + \sum_{i=1}^{M} \mu_i h_i(\boldsymbol{x}),$$

where $\boldsymbol{\lambda}$ and $\boldsymbol{\mu}$ are known as the *Lagrangian multipliers*. Next we introduce the dual function $g : \mathbb{R}^N \times \mathbb{R}^M \to \mathbb{R}$:

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{x} \in \mathcal{D}} \mathrm{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}).$$

Note that the dual function is concave, due to it being the point-wise infimum of a family of affine functions of $(\boldsymbol{\lambda}, \boldsymbol{\mu})$, even if the problem in (2.1) it self is not convex [12]. We have previously claimed that the dual function is a lower bound for $p^\star$, which we prove in the next theorem.

Theorem 2.1. Dual Function is a Lower Bound

The optimal value $p^\star$ for an optimisation problem is lower bounded by the dual function of said optimisation problem, that is

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \leq p^\star,$$

for any $\boldsymbol{\lambda} \succeq 0$ and $\boldsymbol{\mu}$.

Proof

Assume $\bar{\boldsymbol{x}} \in \mathcal{F}$ and $\boldsymbol{\lambda} \succeq 0$, thus we have

$$\sum_{i=1}^{N} \lambda_i f_i(\bar{\boldsymbol{x}}) + \sum_{i=1}^{M} \mu_i h_i(\bar{\boldsymbol{x}}) \leq 0, \tag{2.3}$$

since the first sum is less than or equal to zero and the second sum is exactly zero. Equation (2.3) implies that

$$\mathrm{L}(\bar{\boldsymbol{x}}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f_0(\bar{\boldsymbol{x}}) + \sum_{i=1}^{N} \lambda_i f_i(\bar{\boldsymbol{x}}) + \sum_{i=1}^{M} \mu_i h_i(\bar{\boldsymbol{x}}) \leq f_0(\bar{\boldsymbol{x}}),$$

and hence

$$g(\boldsymbol{\lambda}, \boldsymbol{\mu}) = \inf_{\boldsymbol{x} \in \mathcal{D}} \mathrm{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq \mathrm{L}(\bar{\boldsymbol{x}}, \boldsymbol{\lambda}, \boldsymbol{\mu}) \leq f_0(\bar{\boldsymbol{x}}),$$

yielding the desired inequality and completing the proof. ■

Note that there are technically no constraints on the Lagrangian multipliers, but the dual function only gives a nontrivial lower bound on $p^\star$ when $\boldsymbol{\lambda} \succeq 0$. Thus, we call a pair $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ *dual feasible*, when $\boldsymbol{\lambda} \succeq 0$, $(\boldsymbol{\lambda}, \boldsymbol{\mu}) \in \operatorname{dom} g$ and $g(\boldsymbol{\lambda}, \boldsymbol{\mu}) > -\infty$.

Knowing that the dual problem gives a lower bound on the optimal value of an optimisation problem, the question of what the best lower bound is naturally arises, that is, the maximum values of the dual function. This leads us to the formulation of the dual problem:

$$\begin{aligned} \max_{\boldsymbol{x}} \quad & g(\boldsymbol{\lambda}, \boldsymbol{\mu}) \\ \text{s.t.} \quad & \boldsymbol{\lambda} \succeq 0. \end{aligned} \tag{2.4}$$

Note that we refer to the pair $(\boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star)$ as the *optimal Lagrangian multipliers* and $d^\star$ as the optimal value for the dual problem. Lastly note that (2.4) is a convex problem as it maximises a concave function.

It is not always the case that $d^\star = p^\star$, in which case *weak duality* holds and we refer to the scalar $p^\star - d^\star$ as the *duality gap*. If, on the other hand is equality, i.e. the duality gap is zero, *strong duality* holds. Strong duality is obviously a nice property, but not one that holds in general. Next we introduce *Slater's theorem*, a result presenting a sufficient condition for strong duality. First we define *Slater's condition*:

Definition 2.3. Slater's Condition

A convex optimisation problem on the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f_0(\boldsymbol{x}) \\ \text{s.t.} \quad & f_i(\boldsymbol{x}) \leq 0 \quad \text{for } i = 1, \dots N \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned}$$

for which there exists a strictly feasible point $\boldsymbol{x} \in \operatorname{relint} \mathcal{D}$, is said to satisfy Slater's condition.

Theorem 2.2. Slater's Theorem

For a convex optimisation problem on the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f_0(\boldsymbol{x}) \\ \text{s.t.} \quad & f_i(\boldsymbol{x}) \leq 0 \quad \text{for } i = 1, \dots N \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned}$$

satisfying Slater's condition, strong duality holds.

Proof

To simplify the proof we assume that the interior of $\mathcal{D}$ is nonempty, thus relint $\mathcal{D} = \text{int}\,\mathcal{D}$. We also assume that $p^\star$ is finite and that $\boldsymbol{A}$ has full row rank. We construct two sets:

$$\mathcal{A} = \left\{(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathbb{R}^N \times \mathbb{R}^M \times \mathbb{R} \,\middle|\, \exists \boldsymbol{x} \in \mathcal{D}, f_i(\boldsymbol{x}) \leq u_i,\ i = 1, \dots, N,\ \boldsymbol{A}\boldsymbol{x} - \boldsymbol{b} = \boldsymbol{v},\ f_0(\boldsymbol{x}) \leq t \right\}$$
$$\mathcal{B} = \left\{(\boldsymbol{0}, \boldsymbol{0}, t) \in \mathbb{R}^N \times \mathbb{R}^M \times \mathbb{R} | t < p^\star \right\}.$$

Obviously $\mathcal{A} \bigcap \mathcal{B} = \emptyset$, and both $\mathcal{A}$ and $\mathcal{B}$ are covex. Thus, the separating hyperplane theorem [12, §2.51] ensures the existence of $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \xi) \neq \boldsymbol{0}$ and α, such that

$$(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathcal{A} \implies \boldsymbol{\lambda}^\mathsf{T}\boldsymbol{u} + \boldsymbol{\mu}^\mathsf{T}\boldsymbol{v} + \xi t \geq \alpha \tag{2.5}$$
$$(\boldsymbol{u}, \boldsymbol{v}, t) \in \mathcal{B} \implies \boldsymbol{\lambda}^\mathsf{T}\boldsymbol{u} + \boldsymbol{\mu}^\mathsf{T}\boldsymbol{v} + \xi t \leq \alpha. \tag{2.6}$$

Equation (2.5) implies that $\boldsymbol{\lambda} \geq \boldsymbol{0}$ and $\xi \geq 0$, otherwise, $\boldsymbol{\lambda}^\mathsf{T}\boldsymbol{u} + \xi t$ would be unbounded from below, contradicting (2.5). Equation (2.6) implies that $\xi t \leq \alpha$ since $\boldsymbol{\lambda}^\mathsf{T}\boldsymbol{u} + \boldsymbol{\mu}^\mathsf{T}\boldsymbol{v} = 0$. Since $\xi t \leq \alpha\ \forall t < p^\star$ we have $\xi p^\star \leq \alpha$ [12]. We shall next show that strong duality holds when $\xi > 0$, and then show that $\xi = 0$ cannot occur. Firstly, expanding (2.5), gives

$$\sum_{i=1}^{N} \lambda_i f_i(\boldsymbol{x}) + \boldsymbol{\mu}^\mathsf{T}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + \xi f_0(\boldsymbol{x}) \geq \xi p^\star. \tag{2.7}$$

Assuming $\xi > 0$:

Dividing (2.7) by ξ, which is possible by the assumption of $\xi > 0$, gives

$$\sum_{i=1}^{N} \frac{\lambda_i}{\xi} f_i(\boldsymbol{x}) + \frac{1}{\xi}\boldsymbol{\mu}^\mathsf{T}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + f_0(\boldsymbol{x}) \geq p^\star,$$

from which it is clearly seen that the dual function is lower bounded by the optimal value $p^\star$, that is

$$g(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}) = \inf_{\boldsymbol{x}} \mathrm{L}(\boldsymbol{x}, \bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}) = \sum_{i=1}^{N} \frac{\lambda_i}{\xi} f_i(\boldsymbol{x}) + \frac{1}{\xi}\boldsymbol{\mu}^\mathsf{T}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) + f_0(\boldsymbol{x}) \geq p^\star,$$

defining $\bar{\boldsymbol{\lambda}} = \boldsymbol{\lambda}/\xi$ and $\bar{\boldsymbol{\mu}}/\xi$. But from theorem 2.1 we know that $g(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}) \leq p^\star$, thus we must have

$$g(\bar{\boldsymbol{\lambda}}, \bar{\boldsymbol{\mu}}) = p^\star,$$

and strong duality holds.

Assuming $\xi = 0$:
Consider (2.7) for $\xi = 0$, thus yielding

$$\sum_{i=1}^{N} \lambda_i f_i(\boldsymbol{x}) + \boldsymbol{\mu}^\mathsf{T} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \geq 0.$$

Since the primal problem upholds Slater's condition, there exists a point $\boldsymbol{x}' \in \text{int}\mathcal{D}$, such that $\boldsymbol{A}\boldsymbol{x}' - \boldsymbol{b} = \boldsymbol{0}$, thus

$$\sum_{i=1}^{N} \lambda_i f_i(\boldsymbol{x}') \geq 0,$$

however, particular: $f_i(\boldsymbol{x}') < 0$ for $i = 1, \ldots, N$, thus $\boldsymbol{\lambda}$ must equate to zero. Recall that the separating hyperplane theorem ensure the existence of $(\boldsymbol{\lambda}, \boldsymbol{\mu}, \xi) \neq \boldsymbol{0}$, and so far both $\xi = 0$ and $\boldsymbol{\lambda} = \boldsymbol{0}$, which implies that $\boldsymbol{\mu} \neq \boldsymbol{0}$. This means that

$$\boldsymbol{\mu}^\mathsf{T} (\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}) \geq 0.$$

However, since $\boldsymbol{x}' \in \text{int}\mathcal{D}$ there exists $\boldsymbol{\epsilon} > 0$ such $\boldsymbol{x}' + \boldsymbol{\epsilon} \in \mathcal{D}$ and

$$\boldsymbol{\mu}^\mathsf{T} \left(\boldsymbol{A}\left(\boldsymbol{x}' + \boldsymbol{\epsilon}\right) - \boldsymbol{b} \right) < 0,$$

unless $\boldsymbol{A}^\mathsf{T}\boldsymbol{\mu} = 0$, however, this contradicts the assumption of $\boldsymbol{A}$ having full rank. Thus we have proven that $\xi = 0$ cannot be the case and the proof is complete.

■

Next we assume that $f_0(\ldots), \ldots, f_N(\ldots)$ are differentiable, and let $\boldsymbol{x}^\star$ and $\boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star$ be the primal and dual optimal points. Since $\boldsymbol{x}^\star$ minimises the Lagrangian, the gradient must vanish at $\boldsymbol{x}^\star$, that is

$$\begin{aligned} 0 &= \nabla \mathrm{L}(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star) \\ &= \nabla f_0(\boldsymbol{x}^\star) + \sum_{i=1}^{N} \lambda_i^\star \nabla f_i(\boldsymbol{x}^\star) + \sum_{i=1}^{M} \lambda_i^\star \nabla h_i(\boldsymbol{x}^\star). \end{aligned}$$

Hereby, we have that, for $\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star$ to be optimal points, the following must old

$$\begin{aligned} f_i(\boldsymbol{x}^\star) &\leq 0 \\ h_i(\boldsymbol{x}^\star) &= 0 \\ \lambda_i^\star &\geq 0 \\ \lambda_i^\star f_i(\boldsymbol{x}^\star) &= 0 \\ \nabla f_0(\boldsymbol{x}^\star) + \sum_{i=1}^{N} \lambda_i^\star \nabla f_i(\boldsymbol{x}^\star) + \sum_{i=1}^{M} \lambda_i^\star \nabla h_i(\boldsymbol{x}^\star) &= 0. \end{aligned}$$

These conditions are known as the *Karush-Kuhn-Tucker* (KKT) conditions.

Next we present a result that claims strong duality for convex optimisation problems.

Theorem 2.3.

For a convex optimisation problem on the form

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f_0(\boldsymbol{x}) \\ \text{s.t.} \quad & f_i(\boldsymbol{x}) \leq 0 \quad \text{for } i = 1, \dots N \\ & h_i(\boldsymbol{x}) = 0 \quad \text{for } i = 1, \dots M, \end{aligned}$$

where $f_i(\cdot)$ for $i = 0, \dots, N$ and $h_i(\cdot)$ for $i = 1, \dots, M$ are differentiable, strong duality holds.

Proof

let $\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star$ uphold the KKT conditions, thus

$$\begin{aligned} g(\boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star) &= \mathrm{L}(\boldsymbol{x}^\star, \boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star) \\ &= f_0(\boldsymbol{x}^\star) + \sum_{i=1}^{N} \lambda_i^\star f_i(\boldsymbol{x}^\star) + \sum_{i=1}^{M} \lambda_i^\star h_i(\boldsymbol{x}^\star) \\ &= f_0(\boldsymbol{x}^\star), \end{aligned}$$

since $\lambda_i^\star f_i(\boldsymbol{x}^\star) = 0$ for $i = 1, \dots, N$ and $h_i(\boldsymbol{x}^\star) = 0$ for $i = 1, \dots, M$, and the proof is complete.

■

This means that, if a convex optimisation problem has differentiable objective function and constraints, and satisfies Slater's condition, the KKT conditions provide necessary and sufficient conditions for optimality.

Lastly we present a result on uniqueness of solutions.

Proposition 2.1.

If f is a convex function with convex domain $\mathcal{D}$, then any local minimiser of f is also a global minimiser.

Proof

Assume f is convex with convex domain $\mathcal{D}$ and let $N \subseteq \mathcal{D}$, be a neighbourhood of the domain of f. Then let $\boldsymbol{x}^\star \in N$, such that for all $\boldsymbol{x} \in N$ we have $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x})$. Next suppose towards a contradiction that there exists $\tilde{\boldsymbol{x}} \in \mathcal{D}$, such that $f(\tilde{\boldsymbol{x}}) < f(\boldsymbol{x}^\star)$. Now let $\boldsymbol{x}(t) = t\boldsymbol{x}^\star + (1-t)\tilde{\boldsymbol{x}}$ for $t \in [0,1]$. Note that by convexity of $\mathcal{D}$, $\boldsymbol{x}(t) \in \mathcal{D}$. Then by convexity of f, we have

$$f(\boldsymbol{x}(t)) \leq t f(\boldsymbol{x}^\star) + (1-t) f(\tilde{\boldsymbol{x}}) < t f(\boldsymbol{x}^\star) + (1-t) f(\boldsymbol{x}^\star) = f(\boldsymbol{x}^\star). \tag{2.8}$$

However, we can choose t sufficiently close to 1, such that $\boldsymbol{x}(t) \in N$, which implies that $f(\boldsymbol{x}^\star) \leq f(\boldsymbol{x}(t))$. However, this is in contradiction with (2.8) and the proof is complete. ■

Proposition 2.2. Uniqueness of minima in Strictly Convex Functions

If f is a strictly convex function and has convex domain $\mathcal{D}$, then there exists at most one local minimum. If it exists, then it is a global minimum.

Proof

The second statement follows directly from the first by proposition 2.1. Thus, we only have to show that if a local minimiser exists, it is unique. Suppose that $\boldsymbol{x}^\star \in \mathcal{D}$ is a local minimiser of f. Then suppose towards a contradiction that $\tilde{\boldsymbol{x}} \in \mathcal{D}$ too is a local minimiser of f, such $\boldsymbol{x}^\star \neq \tilde{\boldsymbol{x}}$. As f is strictly convex, it is convex and thus $f(\boldsymbol{x}^\star) = f(\tilde{\boldsymbol{x}})$. Consider $\boldsymbol{x}(t) = t\boldsymbol{x}^\star + (1-t)\tilde{\boldsymbol{x}}$ for $t \in [0,1]$, and due to the convexity of $\mathcal{D}$, we have $\boldsymbol{x}(t) \in \mathcal{D}$ for all t. Since f is strictly convex, we have

$$f(\boldsymbol{x}(t)) < tf(\boldsymbol{x}^\star) + (1-t)f(\tilde{\boldsymbol{x}}) = tf(\boldsymbol{x}^\star) + (1-t)f(\boldsymbol{x}^\star) = f(\boldsymbol{x}^\star).$$

However this shows that $f(\boldsymbol{x}(t)) < f(\boldsymbol{x}^\star)$ contradicting that $\boldsymbol{x}^\star$ is a global minimiser. Thus, if $\tilde{\boldsymbol{x}}$ is a local minimiser, then $\tilde{\boldsymbol{x}} = \boldsymbol{x}^\star$, and $\boldsymbol{x}^\star$ is unique. ■

2.2 Solving Inequality Constrained Optimisation Problems

Consider the following optimisation problem

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f_0(\boldsymbol{x}) \\ \text{s.t.} \quad & f_i(\boldsymbol{x}) \leq 0 \quad \text{for } i = 1, \dots N \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned} \tag{2.9}$$

where $f_0, \dots f_N : \mathbb{R}^M \to \mathbb{R}$ are twice continuous differentiable and convex, and $\boldsymbol{A} \in \mathbb{R}^{n \times m}$ with $\text{rank}\boldsymbol{A} = n < m$. We assume the existence of a solution to (2.9), which we denote $f_0(\boldsymbol{x}^\star) = p^\star$. Additionally, Slater's theorem is assumed to be fulfilled, i.e. strong duality holds.

Having inequalities constraints in (2.9) makes it difficult to solve, however, we can eliminate the constraints by instead including them into the objective function as such

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & f_0(\boldsymbol{x}) + \textstyle\sum_{i=1}^N I(f_i(\boldsymbol{x})) \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned} \tag{2.10}$$

where I is the indicator function given as

$$I(v) = \begin{cases} 0 & \text{if } v \leq 0 \\ \infty & \text{if } v \leq 0\,. \end{cases}$$

Equation (2.10) has no inequalities, however, it is no longer differentiable. One solution to this problem is to approximate the indicator function with a logarithmic function as such

$$I(v) \approx -\frac{1}{t}\ln(-v),$$

where $t > 0$ controls the accuracy. That is

$$\lim_{t\to\infty} -\frac{1}{t}\ln(-v) = I(v). \tag{2.11}$$

Thus we state the convex optimisation problem, which we aim to solve

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & t f_0(\boldsymbol{x}) + \phi(\boldsymbol{x}) \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}, \end{aligned} \tag{2.12}$$

which has the same optimal value as (2.9), and $\phi(\boldsymbol{x}) = -\sum_{i=1}^{N}\ln(-f_i(\boldsymbol{x}))$. Equation (2.11) gives the intuitive idea that increasing t yields a solution closer to the solution of (2.9).

The method for solving (2.12) is the *barrier method*, which functions by solving a series of equality constrained optimisation problems using Newton's method, using the previous optimum as starting point for the next problem. With each iteration the t-value is increased by a factor τ such $-1/t\ln(\cdot)$ better approximates the indicator function. This also circumvents the problem of (2.12) begin difficult to solve for large t by using Newton's method, as t will only be large for final few times the problem is solved.

Each time (2.12) is solved for given t-value, a minimiser $\boldsymbol{x}^\star(t)$ is found. Imagine that this process is done multiple times, such that we achieve a set of these minimisers: $\mathcal{C}_p$, which is referred to as the *central path*. Common among the points on the central path is that they are strictly feasible [12], that is for any $\boldsymbol{x}^\star(t) \in \mathcal{C}_p$

$$\boldsymbol{A}\boldsymbol{x}^\star(t) = \boldsymbol{b}, \qquad f_i(\boldsymbol{x}^\star(t)) < 0,\ \forall i$$

and that there exists $\hat{\boldsymbol{\mu}}$ such

$$\begin{aligned} 0 &= t\nabla f_0(\boldsymbol{x}^\star(t)) + \nabla\phi(\boldsymbol{x}^\star(t)) + \boldsymbol{A}^\mathsf{T}\hat{\boldsymbol{\mu}} \\ &= t\nabla f_0(\boldsymbol{x}^\star(t)) + \sum_{i=1}^{N}\frac{1}{-f_i(\boldsymbol{x}^\star(t))}\nabla f_i((\boldsymbol{x}^\star(t)) + \boldsymbol{A}^\mathsf{T}\hat{\boldsymbol{\mu}}. \end{aligned}$$

Next we present a result that further strengthens our intuition that the increase of t will yield more accurate solutions

Theorem 2.4. Duality Gap for Central Points

> Every point on the central path $\boldsymbol{x}^\star(t) \in \mathcal{C}_p$ yields a dual feasible pair $(\boldsymbol{\lambda}^\star, \boldsymbol{\mu}^\star)$, and for a particular $\boldsymbol{x}^\star(t) \in \mathcal{C}_p$ the dual gap is
>
> $$p^\star - d^\star(t) = N/t.$$

Proof

There exists a $\hat{\boldsymbol{\mu}}$ such

$$0 = t\nabla f_0(\boldsymbol{x}^\star(t)) + \sum_{i=1}^{N}\frac{1}{-f_i(\boldsymbol{x}^\star(t))}\nabla f_i((\boldsymbol{x}^\star(t)) + \boldsymbol{A}^\mathsf{T}\hat{\boldsymbol{\mu}}, \tag{2.13}$$

for every point in $\mathcal{C}_p$ and $t > 0$. Next we claim that the pair

$$\lambda_i^\star(t) = -\frac{1}{t f_i(\boldsymbol{x}^\star)} \quad \text{and} \quad \boldsymbol{\mu}^\star(t) = \hat{\boldsymbol{\mu}}/t$$

is dual feasible. This requires that $\lambda_i^\star(t) > 0$, which it is since $f_i(\boldsymbol{x}^\star) < 0$ and that $\boldsymbol{x}^\star$ minimises the Lagrangian. To see this we rewrite (2.13) as

$$0 = \nabla f_0(\boldsymbol{x}^\star(t)) + \sum_{i=1}^{N} \lambda_i^\star(t) \nabla f_i((\boldsymbol{x}^\star(t)) + \boldsymbol{A}^\mathsf{T} \boldsymbol{\mu}^\star(t),$$

which is the derivative of the Lagrangian equated to zero. Thus $\boldsymbol{x}^\star$ is seen to minimise the Lagrangian

$$\mathrm{L}(\boldsymbol{x}, \boldsymbol{\lambda}, \boldsymbol{\mu}) = f_0(\boldsymbol{x}) + \sum_{i=1}^{N} \lambda_i f_i((\boldsymbol{x}) + \boldsymbol{\mu}(\boldsymbol{A}\boldsymbol{x} - \boldsymbol{b}),$$

where $\lambda_i = \lambda_i^\star(t)$ and $\boldsymbol{\mu} = \mu^\star(t)$, which in turn means that $(\boldsymbol{\lambda}, \boldsymbol{\mu})$ is dual feasible. Thus the dual function is finite and

$$\begin{aligned}
g(\boldsymbol{\lambda}^\star(t), \boldsymbol{\mu}^\star(t)) &= \mathrm{L}(\boldsymbol{x}^\star(t), \boldsymbol{\lambda}^\star(t), \boldsymbol{\mu}^\star(t)) \\
&= f_0(\boldsymbol{x}^\star(t)) + \sum_{i=1}^{N} \lambda_i^\star(t) f_i((\boldsymbol{x}^\star(t)) + \boldsymbol{\mu}^\star(t)(\boldsymbol{A}\boldsymbol{x}^\star(t) - \boldsymbol{b}) \\
&= f_0(\boldsymbol{x}^\star(t)) - \sum_{i=1}^{N} \frac{1}{t f_i(\boldsymbol{x}^\star)} f_i((\boldsymbol{x}^\star(t)) \\
&= f_0(\boldsymbol{x}^\star(t)) - N/t
\end{aligned}$$

Thus we can show that the duality gap is:

$$\begin{aligned}
p^\star - d^\star(t) &= f_0(\boldsymbol{x}^\star(t)) - (f_0(\boldsymbol{x}^\star(t)) - N/t) \\
&= N/t,
\end{aligned}$$

and the proof is complete.

■

This result increases our confidence in the intuition; that letting $t \to \infty$ will give a more accurate solution.

Lastly we arrive at the description of the barrier method. Using the barrier method we can obtain arbitrary small accuracy $\epsilon > 0$, simply by continuing the series of solving (2.12) until $t > N/\epsilon$.

Algorithm 1 The Barrier Method Algorithm

Input: strictly feasible $\boldsymbol{x}$, $t := t^{(0)} > 0$, $\tau > 1$, $\epsilon > 0$
Output: Optimal point $\boldsymbol{x}^\star$

1: **while** $t < N/\epsilon$ **do**
2: Compute $\boldsymbol{x}^\star(t)$ by solving: $\min_{\boldsymbol{x}'} t f_0(\boldsymbol{x}') + \phi(\boldsymbol{x}')$ s.t. $\boldsymbol{A}\boldsymbol{x} = \boldsymbol{b}$ starting at $\boldsymbol{x}$
3: Update $\boldsymbol{x} \leftarrow \boldsymbol{x}^\star(t)$ from the previous step
4: Increase t: $t \leftarrow \tau t$
5: **end while**

Example 2.1 (Quadratic Optimisation)**.**
The standard form for quadratic optimisation is

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \tfrac{1}{2}\boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} + \boldsymbol{q}^\top \boldsymbol{x} + r \\ \text{s.t.} \quad & \boldsymbol{G}\boldsymbol{x} \preceq \boldsymbol{h} \\ & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}, \end{aligned} \tag{2.14}$$

where $\boldsymbol{x}, \boldsymbol{q} \in \mathbb{R}^n$, $\boldsymbol{P} \in \mathbb{R}^{n\times n}$, $r \in \mathbb{R}$, $\boldsymbol{G} \in \mathbb{R}^{m\times n}$, $\boldsymbol{h} \in \mathbb{R}^m$, $\boldsymbol{A} \in \mathbb{R}^{p\times n}$ *and* $\boldsymbol{0}$ *is the zero vector of appropriate size. We first note that the objective function and the constraint are differentiable and thus by theorem* (2.3) *strong duality holds. We may present* (2.14) *using the indicator trick described in this section, as*

$$\begin{aligned} \min_{\boldsymbol{x}} \quad & \tfrac{t}{2}\boldsymbol{x}^\top \boldsymbol{P}\boldsymbol{x} + t\boldsymbol{q}^\top \boldsymbol{x} + tr - \textstyle\sum_i^m \ln\left(-(\boldsymbol{G}_i^\top \boldsymbol{x} - h_i)\right) \\ \text{s.t.} \quad & \boldsymbol{A}\boldsymbol{x} = \boldsymbol{0}, \end{aligned}$$

where $\boldsymbol{G}_i$ *is the* i*th row of the matrix* $\boldsymbol{G}$ *and* h_i *is the* i*th entry in the vector* $\boldsymbol{h}$*. Thus we may solve this problem using the barrier method.*

Next we show a convergence result of the barrier method.

Theorem 2.5. Convergence in k steps

> Assuming $t f_0(\boldsymbol{x}) + \phi(\boldsymbol{x})$ can be minimised using Newton's method for $t = t^{(0)}, \tau t^{(0)}, \tau^2 t^{(0)}, \ldots$, then the desired accuracy ϵ is obtained after exactly
>
> $$k = \left\lceil \frac{\ln\left(\dfrac{m}{\epsilon t^{(0)}}\right)}{\ln(\tau)} \right\rceil .$$
>
> iterations, where $\lceil \cdot \rceil$ is the ceiling function rounding up to the nearest integer.

Proof
Assume that the stopping criterion is met, that is

$$\frac{m}{\tau^k t^{(0)}} < \epsilon$$

for $k \in \mathbb{N}$. This implies that

$$\begin{aligned} \frac{m}{\epsilon t^{(0)}} &< \tau^k \quad \Longrightarrow \\ \ln \frac{m}{\epsilon t^{(0)}} &< k \ln \tau \quad \Longrightarrow \\ \frac{\ln \dfrac{m}{\epsilon t^{(0)}}}{\ln \tau} &< k. \end{aligned}$$

Thus if we round the left side of the last equation up to the nearest integer, the desired result is obtained.

■

We have in this section introduced the barrier method for solving quadratic optimisation problems, which shall later prove useful.

3 Machine Learning

We live in a world were data is becoming more and more accessible and there is more of it. A wide range of data is collected every second from all over the world. These vast and enormous data sets are incomprehensible for a human, thus we turn to machines to make the analyses using machine learning. Machine learning is the science of fitting a specific model to a specific data set and finding patterns within said data set. Such a model can then be used for e.g. predicting the evolution of stock prices, or the classification of abnormal transactions within the banking world, dependent on the type of model fitted to the data. We shall in this chapter discuss a few techniques relevant for the over all goal of the thesis: To classify pumps as faulty or functioning. The goal of this chapter is to introduce the necessary component to describe and build a federated multi-task learning scheme. [13, 14]

3.1 Support Vector Machine

Support vector machine (SVM) is the term used to describe a maximum marginal classifier. It functions under the assumption that the data in linear separable and aims to find a hyperplane which separates the data in an desirable way. The hyperplane is chosen such the margin is maximised, see figure 3.1. Based on the hyperplane, we may classify new data and determine whether this new data most likely to belong to one or the other type. This section is, unless otherwise stated, based on [14].

We consider the N D-dimensional data points $\boldsymbol{x}_1, \ldots, \boldsymbol{x}_N \in \mathbb{R}^D$, with associated labels $t_1, \ldots, t_N \in \{-1, 1\}$, as the training data for the binary classification model

$$y(\boldsymbol{x}) = \boldsymbol{w}^\mathsf{T} \phi(\boldsymbol{x}) + b. \tag{3.1}$$

where $\phi(\cdot) : \mathbb{R}^D \to \mathbb{R}^n$ maps the data into a feature space. It is assumed that the data is linear separable in the feature space, that is, at leased one choice for $\boldsymbol{w}$ and b exists where (3.1) satisfies both

$$\begin{aligned} &y(\boldsymbol{x}_i) > 0 \ \forall i \text{ where } t_i = 1 \quad \text{and} \\ &y(\boldsymbol{x}_i) < 0 \ \forall i \text{ where } t_i = -1, \end{aligned} \tag{3.2}$$

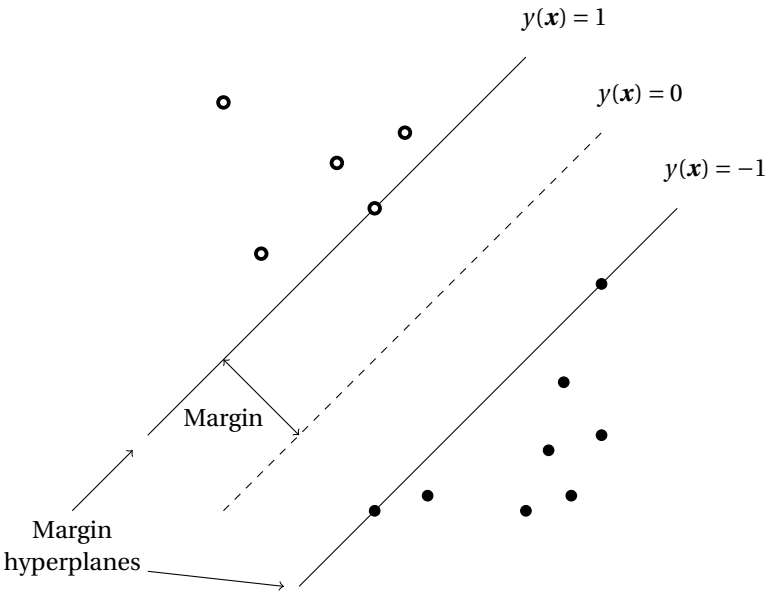


Figure 3.1. Depicts the concept of SVM.

thus also $t_i y(\boldsymbol{x}_i) > 0 \; \forall i$. This way, a new point $\boldsymbol{x}_0$ is classified as belonging to class $\mathcal{C}_1$ if $y(\boldsymbol{x}_0) < 0$ or class $\mathcal{C}_2$ otherwise. Any point $\boldsymbol{x}$ satisfying $y(\boldsymbol{x}) = 0$ is said to lie on the decision bound, the hyperplane separating the two classes.

Let $\boldsymbol{x}_a$ and $\boldsymbol{x}_b$ be any two distinct points on the decision bound. Because $y(\boldsymbol{x}_a) = y(\boldsymbol{x}_b) = 0$ we have that $\boldsymbol{w}^\top(\boldsymbol{x}_a - \boldsymbol{x}_b) = 0$. This shows that $\boldsymbol{w}$ is perpendicular to all vectors lying within the decision surface. Thereby, $\boldsymbol{w}$ controls the orientation of the decision bound.

The perpendicular distance from any point $\boldsymbol{x}$ to the decision bound is given by

$$r(\boldsymbol{x}) = \frac{|y(\boldsymbol{x})|}{\|\boldsymbol{w}\|}. \tag{3.3}$$

Note that the distance from origin to the decision bound is given by $|b|/\|\boldsymbol{w}\|$, thus b controls the placement of the decision bound. Since choices for $\boldsymbol{w}$ and b exist satisfying (3.2), we may rewrite (3.3) as follows

$$r(\boldsymbol{x}_i) = \frac{t_i y(\boldsymbol{x}_i)}{\|\boldsymbol{w}\|} = \frac{t_i(\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b)}{\|\boldsymbol{w}\|}.$$

The margin is the smallest perpendicular distance between the decision bound and any one training data point. The goal is to find $\boldsymbol{w}$ and b such this distance is maximised. Thus, we formulate the following optimisation problem

$$\underset{\boldsymbol{w},b}{\arg\max}\left(\frac{1}{\|\boldsymbol{w}\|} \min_i \left[t_i(\boldsymbol{w}^\top \phi(\boldsymbol{x}_i) + b) \right] \right). \tag{3.4}$$

Where $1/\|\boldsymbol{w}\|$ is excluded in the minimisation over i as it does not depend on i. Note that $r(\cdot)$ from (3.3) is invariant under scaling of $\boldsymbol{w}$ and b. Let i' solve $\arg\min_i \; [t_i(\boldsymbol{w}^\top \boldsymbol{x}_i + b)]$, then there exists a $\kappa \in \mathbb{R}_+$ such

$$\frac{t_i'(\kappa \boldsymbol{w}^\top \phi(\boldsymbol{x}_i') + \kappa b)}{\|\kappa \boldsymbol{w}\|} = 1,$$

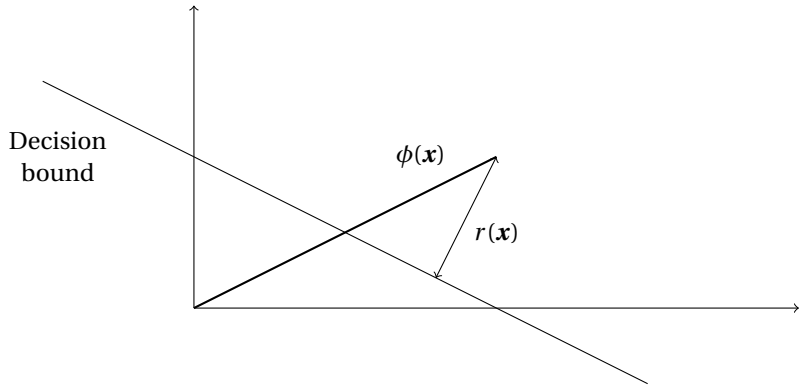


Figure 3.2. Distance to the decision bound

where κ is omitted from now on. In this case all data points will satisfy the following condition

$$\frac{t_i(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_i)+b)}{\|\boldsymbol{w}\|} \geq 1 \quad \text{for } i = 1,\dots N.$$

Thus, the optimisation problem from (3.4) can be rewritten as

$$\begin{aligned} \arg\min_{\boldsymbol{w},b} \quad & \tfrac{1}{2}\|\boldsymbol{w}\|^2 \\ \text{s.t.} \quad & t_i(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_i)+b) \geq 1 \quad \text{for } i = 1,\dots N \end{aligned} \tag{3.5}$$

where the introduction of the scalar in the objective function is for ease of derivation later on. Note that the maximisation problem has changed to a minimisation and that we examine $\|\boldsymbol{w}\|^2$ and not $\|\boldsymbol{w}\|^{-1}$.

A method for solving the constrained optimisation problem in (3.5) is to introduce Lagrangian multipliers $\mu_i \geq 0$, one multiplier for each of the constrains, giving the Lagrangian

$$\mathrm{L}(\boldsymbol{w}, b, \boldsymbol{\mu}) = \frac{1}{2}\|\boldsymbol{w}\|^2 - \sum_{i=1}^{N} \mu_i \left(t_i(\boldsymbol{w}^\mathsf{T}\phi(\boldsymbol{x}_i)+b) - 1\right), \tag{3.6}$$

where $\boldsymbol{\mu} = (\mu_1, \dots, \mu_N)$.

Differentiating (3.6) w.r.t $\boldsymbol{w}$ and b, and equating to zero yields

$$\begin{aligned} \boldsymbol{w} &= \sum_{i=1}^{N} \mu_i t_i \phi(\boldsymbol{x}) \\ 0 &= \sum_{i=1}^{N} \mu_i t_i. \end{aligned} \tag{3.7}$$

Substituting these equalities into (3.6), yields

$$\widetilde{\mathrm{L}}(\boldsymbol{\mu}) = \sum_{i=1}^{N} \mu_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N} \mu_i \mu_k t_i t_k k(\boldsymbol{x}_i, \boldsymbol{x}_k), \tag{3.8}$$

where $k(\boldsymbol{x}, \boldsymbol{x}') = \phi(\boldsymbol{x}_i)^\mathsf{T}\phi(\boldsymbol{x}_k)$, thus relieving the necessity of knowing $\phi(\boldsymbol{x})$, only requiring to know the kernel $k(\cdot,\cdot)$. Note the removing of the dependencies on $\boldsymbol{w}$ and b. Hereby, we have

the optimisation problem

$$\begin{aligned} \arg\max_{\boldsymbol{\mu}} \quad & \widetilde{\mathrm{L}}(\boldsymbol{\mu}) \\ \text{s.t.} \quad & \mu_i \geq 0, \quad \text{for } i = 1, \ldots, N \\ & 0 = \textstyle\sum_{i=1}^{N} \mu_i t_i \end{aligned} \tag{3.9}$$

It can be seen, that if the kernel function $k(\boldsymbol{x}, \boldsymbol{x}')$ is positive definite, (3.8) is bounded from below, thus (3.9) is well defined [14]. It is the solution to (3.8) that will in turn give the optimal values for $\boldsymbol{w}$ and b from (3.1). This is a quadratic optimisation problem, and we may thus reformulate (3.8) such it appears on the standard form of (2.14) as such:

$$\begin{aligned} \arg\min_{\boldsymbol{\mu}} \quad & \tfrac{1}{2}\boldsymbol{\mu}^\mathsf{T}\boldsymbol{P}\boldsymbol{\mu} - \boldsymbol{q}^\mathsf{T}\boldsymbol{\mu} \\ \text{s.t.} \quad & \boldsymbol{G}\boldsymbol{\mu} \preceq \boldsymbol{h} \\ & \boldsymbol{a}^\mathsf{T}\boldsymbol{\mu} = \boldsymbol{b}, \end{aligned} \tag{3.10}$$

where $\boldsymbol{P}_{(k,i)} = t_k t_i k(\boldsymbol{x}_k, \boldsymbol{x}_i)$, $\boldsymbol{q} = [1, 1, \ldots, 1]^\mathsf{T}$, $\boldsymbol{G} = -\boldsymbol{I}$, $\preceq$ indicates an elementwise inequality, $\boldsymbol{h} = \boldsymbol{0}$, $\boldsymbol{a} = [t_1, t_2, \ldots t_N]^\mathsf{T}$ and $\boldsymbol{b} = \boldsymbol{0}$. This also means that we now have a way of solving this problem using the barrier method.

Any minimiser of (3.9) must adhere to the Karush-Kuhn-Tucker (KKT) constraints which in this case are

$$\begin{aligned} \mu_i &\geq 0, \quad \text{for } i = 1, \ldots, N \\ t_i y(\boldsymbol{x}_i) - 1 &\geq 0, \quad \text{for } i = 1, \ldots, N \\ \mu_i \left(t_i y(\boldsymbol{x}_i) - 1 \right) &\geq 0, \quad \text{for } i = 1, \ldots, N. \end{aligned} \tag{3.11}$$

From (3.11) it is clearly seen that for every $i = 1, \ldots, N$ either $\mu_i = 0$ or $t_i y(\boldsymbol{x}_i) = 1$. Any training point $\boldsymbol{x}_i$ for which $t_i y(\boldsymbol{x}_i) = 1$ is called a support vector, hence the name support vector machines. Let $\mathcal{S}$ be the set of indices associated with the support vectors, that is $\mathcal{S} = \{i = 1, \ldots, N \mid t_i y(\boldsymbol{x}_i) = 1\}$ Thus, we know form our observation in regards to (3.11) that for any $i \in \mathcal{S}$

$$t_i \left(\sum_{m \in \mathcal{S}} \mu_m t_m k(\boldsymbol{x}_i, \boldsymbol{x}_m) + b \right) = 1. \tag{3.12}$$

The scalar b could be determine directly from (3.12) using any support vector, however, in the following we show a more numerically strong way of determining b, utilising an average over all support vector. Note that $t_i^2 = 1$ for $i = 1, \ldots, N$, thus:

$$b = \frac{1}{|\mathcal{S}|} \sum_{i \in \mathcal{S}} \left(t_i - \sum_{m \in \mathcal{S}} \mu_m t_m k(\boldsymbol{x}_i, \boldsymbol{x}_m) \right),$$

where $|\mathcal{S}|$ denotes the cardinality of $\mathcal{S}$. Thus, we have expressions for $\boldsymbol{w}$ in (3.7) and b as just shown.

Imagine having a trained model, wanting to predict what class new data points belong to. We consider the new data point $\boldsymbol{x}$ with associated label t. If $y(\boldsymbol{x}) \geq 0$ then $\boldsymbol{x}$ belongs to class $t = 1$

and vice versa. Thus we can determine whether we correctly classified $\boldsymbol{x}$ by

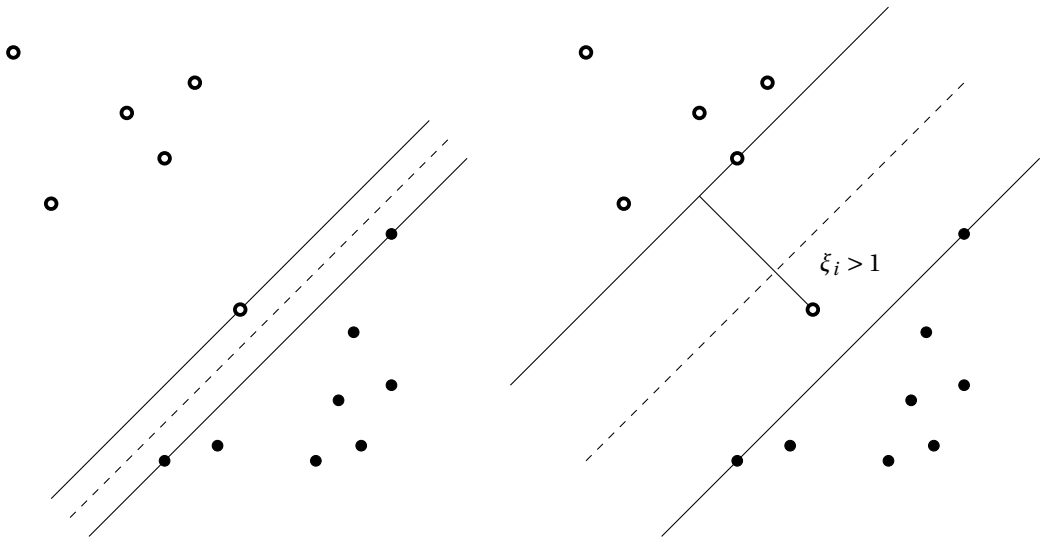
$$ty(\boldsymbol{x}) = \begin{cases} \geq 0 & \text{correctly classified} \\ < 0 & \text{incorrectly classified.} \end{cases}$$

3.1.1 Non-perfect classifiers

So far we have assumed the training data was perfectly linearly separable in the feature space, however, this is often not the case. Sometimes class distributions overlap, and if we were to create a model that correctly classified all training points, even if they were deep within the convex hull of the other class, it might lead to a model that perform worse in general. Thus, it can sometimes be advantageous to purposefully misclassify some training data to build a more robust model. If we compare the figures 3.3a and 3.3b, we see that allowing the one misclassification actually builds a more robust model, that we expect to perform much better in the long run.

We introduce slack variables $\xi_i \geq 0$ for $1 = 1, \ldots, N$, such there are one for each training data point, to allow training points to be on the "wrong" side of the margin hyperplanes, as seen on the figures 3.3a and 3.3b. Points that are on the correct side of the margin hyperplanes have a slack variable of $\xi_i = 0$, while every other point will have a slack variable of $\xi_i = |t_i - y(\boldsymbol{x}_i)|$. This implies that points on the decision bound will have $\xi_i = 1$, while points with $\xi_i > 1$ will be misclasified, just as seen on figure 3.3. Points with $0 < \xi_i < 1$ are on the wrong side of the hyperplane, yet still classified correctly. We introduce the *Hinge loss function* which does exactly all this.

$$\ell_H(\boldsymbol{x}_i, t_i) = \max(0, 1 - t_i y(\boldsymbol{x}_i)).$$



(a) Allowing no misclassification. **(b)** Allowing some misclassification.

Figure 3.3. Shows the difference between allowing and not allowing misclassification.

Therefore, we simply define $\xi_i = \ell_H(\boldsymbol{x}_i, t_i)$.

The objective is now to maximise the margin, while including penalisation of training points not correctly classified, as seen

$$\begin{aligned} \arg\min_{\boldsymbol{w},b} \quad & \tfrac{1}{2}\|\boldsymbol{w}\|^2 + C\textstyle\sum_{i=1}^{N}\xi_i \\ \text{s.t.} \quad & t_i(\boldsymbol{w}^\top\phi(\boldsymbol{x}_i)+b) \geq 1-\xi_i \quad \text{for } i = 1,\dots N \\ & \xi_i \geq 0 \quad \text{for } i = 1,\dots N \end{aligned} \tag{3.13}$$

where $C > 0$ is a scalar that controls the trade-off between the importance of a large margin and the slack variables. In the limit $C \to \infty$, (3.13) will be equivalent to (3.5), while if $C \to 0$ there will be almost no penalty for having huge slack variables, which will in most cases lead to useless models. The KKT conditions are similar, however, this time we introduced the slack ξ_i which is thus influencing the KKT conditions, as seen

$$\begin{aligned} \mu_i &\geq 0, && \text{for } i = 1,\dots,N \\ t_i y(\boldsymbol{x}_i) - 1 + \xi_i &\geq 0, && \text{for } i = 1,\dots,N \\ \mu_i\left(t_i y(\boldsymbol{x}_i) - 1 + \xi_i\right) &\geq 0, && \text{for } i = 1,\dots,N \\ \xi_i &\geq 0, && \text{for } i = 1,\dots,N \\ \lambda_i &\geq 0, && \text{for } i = 1,\dots,N \\ \lambda_i\xi_i &= 0, && \text{for } i = 1,\dots,N. \end{aligned} \tag{3.14}$$

Again, the Lagrangian is determined

$$\mathrm{L}(\boldsymbol{w}, b, \boldsymbol{\xi}, \boldsymbol{\mu}, \boldsymbol{\lambda}) = \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_{i=1}^{N}\xi_i - \sum_{i=1}^{N}\mu_i\left(t_i\left(\boldsymbol{w}^\top\phi(\boldsymbol{x}_i)+b\right) - 1 + \xi_i\right) - \sum_{i=1}^{N}\lambda_i\xi_i, \tag{3.15}$$

differentiated w.r.t $\boldsymbol{w}$, b and ξ_n, equated to zeros, yields

$$\boldsymbol{w} = \sum_{i=1}^{N}\mu_i t_i \phi(\boldsymbol{x}_i) \tag{3.16}$$

$$0 = \sum_{i=1}^{N}\mu_i t_i \tag{3.17}$$

$$\mu_i = C - \lambda_i. \tag{3.18}$$

These may be used to eliminate $\boldsymbol{w}$, b and ξ_n from (3.15) by insertion, thus yielding

$$\widetilde{\mathrm{L}}(\boldsymbol{\mu}) = \sum_{i=1}^{N}\mu_i - \frac{1}{2}\sum_{i=1}^{N}\sum_{k=1}^{N}\mu_i\mu_k t_i t_k k(\boldsymbol{x}_i, \boldsymbol{x}_k). \tag{3.19}$$

Next, we optimise (3.19)

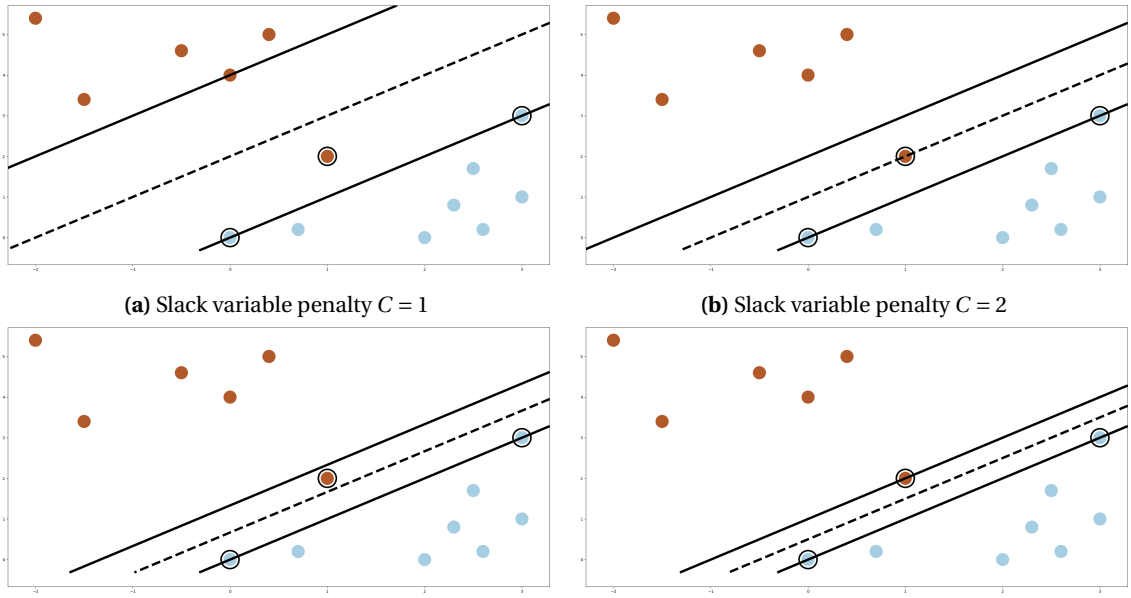
$$\begin{aligned} \arg\max_{\boldsymbol{\mu}} \quad & \widetilde{\mathrm{L}}(\boldsymbol{\mu}) \\ \text{s.t.} \quad & 0 \leq \mu_i \leq C, \quad \text{for } i = 1,\dots,N \\ & 0 = \textstyle\sum_{i=1}^{N}\mu_i t_i. \end{aligned}$$

Note that this is again a quadratic optimisation problem, and can be reformulated similar to (3.10), with the only difference being $\boldsymbol{G} = [-\boldsymbol{I}^\top, \boldsymbol{I}^\top]^\top$ and $\boldsymbol{h} = [\boldsymbol{0}^\top, C, C, \dots C]^\top$.

Any training point $\boldsymbol{x}_i$ for which $\mu_i < C$, we have by (3.18) that $\lambda_i > 0$, and thus by KKT conditions in (3.14) $\xi_i = 0$. Hereby, such $\boldsymbol{x}_i$ is correctly classified and lie on the correct side of the margin or on the margin. However, points for which $\mu_i = C \implies \lambda_i = 0$, thus lie within the margin and are correctly classified if $\xi_i \leq 1$ or misclassified if $\xi_i > 1$. Lastly we simply need to determine b, which again will be done through an average. Any training point $\boldsymbol{x}_i$ for which $0 < \mu_i < C$ will have $\xi_i = 0$ implying that $t_i y(\boldsymbol{x}_i) = 1$, thus we can determine b as,

$$b = \frac{1}{|\mathcal{M}|} \sum_{m \in \mathcal{M}} \left(t_m - \sum_{i \in \mathcal{S}} \mu_i t_i k(\boldsymbol{x}_i, \boldsymbol{x}_m) \right),$$

where $\mathcal{M} = \{i \in [1, N] | 0 < \mu_i < C\}$ and $|\mathcal{M}|$ denotes the cardinality of $\mathcal{M}$.



(a) Slack variable penalty $C = 1$

(b) Slack variable penalty $C = 2$

(c) Slack variable penalty $C = 3$

(d) Slack variable penalty $C = 4$

Figure 3.4. Experiments with four different penalty coefficients, where the support vectors are circled.

One thing that has yet to be discussed is the choice of the penalty coefficient, C. Figure figure 3.4 depicts the result of four learning instances of the SVM discussed in this section, using four different values for C. The learning has been done with the use the sklearn module for python [15]. As can be seen, four different results with different placements of the decision bound and widths of the margin are obtained. Dependent on the application, one has to experiment and determine a suitable value for the penalty coefficient.

3.2 Federated Learning

Federated learning is the concept of learning a single model from the average of multiple models all trained on distinct data sets $\mathcal{D}_1, \mathcal{D}_2 \ldots \mathcal{D}_K$ [16]. An example of this could be if a mobile phone company wants to help their customers write better text messages. Specifically, help include apostrophes in contractions such as "I've". The traditional way of considering such a problem, would be to communicate all the data, i.e. the text messages, to a central server and commence with the learning once all the data had been collected and stored on the server [16]. However, this way of considering the problem has its downsides. The communication between the devices and the server is time consuming and inefficient. If we instead move the learning process to the individual devices (nodes, as they will be referred to from now on) we save the transmission cost of the entire data sets. Instead, only the weights - a set of parameters that entirely describe the model - are communicated which will be of a far less cost. The concept is depicted in figure 3.5.

In the text message case, we are unable to share the data with a central server due to a different and yet maybe even more important concern; privacy. However, with the recent strides in mobile technology we luckily do not have to, since today's mobile phones have huge computational power compared those of past decades, which allows us to perform the learning on the nodes [7]. We shall from now on refer to a single text message as a data point. As one can imagine, the frequency of data points being collected will be different from node to node, as people write text messages with different frequency. Likewise, people write different styles of text massages, thus the data will in its nature will be different. Thus, we can conclude, that a solution to this problem must be able to handle non-IID and unbalanced data sets.

3.2.1 The Federated Averaging Algorithm

This problem is considered as one problem, meaning that in the end a single weight vector is found, even though the data separated into K parts (one for each devise), all of which are val-

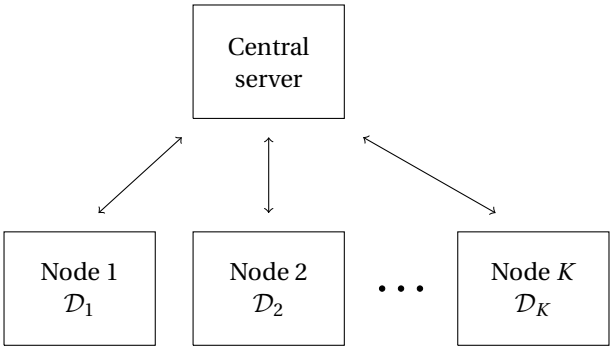


Figure 3.5. Depicts the star network setup of the federated learning.

ued equally. It makes sense to consider the problem of minimising the average over all the loss functions, as seen below. We seek to solve the following problem

$$\min_{\boldsymbol{w}} \quad \frac{1}{N}\sum_{i=1}^{N} f_i(\boldsymbol{w}), \tag{3.20}$$

where $\boldsymbol{w} \in \mathbb{R}^d$ are the weights of the global model, $N \in \mathbb{R}$ is the total number of data points and $f_i(\boldsymbol{w}) = \ell(\boldsymbol{x}_i, \boldsymbol{w})$ is the loss obtained from training the model using the training data $\boldsymbol{x}_i$, with $\ell : \mathbb{R}^d \times \mathbb{R}^m \to \mathbb{R}_+$. That is, we choose a model, then seek to find the weights $\boldsymbol{w}$ that best fit the entire data set. Unless otherwise stated, this section is based on [17].

However, as previously described, the data in generated and stored on K nodes, such that they each have a distinct data set $\mathcal{D}_k$. Thus, (3.20) is rewritten as

$$\min_{\boldsymbol{w}} \quad \sum_{k=1}^{K} \frac{n_k}{N} F_k(\boldsymbol{w}), \quad \text{where} \quad F_k(\boldsymbol{w}) = \frac{1}{n_k}\sum_{i\in\mathcal{D}_k} f_i(\boldsymbol{w}) \tag{3.21}$$

and $n_k = |\mathcal{D}_k|$. This means that we cannot directly solve (3.20), as we are not allowed to share data. We instead turn to solving (3.21), that is solving K optimisation problem by fitting the same type of model to the K distinct data sets $\mathcal{D}_1, \ldots, \mathcal{D}_K$.

One way of solving (3.21) would be to take a gradient descent (abbr. GD) approach. GD is a technique that finds the gradient of the loss function and updates the weights by subtracting the gradient of the loss function scaled with a learning step constant η [12]. This is then repeated a number of times until the result is sufficient. One could also have approach this problem using a SVM solution. In the case of a GD solution, it would be to find the gradient of $F_k(\boldsymbol{w}_m)$ at the mth iteration, that is

$$\boldsymbol{g}_k = \nabla F_k(\boldsymbol{w}_m) \quad \text{for} \quad k = 1, \ldots, K.$$

In our federated setting, the updates of the weights of the global model would then happen as

$$\boldsymbol{w}_{m+1} = \boldsymbol{w}_m - \eta \sum_{k=1}^{K} \frac{n_k}{N} \boldsymbol{g}_k. \tag{3.22}$$

This way of doing the update is not arbitrary, as it has strong links to the original problem of (3.20):

$$\begin{aligned}
\sum_{k=1}^{K} \frac{n_k}{N} \boldsymbol{g}_k &= \sum_{k=1}^{K} \frac{n_k}{N} \nabla F_k(\boldsymbol{w}) \\
&= \nabla \sum_{k=1}^{K} \frac{n_k}{N} F_k(\boldsymbol{w}) \\
&= \nabla \frac{1}{N} \sum_{i=1}^{N} f_i(\boldsymbol{w}),
\end{aligned}$$

which is the gradient of the objective function. This seems natural, as we are considering a GD approach.

If we examine (3.22), it may be rewritten as

$$\begin{aligned}
\boldsymbol{w}_{m+1} &= \boldsymbol{w}_m - \eta \sum_{k=1}^{K} \frac{n_k}{N} \boldsymbol{g}_k \\
&= \sum_{k=1}^{K} \frac{n_k}{N} \boldsymbol{w}_m - \eta \sum_{k=1}^{K} \frac{n_k}{N} \boldsymbol{g}_k \\
&= \sum_{k=1}^{K} \frac{n_k}{N} (\boldsymbol{w}_m - \eta \boldsymbol{g}_k) \\
&= \sum_{k=1}^{K} \frac{n_k}{N} \boldsymbol{w}_{m+1}^k,
\end{aligned}$$

where $\boldsymbol{w}_{m+1}^k = \boldsymbol{w}_m - \eta \boldsymbol{g}_k$. This shows that instead of each node communicating the derivative of its loss function, the nodes may communicate their updated weights and the server then averages the weights to achieve the weights of the global model. This also opens up for the opportunity of having more than one update of the weights on the nodes before updating the global model at the central serer. We refer to E as the number of local epochs on each node. Note that we distinct between iterations and local epochs. An iteration is one full round through algorithm 2, while a local epoch is one round through algorithm 3.

If there are a large number of nodes to base an update upon, it may advantageous to randomly choose some of them instead. This concept is also seen in techniques such as the stochastic gradient descents (abbr. SGD). We refer to $0 \leq C \leq 1$ as the fraction of nodes to base the update upon. This is included as it could be of interest to make global updates on only some of the devises. In the mobile phone case, with an enormous amount of devises it could take a long time to receive updates from them all. Thus, we randomly choose only some of them. The amount is decided by the value of C.

It may also be advantageous to split the data sets into smaller mini batches of data and only use a single mini batch at a time in the on-node updates, similar to SGD. This is different from having more than one epoch on the node. The idea is to split the data $\mathcal{D}_k$ of the kth node into minim batches of size of B. Then update the weights of the node on the mini batches, one at a time. Note that $B = \infty$ means the entire local data set is used, thus we do not split the data set into smaller mini batches at all.

Next we present the *Federated Average Algorithm*.

Algorithm 2 The Federated Averaging Algorithm [17]

Input: Number of local epochs E, the learning rate η, the size of local mini batches B, the fraction of nodes to receive updates from C, the number of nodes K and initial weights $\boldsymbol{w}_0$

Output: The weights for a global model $\boldsymbol{w}$

for each communication round $t = 1, 2, \ldots$ **do**
  $m \leftarrow \max(C \cdot K, 1)$
  $\mathcal{S}_t \leftarrow$(random set of m indices)
  for each node $k \in \mathcal{S}_t$ in parallel **do**
    $\boldsymbol{w}_{t+1}^k \leftarrow \texttt{NodeUpdate}(k, E, \eta, B, \boldsymbol{w}_t)$
  end for
  $\boldsymbol{w}_{t+1} \leftarrow \sum_{k=1}^{|\mathcal{S}_t|} \frac{n_k}{N} \boldsymbol{w}_{t+1}^k$
end for

Where `NodeUpdate`$(\cdot)$ as the name suggests is the function that updates the local weights on the nodes. The `NodeUpdate` function is given as.

Algorithm 3 `NodeUpdate` Algorithm [17]

Input: The node number k, the number of local epochs E, the learning rate η, the size of local mini batches B and the current weights $\boldsymbol{w}_t$.

Output: The updated weights from node k: $\boldsymbol{w}_{t+1}^k$.

$\mathcal{B} \leftarrow$ (randomly split $\mathcal{D}_k$ into mini batches of size B)
for each local epoch $i = 1, 2, \ldots E$ **do**
  for each mini batch $\boldsymbol{b} \in \mathcal{B}$ **do**
    $\boldsymbol{w} \leftarrow \boldsymbol{w} - \eta \nabla \ell(\boldsymbol{b}, \boldsymbol{w})$
  end for
end for

Note that $C = 0.0$ means the update is based upon a single node.

Example 3.1 (Federated Learning Using SVM)**.**
Imagine wanting to classify the gender of mobile device users based on the text messages written. One way of approaching a solution to this obvious classification problem is using the SVM technique and the federated averaging algorithm, both previously discussed. Recall the loss function for an SVM scheme is the Hinge loss function, given as:

$$\ell_H(\boldsymbol{b}_i, \boldsymbol{w}) = \max(0, 1 - t_i\, y(\boldsymbol{b}_i)),$$

where $\boldsymbol{b}_i$ is the ith data point in the training set with associated label t_i and $\boldsymbol{w}$ are the weights needed for the classification. Depicted in figure 3.6 is $\ell_H(\cdot, \cdot)$ as a function of $t_i\, y(\boldsymbol{b}_i)$.

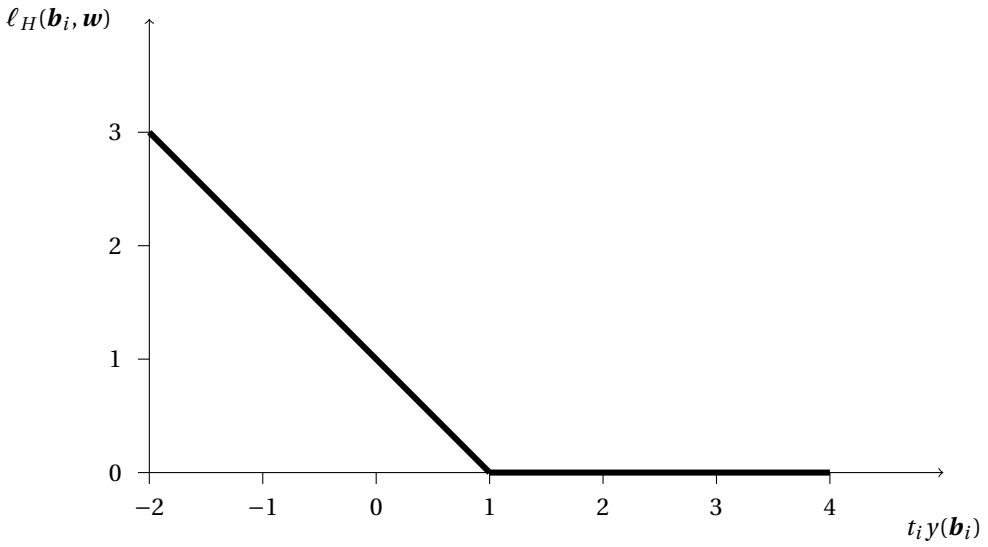


Figure 3.6. Depicts the Hinge loss function as a function of the product between the output of the decision function and the true label of the data point.

From this visual inspection it is clear that the function is not differentiable, however, it is piecewise differentiable. Thus,

$$\nabla \ell_H(\boldsymbol{b}_i, \boldsymbol{w}) = \begin{cases} -t_i \phi(\boldsymbol{b}_i) & t_i\, y(\boldsymbol{b}_i) < 1 \\ 0 & \textit{otherwise} \end{cases},$$

where we have defined the derivative to be zero if $t_i\, y(\boldsymbol{b}_i) = 1$. Thus we have the necessary tool to construct federated learning scheme to solve this problem using SVM on the nodes.

3.2.2 Parameter Discussion

We have so far introduced the Federated Averaging algorithm, given an example of an applicable case and given some parameters which need to be specified. We begin by discussing the parallelism i.e. the C parameter. This parameter is included in the algorithm to minimise the communication needed for the algorithm to run. That is, the fewer nodes the central server need to communicate with, the less of a communication cost occurs. On the other hand, intuitively having a large C updates the model bases upon more data, thus should yield a faster convergence in terms of numbers of communication rounds. One thing, yet to be discussed is the fact that one cannot expect all nodes to be available all the time. This could be if the mobile phone company has agreed to only do learning updates during the night, avoiding user frustration due to allocated processing resources slowing down the mobile phone, had the learning update been done during the hours of which the phone is operated by the user. Thus choosing C large would increase the expected time a single communication round takes, leading to slower convergence. From the experiments conducted in [16, 17] a conclusion is drawn

that choosing $C = 0.1$ strikes a good balance between the number of communication rounds needed and the computational efficiency (i.e. the availability of nodes).

The number of computations on each nodes is controlled both by the mini batch size B and the number of local epochs E, thus these two parameter are connected and it makes sense to discuss these simultaneously. The way the `NodeUpdate` algorithm is constructed, the expected number of local computations is given as:

$$u = \left(\frac{\mathbb{E}[n_k]}{B}\right) E = E\left(\frac{N}{KB}\right),$$

where $\mathbb{E}[\cdot]$ is the expectation operator. Having larger u generally leads to fewer communication rounds [16, 17], thus we are interested in having large u. We may increase u by either increasing E or decreasing B. The parameter B should always be large enough to take full advantage of the parallelism of the node hardware, meaning that the node hardware should be fully occupied at all times during the node update. That is, if B is chosen sufficiently small, some of the computational power on the node might not be used to full capacity. However, that said, if B is large enough (such the parallelism on the node is fully used), there is essentially no computational cost in decreasing B [16, 17]. Recall that decreasing B leads to more local computations, increasing parallelism of the entire learning scheme. Thus, B should be the first parameter to be tuned [16, 17]. On the other hand, a larger u puts larger constraints on the hardware available on the nodes as more computations are needed. Obviously, not all nodes (dependent on the application) will have the same computational speed. In the mobile phone case, one or more mobile phones might be older and thus their computational power will be inferior to some of the newer ones. This will lead to a slower convergence in terms of time. One has to consider the application when choosing both B and E: Is it important to keep the number of communication rounds low, sacrificing the convergence in terms of time, or does the application allow for a larger number of communication rounds, having less local computations.

3.3 Multi-Task Learning

Multi-task learning (abbr. MTL) is the concept of learning a number of tasks faster or more proficiently by learning them simultaneously, rather than if the tasks were learned independently [18]. The goal is to learn T task, all equally important, simultaneously. That is, to build T models with model parameters $\boldsymbol{\theta}_1, \ldots, \boldsymbol{\theta}_T \in \mathbb{R}^n$. However, to have multi-task, one needs to know what a single task is.

Definition 3.1. A task

We define the tth task in a MTL scheme as

$$\mathcal{T}_t \triangleq \{\mathcal{D}_t,\ \mathcal{L}_i(\boldsymbol{\theta}_i, \boldsymbol{x}, \boldsymbol{y})\}, \tag{3.23}$$

where $\mathcal{D}_t = \{(\boldsymbol{x}_{1,t}, \boldsymbol{y}_{1,t}), (\boldsymbol{x}_{2,t}, \boldsymbol{y}_{2,t}), \ldots, (\boldsymbol{x}_{N_t,t}, \boldsymbol{y}_{N_t,t})\}$ is the data set, N_t denotes the number of

data point in the data set, $\mathcal{L}_t(\cdot,\cdot,\cdot)$ is the loss function and $\boldsymbol{\theta}_i \in \mathbb{R}^m$ is the model parameters all associated with the tth task.

Note that the terms in a task are indexed, which means that they could all be distinct for each task. That is, one might consider a MTL scheme of classifying images from clothes shopping websites. This could be the case where one wanted to classify the colour of the product (on the image) and the type of clothes (blouse, dress, shirt etc.) as two tasks. In these cases the data points and the loss function (possibly cross-entropy) would be the same across the two tasks. However, the labels would differ.

To learn T tasks faster or more proficiently we assume that tasks are related.

Definition 3.2. Related Tasks

Two tasks are said to be related if the models perform well while sharing some model parameters.

This means that we may split each model $\boldsymbol{\theta}_1, \ldots \boldsymbol{\theta}_T$ into a part that is shared and one that is task specific. If none of the chosen T tasks are related, then multi-task learning could yield a worse result, than learning the tasks individually. This is due to the fact that, the multi-task learning scheme enforces similarity between the tasks - which is not there in this case - as we shall see later.

A simple formulation of a MTL scheme is to minimise the sum of the loss over each task

$$\min_{\boldsymbol{\Theta}} \quad \sum_{t=1}^{T} \sum_{n=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{\theta}_t,\, \boldsymbol{x}_{n,t},\, y_{n,t}\right), \tag{3.24}$$

where $\boldsymbol{\theta}_t$ is the tth column of $\boldsymbol{\Theta} \in \mathbb{R}^{m \times T}$ and the model parameters for the tth task. Equation (3.24) is the very basic concept of MTL, however, we often include regularisation terms and constraints to penalise undesirable behaviour of the model parameters [18]. Different regularisation terms and constraints are used dependent on what outcome is desired. Some of the possible approaches include a low-rank approach, task clustering approach and task grouping approach [18]. We shall, however, investigate another approach, described in the following subsection.

3.3.1 Feature learning approach

In a feature learning approach, the model tries to learn a feature space that is useful for the problem. This could be a linear transformation of the data before the model parameters are applied [18]. We have seen this concept in section 3.1 with the $\phi(\cdot)$ function, that brings the data into a feature space where it is linear separable. Solely using a linear transformation has its limitations, as we are thus only able to achieve something similar to principal component analysis or linear discriminant analysis. We shall in the later investigate the possibility of including a nonlinear transformation of the data.

This way our baseline problem (3.24) becomes [18]

$$\begin{aligned} \min_{\boldsymbol{\Theta},\boldsymbol{U}} \quad & \sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{\theta}_t,\, \boldsymbol{U}^\top \boldsymbol{x}_{n,t},\, y_{n,t}\right) + R(\boldsymbol{\Theta}) \\ \text{s.t.} \quad & \boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{I}, \end{aligned} \tag{3.25}$$

where $\boldsymbol{U} \in \mathbb{R}^{m\times m}$ is the transformation matrix, $R(\boldsymbol{\Theta})$ is the regularisation terms which we shall discuss later and $\boldsymbol{I}$ is the identity matrix of appropriate size. Note the constraint of $\boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{I}$, this ensures that the transformation matrix is orthogonal.

Next we concern ourselves with the choice of regularisation term.

Definition 3.3.

For $\boldsymbol{B} \in \mathbb{R}^{k\times l}$ and $p, q \in \mathbb{N}$ such $p, q > 0$, define

$$\|\boldsymbol{B}\|_{p,q} = \| \, \|\boldsymbol{b}^1\|_p, \ldots, \|\boldsymbol{b}^k\|_p \, \|_q,$$

where $\boldsymbol{b}^i$ is the ith row of $\boldsymbol{B}$.

We choose our regularisation term as $R(\boldsymbol{\Theta}) = \lambda\|\boldsymbol{\Theta}\|_{2,1}^2$, for $\lambda \in \mathbb{R}$. Choosing $\|\cdot\|_{2,1}$ enforces the model parameters to be similar across tasks and sparsity within model parameters. That is, the enforcement of a few large model parameters and similarity of model parameters across the tasks. Assume that the prediction function is a linear function given as

$$\mathcal{P}(\boldsymbol{x}, \boldsymbol{\theta}) = \boldsymbol{\theta}^\top \boldsymbol{x}. \tag{3.26}$$

The loss function is modified to simply take the prediction as input, and we have the MTL problem of

$$\begin{aligned} \min_{\boldsymbol{\Theta},\boldsymbol{U}} \quad & \sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{\theta}_t^\top \boldsymbol{U}^\top \boldsymbol{x}_{n,t},\, y_{n,t}\right) + \lambda\|\boldsymbol{\Theta}\|_{2,1}^2 \\ \text{s.t.} \quad & \boldsymbol{U}\boldsymbol{U}^\top = \boldsymbol{I}. \end{aligned} \tag{3.27}$$

Unfortunately (3.27) is non-smooth and non-convex [9]. We shall later show how to combat these issues, however, first we introduce a lemma which plays i vital part in a later result.

Lemma 3.1.

For any $\boldsymbol{b} = [b_1, \ldots, b_d]^\top \in \mathbb{R}^d$, such that $b_i \neq 0 \; \forall i = 1, \ldots, d$, we have

$$\min_{\boldsymbol{\lambda}} \left\{ \sum_{i=1}^{d} \frac{b_i^2}{\lambda_i} \;:\; \lambda_i > 0 \; \forall i \in [1,d],\; \sum_{i=1}^{d} \lambda_i \leq 1 \right\} = \|\boldsymbol{b}\|_1^2 \tag{3.28}$$

where $\boldsymbol{\lambda} = [\lambda_1, \ldots, \lambda_d]^\top$, and the minimiser is given as

$$\boldsymbol{\lambda}^\star = \left[\frac{|b_1|}{\|\boldsymbol{b}\|_1}, \ldots, \frac{|b_d|}{\|\boldsymbol{b}\|_1} \right].$$

Proof

We begin by showing that $\|\boldsymbol{b}\|_1^2$ is an obtainable lower bound to (3.28).

$$\begin{aligned}
\|\boldsymbol{b}\|_1 &= \sum_{i=1}^{d} |b_i| \\
&\overset{(a)}{=} \sum_{i=1}^{d} \lambda_i^{1/2} \lambda_i^{-1/2} |b_i| && (3.29) \\
&\overset{(b)}{\leq} \left(\sum_{i=1}^{d} \lambda_i \right)^{1/2} \left(\sum_{i=1}^{d} \lambda_i^{-1} |b_i|^2 \right)^{1/2} && (3.30) \\
&\overset{(c)}{\leq} \left(\sum_{i=1}^{d} \lambda_i^{-1} |b_i|^2 \right)^{1/2}, && (3.31)
\end{aligned}$$

where (a) comes from the multiplication of $1 = \lambda_i^{1/2} \lambda_i^{-1/2}$ $\forall i \in [1, d]$, (b) comes from the square root of the Cauchy-Schwarz inequality and (c) comes from the fact that $\sum_{i=1}^{d} \lambda_i \leq 1$. This implies that

$$\|\boldsymbol{b}\|_1^2 \leq \sum_{i=1}^{d} \frac{|b_i|^2}{\lambda_i},$$

showing that $\|\boldsymbol{b}\|_1^2$ is an obtainable lower bound. Next we show conditions for equality in (3.30) and (3.31). For equality to hold in (3.31) we simply require that $\sum_{i=1}^{d} \lambda_i = 1$. The Cauchy-Schwarz inequality has equality if and only if

$$\lambda_i^{1/2} = k \left(\lambda_i^{-1/2} |b_i| \right),$$

for some $k \in \mathbb{R}$. Thus, (3.30) will have equality if and only if

$$\frac{\lambda_i^{1/2}}{\lambda_i^{-1/2} |b_i|} = \frac{\lambda_j^{1/2}}{\lambda_j^{-1/2} |b_j|} \quad \forall i, j = 1, \ldots, d.$$

Finally, we show that the solution

$$\boldsymbol{\lambda}^\star = \left[\frac{|b_1|}{\|\boldsymbol{b}\|_1}, \ldots, \frac{|b_d|}{\|\boldsymbol{b}\|_1} \right]$$

fulfils all requirements. First,

$$\begin{aligned}
\sum_{i=1}^{d} \frac{|b_i|}{\|\boldsymbol{b}\|_1} &= \frac{1}{\|\boldsymbol{b}\|_1} \sum_{i=1}^{d} |b_i| \\
&= \frac{\|\boldsymbol{b}\|_1}{\|\boldsymbol{b}\|_1} \\
&= 1.
\end{aligned}$$

Secondly,

$$\begin{aligned}
\frac{\lambda_i^{1/2}}{\lambda_i^{-1/2} |b_i|} &= \frac{\lambda_i}{|b_i|} \\
&= \frac{\frac{|b_i|}{\|\boldsymbol{b}\|_1}}{|b_i|} \\
&= \frac{1}{\|\boldsymbol{b}\|},
\end{aligned}$$

is a constant independent of index and the proof is complete. ■

With the lemma introduced, we next introduce a theorem that states (3.27) may be rewritten into a convex and smooth equivalent version [9].

Theorem 3.1.

The minimisation problem

$$\begin{aligned} \min_{\boldsymbol{\Theta},\boldsymbol{U}} \quad & \sum_{t=1}^{T}\sum_{n=1}^{N_t}\mathcal{L}_t\left(\boldsymbol{\theta}_t^\intercal \boldsymbol{U}^\intercal \boldsymbol{x}_{n,t},\ y_{n,t}\right) + \lambda\|\boldsymbol{\Theta}\|_{2,1}^2 \\ \text{s.t.} \quad & \boldsymbol{U}\boldsymbol{U}^\intercal = \boldsymbol{I}. \end{aligned} \tag{3.32}$$

is equivalent to the minimisation problem

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{D}} \quad & \sum_{t=1}^{T}\sum_{n=1}^{N_t}\mathcal{L}_t\left(\boldsymbol{w}_t^\intercal \boldsymbol{x}_{n,t},\ y_{n,t}\right) + \lambda\,\mathrm{tr}(\boldsymbol{W}^\intercal \boldsymbol{D}^{+}\boldsymbol{W}) \\ \text{s.t.} \quad & \mathrm{tr}(\boldsymbol{D}) \leq 1, \quad \boldsymbol{D} \succeq \boldsymbol{0}, \quad \mathrm{range}(\boldsymbol{W}) \subseteq \mathrm{range}(\boldsymbol{D}), \end{aligned} \tag{3.33}$$

where $\boldsymbol{D}^{+}$ is the pseudoinverse of $\boldsymbol{D}$, $\mathrm{tr}(\cdot)$ is the trace operator and $\boldsymbol{B} \succeq \boldsymbol{C}$ means $\boldsymbol{B} - \boldsymbol{C}$ is positive semi definite. In particular, if $(\boldsymbol{U}^\star, \boldsymbol{\Theta}^\star)$ is a minimiser of (3.32) then

$$(\boldsymbol{W}^\star, \boldsymbol{D}^\star) = \left(\boldsymbol{U}^\star\boldsymbol{\Theta}^\star, \boldsymbol{U}^\star\,\mathrm{diag}\left(\left[\frac{\|\boldsymbol{\theta}^{1\star}\|_2}{\|\boldsymbol{\Theta}^\star\|_{2,1}}, \ldots, \frac{\|\boldsymbol{\theta}^{n\star}\|_2}{\|\boldsymbol{\Theta}^\star\|_{2,1}}\right]^\intercal\right)\boldsymbol{U}^{\star\intercal}\right),$$

is a minimiser of (3.33), where $\mathrm{diag}(\cdot)$ forms a diagonal matrix with the elements of the vector input and $\boldsymbol{\theta}^{i\star}$ denotes the ith row of the matrix $\boldsymbol{\Theta}^\star$. Conversely, if $(\boldsymbol{W}^\star, \boldsymbol{D}^\star)$ is a minimiser of (3.33), then any $(\boldsymbol{\Theta}, \boldsymbol{U})$ such $\boldsymbol{U}$ forms an orthonormal basis of the eigenvectors of $\boldsymbol{D}^\star$ and $\boldsymbol{\Theta} = \boldsymbol{U}^\intercal\boldsymbol{W}^\star$ will minimise (3.32).

Proof

Assume that $(\boldsymbol{U}, \boldsymbol{\Theta})$ is in the feasible set of (3.32), that $\boldsymbol{W} = \boldsymbol{U}\boldsymbol{\Theta}$ and that

$$\boldsymbol{D} = \boldsymbol{U}\,\mathrm{diag}\left(\left[\frac{\|\boldsymbol{\theta}^{1}\|_2}{\|\boldsymbol{\Theta}\|_{2,1}}, \ldots, \frac{\|\boldsymbol{\theta}^{n}\|_2}{\|\boldsymbol{\Theta}\|_{2,1}}\right]^\intercal\right)\boldsymbol{U}^\intercal.$$

Firstly, we show that indeed the objective functions of (3.32) and (3.33) are equal. Using the assumptions above, then

$$\sum_{t=1}^{T}\sum_{n=1}^{N_t}\mathcal{L}_t\left(\boldsymbol{\theta}_t^\intercal\boldsymbol{U}^\intercal\boldsymbol{x}_{n,t},\ y_{n,t}\right) = \sum_{t=1}^{T}\sum_{n=1}^{N_t}\mathcal{L}_t\left(\boldsymbol{w}_t^\intercal\boldsymbol{x}_{n,t},\ y_{n,t}\right).$$

And,

$$\begin{aligned}
\operatorname{tr}\left(\boldsymbol{W}^{\mathsf{T}}\boldsymbol{D}^{+}\boldsymbol{W}\right) &\overset{(a)}{=} \operatorname{tr}\left(\boldsymbol{\Theta}^{\mathsf{T}}\boldsymbol{U}^{\mathsf{T}}\boldsymbol{U}\operatorname{diag}\left(\|\boldsymbol{\Theta}\|_{2,1}\left[\|\boldsymbol{\theta}^1\|_2^+,\ldots,\|\boldsymbol{\theta}^n\|_2^+\right]^{\mathsf{T}}\right)\boldsymbol{U}^{\mathsf{T}}\boldsymbol{U}\boldsymbol{\Theta}\right) \\
&\overset{(b)}{=} \|\boldsymbol{\Theta}\|_{2,1}\operatorname{tr}\left(\boldsymbol{\Theta}^{\mathsf{T}}\operatorname{diag}\left(\left[\|\boldsymbol{\theta}^1\|_2^+,\ldots,\|\boldsymbol{\theta}^n\|_2^+\right]^{\mathsf{T}}\right)\boldsymbol{\Theta}\right) \\
&\overset{(c)}{=} \|\boldsymbol{\Theta}\|_{2,1}\operatorname{tr}\left(\operatorname{diag}\left(\left[\|\boldsymbol{\theta}^1\|_2^+,\ldots,\|\boldsymbol{\theta}^n\|_2^+\right]^{\mathsf{T}}\right)\boldsymbol{\Theta}\boldsymbol{\Theta}^{\mathsf{T}}\right) \\
&\overset{(d)}{=} \|\boldsymbol{\Theta}\|_{2,1}\sum_{i=1}^{n}\|\boldsymbol{\theta}^i\|_2^+\|\boldsymbol{\theta}^i\|_2^2 \\
&= \|\boldsymbol{\Theta}\|_{2,1}\sum_{i=1}^{n}\|\boldsymbol{\theta}^i\|_2 \\
&\overset{(e)}{=} \|\boldsymbol{\Theta}\|_{2,1}^2,
\end{aligned}$$

where (a) comes from substitution of the definitions of both $\boldsymbol{W}$ and $\boldsymbol{D}$ (Note that $\boldsymbol{U}$ is orthogonal, thus $\boldsymbol{U}^{+} = \boldsymbol{U}^{\mathsf{T}}$), (b) comes from the fact that $\operatorname{tr}(c\boldsymbol{A}) = c\operatorname{tr}(\boldsymbol{A})$ for $c \in \mathbb{R}$ and $\boldsymbol{A} \in \mathbb{R}^{h\times g}$, (c) comes from the fact that the trace operator is invariant under cyclic permutations, (d) comes from the definition of the trace operator, the fact that the diagonal of $\boldsymbol{\Theta}\boldsymbol{\Theta}^{\mathsf{T}}$ is the squared ℓ_2-norm of the rows of $\boldsymbol{\Theta}$ and that the diagonal of the matrix-product between a diagonal matrix and any other matrix (of appropriate size) is simply the product of the diagonal entries, and (e) comes from definition 3.3. Thus, we have now shown that the objective functions of (3.32) and (3.33) are equal, when $\boldsymbol{W}$ and $\boldsymbol{D}$ are defined based on a feasible $(\boldsymbol{U},\boldsymbol{\Theta})$ of (3.32).

Notice that $\boldsymbol{W}$ is a matrix multiple of the submatrix of $\boldsymbol{U}$ which corresponds to the nonzero rows of $\boldsymbol{\Theta}$. This is easily seen if we without loss of generality (abbr. wlog) assume that the first $k \in [1,n]$ rows of $\boldsymbol{\Theta}$ are zero. Then we may represent $\boldsymbol{W}$ in a block structure:

$$\boldsymbol{W} = \begin{bmatrix}\boldsymbol{U}_1 & \boldsymbol{U}_2\end{bmatrix}\begin{bmatrix}\boldsymbol{\Theta}_1 \\ \boldsymbol{\Theta}_2\end{bmatrix},$$

where $\boldsymbol{\Theta}_1$ represents the first k rows of $\boldsymbol{\Theta}$ and is thus zero. This means that $\boldsymbol{W} = \boldsymbol{U}_2\boldsymbol{\Theta}_2$. Since we assumed that the first k rows of $\boldsymbol{\Theta}$ were zero, we may also rewrite $\boldsymbol{D}$, as

$$\boldsymbol{D} = \boldsymbol{U}_2\operatorname{diag}\left(\left[\frac{\|\boldsymbol{\theta}^{k+1}\|_2}{\|\boldsymbol{\Theta}\|_{2,1}},\ldots,\frac{\|\boldsymbol{\theta}^{n}\|_2}{\|\boldsymbol{\Theta}\|_{2,1}}\right]^{\mathsf{T}}\right)\boldsymbol{U}_2^{\mathsf{T}}. \tag{3.34}$$

This shows that $\boldsymbol{W}$ is also a matrix multiple of the submatrix of $\boldsymbol{U}$ which corresponds to the nonzero eigenvalues of $\boldsymbol{D}$, as (3.34) is an eigenvalue decomposition. Thus, we have the following relation: $\operatorname{range}(\boldsymbol{W}) \subseteq \operatorname{range}(\boldsymbol{D})$.

So far we have shown that for a feasible $(\boldsymbol{U},\boldsymbol{\theta})$, the objective function in (3.32) is equal the objective function in (3.33) for appropriate $(\boldsymbol{W},\boldsymbol{D})$. However, this choice for $(\boldsymbol{W},\boldsymbol{D})$ might not be the optimiser to (3.33), and thus we can conclude that the minimum of (3.33) is less than or equal to the minimum of (3.32).

We now aim to show the converse of the aforementioned conclusion. Namely that the minimum of (3.32) is less than or equal to the minimum of (3.33), such the problems have equal minima. Now let $(\boldsymbol{W},\boldsymbol{D})$ be in the feasible set of (3.33). Assume $\boldsymbol{D} = \boldsymbol{U}\operatorname{diag}([\lambda_1,\ldots,\lambda_n])\boldsymbol{U}^{\mathsf{T}}$

be a eigenvalue decomposition with the columns of $\boldsymbol{U}$ being the eigenvectors and λ_i the ith eigenvalue. Moreover, assume $\boldsymbol{\Theta} = \boldsymbol{U}^\top \boldsymbol{W}$. Then,

$$\begin{aligned}
\operatorname{tr}\left(\boldsymbol{W}^\top \boldsymbol{D}^+ \boldsymbol{W}\right) &= \operatorname{tr}\left(\boldsymbol{\Theta}^\top \boldsymbol{U}^\top \boldsymbol{U} \operatorname{diag}\left(\left[\lambda_1^+, \ldots, \lambda_n^+\right]\right) \boldsymbol{U}^\top \boldsymbol{U} \boldsymbol{\Theta}\right) \\
&= \operatorname{tr}\left(\operatorname{diag}\left(\left[\lambda_1^+, \ldots, \lambda_n^+\right]\right) \boldsymbol{\Theta}\boldsymbol{\Theta}^\top\right) \\
&= \sum_{i=1}^{m} \lambda_i^+ \|\theta^i\|_2^2.
\end{aligned}$$

Due to the positive semi definiteness of $\boldsymbol{D}$, some of the eigenvalues might equate to zero. If $\lambda_i = 0$ then $\boldsymbol{\theta}^i = \boldsymbol{0}$ to obey the range constraint. To show this we make an argument of contradiction. Let $\boldsymbol{e}_i \in \mathbb{R}^T$ be ith column of a $T \times T$ identity matrix, and suppose $\lambda_i = 0$ and $\boldsymbol{\theta}^i \neq \boldsymbol{0}$. Then $\boldsymbol{W}\boldsymbol{e}_i = \boldsymbol{U}\boldsymbol{\Theta}\boldsymbol{e}_i = \boldsymbol{U}\boldsymbol{\theta}_i$. Then by the range constraint, there exists $\boldsymbol{x} \in \mathbb{R}^m$ such $\boldsymbol{\theta}_i = \boldsymbol{U}^\top \boldsymbol{D}\boldsymbol{x}$. Notice that due to the assumption of $\boldsymbol{\theta}^i \neq \boldsymbol{0}$, the ith entry of $\boldsymbol{\theta}_i$ is also nonzero. However, by the assumption of $\lambda_i = 0$

$$\operatorname{diag}([\lambda_1, \ldots, \lambda_{i-1}, 0, \lambda_{i+1}, \ldots, \lambda_n])\boldsymbol{U}^\top = \begin{bmatrix} \lambda_1 \boldsymbol{u}_1 & \ldots & \lambda_{i-1}\boldsymbol{u}_{i-1} & \boldsymbol{0} & \lambda_{i+1}\boldsymbol{u}_{i+1} & \ldots & \lambda_n \boldsymbol{u}_n \end{bmatrix}^\top.$$

This shows that the ith entry of $\boldsymbol{U}^\top \boldsymbol{D}\boldsymbol{x}$, for any $\boldsymbol{x} \in \mathbb{R}^n$, will always be zero. Thus, we cannot find a vector $\boldsymbol{x} \in \mathbb{R}^m$ such $\boldsymbol{\theta}_i = \boldsymbol{U}^\top \boldsymbol{D}\boldsymbol{x}$, and the range constraint does not hold (which it must, by assumption). Hereby, we have concluded that if $\lambda_i = 0$ then $\boldsymbol{\theta}_i$ cannot be nonzero if we must uphold the range constraint. Thus, we have by contradiction shown that if $\lambda_i = 0 \implies \boldsymbol{\theta}_i = \boldsymbol{0}$.

Consequently,

$$\begin{aligned}
\sum_{i=1}^{n} \lambda_i^+ \|\boldsymbol{\theta}^i\|_2^2 &= \sum_{i:\lambda_i \neq 0} \frac{\|\boldsymbol{\theta}^i\|_2^2}{\lambda_i} \\
&\overset{(a)}{\geq} \left(\sum_{i:\lambda_i \neq 0} \lambda_i^{1/2} \lambda_i^{-1/2} \|\boldsymbol{\theta}^{\boldsymbol{i}}\|_2 \right)^2 \\
&= \left(\sum_{i:\lambda_i \neq 0} \|\boldsymbol{\theta}^{\boldsymbol{i}}\|_2 \right)^2 \\
&= \|\boldsymbol{\Theta}\|_{2,1}^2
\end{aligned}$$

where (a) comes from lemma 3.1. This shows that the minimum of (3.32) will be less than or equal to the minimum of (3.33).

We have shown that for feasible choices, the objective functions of (3.32) and (3.33) are equal. Furthermore, we have shown that the minima of the optimisation problems are upper bounded by each other, thus they must equate, and the proof is complete. ■

The constraint on the trace of $\boldsymbol{D}$ to be upper bounded in theorem 3.1 is necessary, as otherwise $\boldsymbol{D} = \infty$ would be the optimal solution. This is the case, as the pseudoinverse of $\boldsymbol{D}$ is used in (3.33). Likewise, the range constraint is necessary, for the regularisation term $\operatorname{tr}(\boldsymbol{W}^\top \boldsymbol{D}^+ \boldsymbol{W})$ to be strictly positive. Otherwise, when $\boldsymbol{W}$ does not have full rank, $\boldsymbol{D}\boldsymbol{W} = \boldsymbol{0}$ is possible, which indeed would result in the regularisation terms being zero, eluding the point of it. [9]

The whole point of theorem 3.1 was to find an equivalent optimisation problem that was convex. Hence, with the next proposition we show exactly that.

Proposition 3.1.

The following optimisation problem

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{D}} \quad & \sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{w}_t^\intercal \boldsymbol{x}_{n,t},\ y_{n,t}\right) + \lambda \operatorname{tr}(\boldsymbol{W}^\intercal \boldsymbol{D}^{+} \boldsymbol{W}) \\ \text{s.t.} \quad & \operatorname{tr}(\boldsymbol{D}) \leq 1, \quad \boldsymbol{D} \succeq \boldsymbol{0}, \quad \operatorname{range}(\boldsymbol{W}) \subseteq \operatorname{range}(\boldsymbol{D}), \end{aligned} \tag{3.35}$$

is convex.

Proof

The proof is omitted, but can be found in [9]. ■

3.3.2 Alternating Minimisation Algorithm for Multi-Task Feature Learning

We shall in this section present an alternation algorithm for solving (3.33). The term alternating refers to the optimisation over one parameter while the other is fixed and vice versa.

If we constrain $\boldsymbol{D}$ to be positive definite in theorem 3.1, we may drop the range constraint [9, Corollary 1]. Thus, (3.33) becomes

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{D}} \quad & \sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{w}_t^\intercal \boldsymbol{x}_{n,t},\ y_{n,t}\right) + \lambda \operatorname{tr}(\boldsymbol{W}^\intercal \boldsymbol{D}^{-1} \boldsymbol{W}) \\ \text{s.t.} \quad & \operatorname{tr}(\boldsymbol{D}) \leq 1, \quad \boldsymbol{D} \succ \boldsymbol{0}. \end{aligned} \tag{3.36}$$

We shall in this section discuss how the minimum of (3.36) is found. Define,

$$\mathcal{R}_\epsilon(\boldsymbol{W},\boldsymbol{D}) = \sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{w}_t^\intercal \boldsymbol{x}_{n,t},\ y_{n,t}\right) + \lambda \operatorname{tr}(\boldsymbol{D}^{-1}(\boldsymbol{W}\boldsymbol{W}^\intercal + \epsilon \boldsymbol{I})),$$

where $\epsilon \in \mathbb{R}$ and $\epsilon > 0$, and $\boldsymbol{I}$ is the identity matrix of appropriate size. Note that,

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{D}} \quad & \mathcal{R}_\epsilon(\boldsymbol{W},\boldsymbol{D}) \\ \text{s.t.} \quad & \operatorname{tr}(\boldsymbol{D}) \leq 1, \quad \boldsymbol{D} \succ \boldsymbol{0}, \end{aligned} \tag{3.37}$$

is similar to (3.36), as is the point. This section will introduce an alternating minimisation algorithm for multi-task feature learning, that is, solving (3.37). We refer to this algorithm as the MTFL algorithm. Next, we aim to show that: (1) equation (3.37) has a unique minima, (2) the MTFL algorithm converges to that minima and (3) that the solution to (3.37) for $\epsilon \to 0$ converges to the solution of (3.36). Thus, we may use the MTFL algorithm to find a solution to (3.36) and by theorem 3.1 also (3.27), which was the originally desired solution.

The MTFL algorithm generally consist of a supervised step, where $\boldsymbol{D}$ is fixed and an unsupervised step, where $\boldsymbol{W}$ is fixed. Let $\boldsymbol{D}$ be fixed, then (3.37) becomes

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & \sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{w}_t^\intercal \boldsymbol{x}_{n,t},\ y_{n,t}\right) + \lambda \operatorname{tr}\left(\boldsymbol{D}^{-1}\boldsymbol{W}\boldsymbol{W}^\intercal\right) \\ \text{st.} \quad & \boldsymbol{W} \in \mathbb{R}^{m \times T} \end{aligned} \tag{3.38}$$

One crucial observation is that when $\boldsymbol{D}$ is fixed the columns of $\boldsymbol{W}$ are decoupled [9]. The $\boldsymbol{D}$ matrix binds the tasks together, such if one of the weights were to chance it would chance $\boldsymbol{D}$ and in turn change the other weights, and so on. This means that (when $\boldsymbol{D}$ is fixed) the minimisation of the columns of $\boldsymbol{W}$ can be carried out independently. That is, one may consider (3.38) as T independent minimisation problems. This fact shall prove to be a key result in the later works of this project.

Let $\boldsymbol{W}$ be fixed, then (3.37) becomes

$$\begin{aligned} \min_{\boldsymbol{D}} \quad & \operatorname{tr}\left(\boldsymbol{D}^{-1}\left(\boldsymbol{W}\boldsymbol{W}^{\mathsf{T}}+\epsilon \boldsymbol{I}\right)\right) \\ \text{st.} \quad & \operatorname{tr}(\boldsymbol{D}) \leq 1, \quad \boldsymbol{D} \succ \boldsymbol{0}. \end{aligned} \tag{3.39}$$

The next theorem shows that (3.39) has a closed form solution.

Theorem 3.2.

The minimum of

$$\begin{aligned} \min_{\boldsymbol{D}} \quad & \operatorname{tr}\left(\boldsymbol{D}^{-1}\left(\boldsymbol{W}\boldsymbol{W}^{\mathsf{T}}+\epsilon \boldsymbol{I}\right)\right) \\ \text{st.} \quad & \operatorname{tr}(\boldsymbol{D}) \leq 1, \quad \boldsymbol{D} \succ \boldsymbol{0}. \end{aligned} \tag{3.40}$$

is

$$\operatorname{tr}\left(\left(\boldsymbol{W}\boldsymbol{W}^{\mathsf{T}}+\epsilon \boldsymbol{I}\right)^{1/2}\right)^2,$$

and is obtained when

$$\boldsymbol{D}(\boldsymbol{W}) = \frac{\left(\boldsymbol{W}\boldsymbol{W}^{\mathsf{T}}+\epsilon \boldsymbol{I}\right)^{1/2}}{\operatorname{tr}\left(\left(\boldsymbol{W}\boldsymbol{W}^{\mathsf{T}}+\epsilon \boldsymbol{I}\right)^{1/2}\right)}. \tag{3.41}$$

Proof

Assume that $\boldsymbol{D} \succ \boldsymbol{0}$, $\operatorname{tr}(\boldsymbol{D}) \leq 1$, and let $\boldsymbol{C} = \boldsymbol{W}\boldsymbol{W}^{\mathsf{T}}+\epsilon \boldsymbol{I} \succeq \boldsymbol{0}$. First, we show the minimum of (3.40) is indeed $\operatorname{tr}\left(\boldsymbol{C}^{1/2}\right)^2$. Under the assumptions,

$$\begin{aligned} \operatorname{tr}\left(\boldsymbol{D}^{-1}\boldsymbol{C}\right) &\overset{(a)}{\geq} \operatorname{tr}\left(\boldsymbol{D}^{-1}\boldsymbol{C}\right)\operatorname{tr}(\boldsymbol{D}) \\ &\overset{(b)}{=} \operatorname{tr}\left(\left(\boldsymbol{D}^{-1/2}\boldsymbol{C}^{1/2}\right)\left(\boldsymbol{D}^{-1/2}\boldsymbol{C}^{1/2}\right)^{\mathsf{T}}\right)\operatorname{tr}\left(\boldsymbol{D}^{1/2}\left(\boldsymbol{D}^{1/2}\right)^{\mathsf{T}}\right) \\ &\overset{(c)}{=} \|\boldsymbol{D}^{-1/2}\boldsymbol{C}^{1/2}\|_{\mathrm{F}}^2 \|\boldsymbol{D}^{1/2}\|_{\mathrm{F}}^2 \\ &\overset{(d)}{\geq} \operatorname{tr}\left(\boldsymbol{D}^{-1/2}\boldsymbol{C}^{1/2}\boldsymbol{D}^{1/2}\right)^2 \\ &\overset{(e)}{=} \operatorname{tr}\left(\boldsymbol{C}^{1/2}\right)^2. \end{aligned} \tag{3.42}$$

Where (a) comes from the assumption of $\operatorname{tr}(\boldsymbol{D}) \leq 1$, and (b) holds, since the trace operator is invariant under cyclic permutations. Here $\boldsymbol{A}^{1/2}$ denotes the square root of $\boldsymbol{A}$. Note that if $\boldsymbol{A} \succeq \boldsymbol{0}$, then $\boldsymbol{A}^{1/2}$ is symmetric. Equality (c) in (3.42) comes from the relation between the trace operator and the Frobenius norm, and (d) comes from the Cauchy-Schwarz inequality for the Frobenius norm.

For (a) in (3.42) to have equality, $\operatorname{tr}(\boldsymbol{D}) = 1$ is required. For the Cauchy-Schwarz inequality (in (d)) to have equality, $\boldsymbol{D}^{-1/2}\boldsymbol{C}^{1/2} = k\boldsymbol{D}^{1/2} \implies \boldsymbol{D}^{-1}\boldsymbol{C}^{1/2} = k\boldsymbol{I}$ for some $k \in \mathbb{R}$ is required. Next, we show that (3.41) solves (3.40). Indeed,

$$\operatorname{tr}(\boldsymbol{D}) = \operatorname{tr}\left(\frac{\boldsymbol{C}^{1/2}}{\operatorname{tr}\left(\boldsymbol{C}^{1/2}\right)}\right) = \frac{\operatorname{tr}\left(\boldsymbol{C}^{1/2}\right)}{\operatorname{tr}\left(\boldsymbol{C}^{1/2}\right)} = 1,$$

and obviously $\operatorname{tr}(\boldsymbol{C}^{1/2})\boldsymbol{C}^{-1/2}\boldsymbol{C}^{1/2} = \operatorname{tr}(\boldsymbol{C}^{1/2})$ is a constant in regards to $\boldsymbol{D}$. This completes the proof.

■

We have now shown that the second (unsupervised) step of the MTFL algorithm has a closed form solution. This is unfortunately not the case for the first (supervised) step of the algorithm. However, with the next result we show that the first step has a unique solution.

Theorem 3.3.

The minimisation problem

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & \textstyle\sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{w}_t^\intercal \boldsymbol{x}_{n,t},\ y_{n,t}\right) + \lambda \operatorname{tr}\left((\boldsymbol{W}\boldsymbol{W}^\intercal + \epsilon\boldsymbol{I})^{1/2}\right)^2 \\ \text{s.t.} \quad & \boldsymbol{W} \in \mathbb{R}^{m\times T} \end{aligned}$$

has a unique solution.

Proof

By proposition 2.2, a strictly convex function has a unique global minimiser. Thus, it is sufficient to show that

$$\boldsymbol{W} \mapsto \operatorname{tr}\left(\left(\boldsymbol{W}\boldsymbol{W}^\intercal + \epsilon\boldsymbol{I}\right)^{1/2}\right)^2, \tag{3.43}$$

is strictly convex, as we have to assume strict convexity in $\mathcal{L}_t$. Note that a sum of functions is strictly convex if and only if each term in the sum is strictly convex [12]. The map in (3.43) can be rewritten as the following mapping of singular values of $(\boldsymbol{W}\boldsymbol{W}^\intercal + \epsilon\boldsymbol{I})^{1/2}$,

$$\boldsymbol{\sigma} \mapsto \left(\sum_{i=1}^{n}\sqrt{\sigma_i^2 + \epsilon}\right)^2. \tag{3.44}$$

This can be done as $\operatorname{tr}(\boldsymbol{A}) = \sum_i \sigma_i$, where σ_i is the ith singular value of $\boldsymbol{A}$. Lastly, we simply need to argue that (3.44) is strictly convex. This is most easily done if we consider (3.44) in one dimension. Then (3.44) simply becomes $\sigma \mapsto \sigma^2 + \epsilon$, which is obviously strictly convex. This generalises to arbitrary dimension. Thus the proof is complete. ■

Based on the above results, we may now present the MTFL algorithm.

Algorithm 4 Multi-Task Feature Learning algorithm [9]

Input: T training sets $\mathcal{D}_t$ for $t = 1,\dots T$, the regularisation constant λ and a tolerances ϵ and *tol*.

Output: The weight matrix $\boldsymbol{W} \in \mathbb{R}^{m\times T}$, and the feature relation matrix $\boldsymbol{D} \in \mathbb{R}^{m\times m}$.

1: **Initialise:** $\boldsymbol{D} = \frac{1}{m}\boldsymbol{I}$.
2: **while** $\|\boldsymbol{W} - \boldsymbol{W}_{\text{prev}}\| > tol$ **do**
3: **for** each task $t = 1, 2, \dots T$ **do**
4: $\boldsymbol{w}_t = \underset{\boldsymbol{w}}{\operatorname{argmin}} \left\{ \sum_{m=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{w}^\intercal \boldsymbol{x}_{m,t},\ y_{m,t}\right) + \lambda\left(\boldsymbol{w}^\intercal \boldsymbol{D}^{-1}\boldsymbol{w}\right)\right\}$
5: **end for**
6: set $\boldsymbol{D} = \dfrac{\left(\boldsymbol{W}\boldsymbol{W}^\intercal + \epsilon\boldsymbol{I}\right)^{1/2}}{\operatorname{tr}\left(\left(\boldsymbol{W}\boldsymbol{W}^\intercal + \epsilon\boldsymbol{I}\right)^{1/2}\right)}$
7: **end while**

Finally we present two important convergence results for algorithm 4. Define,

$$\boldsymbol{D}_\epsilon(\boldsymbol{W}) = \frac{\left(\boldsymbol{W}\boldsymbol{W}^\intercal + \epsilon\boldsymbol{I}\right)^{1/2}}{\operatorname{tr}\left(\left(\boldsymbol{W}\boldsymbol{W}^\intercal + \epsilon\boldsymbol{I}\right)^{1/2}\right)}.$$

As the algorithm alternates, for each iteration we obtain a pair: $\left(\boldsymbol{W}^{(k)},\ \boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k)}\right)\right)$, where the superscript (k) indicates the kth iteration through the while-loop.

Lemma 3.2.

The sequence $\left\{\mathcal{R}_\epsilon\left(\boldsymbol{W}^{(k)},\ \boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k)}\right)\right)\right\}_{k=1}^{\infty}$ is nonincreasing.

Proof

At any given iteration (k)

$$\begin{aligned}\mathcal{R}_\epsilon\left(\boldsymbol{W}^{(k)}, \boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k)}\right)\right) &\geq \min_{\boldsymbol{V}} \mathcal{R}_\epsilon\left(\boldsymbol{V}, \boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k)}\right)\right) \\ &\geq \mathcal{R}_\epsilon\left(\boldsymbol{W}^{(k+1)}, \boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k+1)}\right)\right),\end{aligned}$$

thus showing the sequence to be nonincreasing. ■

Theorem 3.4.

For every $\epsilon > 0$ the sequence $\left\{\left(\boldsymbol{W}^{(k)},\ \boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k)}\right)\right)\right\}_{k=1}^{\infty}$ converges to the minimiser of

$$\begin{aligned}\min_{\boldsymbol{W},\boldsymbol{D}} \quad & \mathcal{R}_\epsilon(\boldsymbol{W}, \boldsymbol{D}_\epsilon(\boldsymbol{W})) = \textstyle\sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{w}_t^\intercal \boldsymbol{x}_{n,t},\ y_{n,t}\right) + \lambda \operatorname{tr}\left(\boldsymbol{D}_\epsilon(\boldsymbol{W})^{-1}\left(\boldsymbol{W}\boldsymbol{W}^\intercal + \epsilon\boldsymbol{I}\right)\right) \\ \text{s.t.} \quad & \operatorname{tr}(\boldsymbol{D}) \leq 1, \quad \boldsymbol{D} \succ \boldsymbol{0}.\end{aligned}$$

Proof

By lemma 3.2, and the fact that $\mathcal{L}(\cdot,\cdot)$ is bounded, $\mathcal{R}_\epsilon\left(\boldsymbol{W}^{(k)},\ \boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k)}\right)\right)$ is bounded. We denote $\lim_{k\to\infty} \mathcal{R}_\epsilon\left(\boldsymbol{W}^{(k)},\ \boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k)}\right)\right) = \mathcal{R}_\epsilon^\star$. Since $\mathcal{R}_\epsilon\left(\boldsymbol{W}^{(k)},\ \boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k)}\right)\right)$ is bounded, we deduce that $\left\{\operatorname{tr}\left(\left(\boldsymbol{W}^{(k)}\boldsymbol{W}^{(k)\intercal} + \epsilon\boldsymbol{I}\right)^{1/2}\right)\right\}_{K=1}^{\infty}$ and in turn $\left\{\boldsymbol{W}^{(k)}\right\}_{k=1}^{\infty}$ also are bounded. Then by Balzano-Weierstrass' theorem [19, sec. 3.4.2], there exists a convergent subsequence of $\left\{\boldsymbol{W}^{(k)}\right\}_{k=1}^{\infty}$, namely

$\{\boldsymbol{W}^{(k_l)}\}_{l=1}^{\infty}$. We denote $\lim_{l\to\infty}\boldsymbol{W}^{(k_l)}=\boldsymbol{W}^{\star}$. Let $g_\epsilon(\boldsymbol{W})=\min_{\boldsymbol{V}}\mathcal{R}_\epsilon\left(\boldsymbol{V},\boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k)}\right)\right)$. Recall from the proof of lemma 3.2 that

$$\mathcal{R}_\epsilon\left(\boldsymbol{W}^{(k)},\boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k)}\right)\right)\geq g_\epsilon(\boldsymbol{W})\geq\mathcal{R}_\epsilon\left(\boldsymbol{W}^{(k+1)},\boldsymbol{D}_\epsilon\left(\boldsymbol{W}^{(k+1)}\right)\right).$$

By the sandwich theorem [19, sec. 3.1.3], $g_\epsilon(\boldsymbol{W})$ will as well converge to $\mathcal{R}_\epsilon^{\star}$. $\mathcal{R}_\epsilon(\boldsymbol{W},\boldsymbol{D}_\epsilon(\boldsymbol{W}))$ is continuous, due to it being convex and defined on an open interval. This is the case, since any function that is convex on an open interval is also continuous on said open interval. Likewise, $g_\epsilon(\boldsymbol{W})$ can be shown to be continuous [9]. As a consequence of this continuity, we conclude that $g_\epsilon(\boldsymbol{W}^{\star})=\mathcal{R}_\epsilon(\boldsymbol{W}\star,\boldsymbol{D}_\epsilon(\boldsymbol{W}^{\star}))$, which implies that $\boldsymbol{W}^{\star}$ is a minimiser of $g_\epsilon(\cdot)$. Moreover, $\boldsymbol{D}_\epsilon(\boldsymbol{W}^{\star})$ is a minimiser of $\mathcal{R}_\epsilon(\boldsymbol{W}^{\star},\cdot)$. Then, as $\operatorname{tr}(\boldsymbol{D}^{-1}(\boldsymbol{W}\boldsymbol{W}^{\intercal}+\epsilon\boldsymbol{I}))$ is smooth, any directional derivatives of $\mathcal{R}_\epsilon(\boldsymbol{W},\boldsymbol{D})$ is the sum of its directional derivatives with respect to $\boldsymbol{W}$ and $\boldsymbol{D}$ [9]. Therefore, $(\boldsymbol{W}^{\star},\boldsymbol{D}_\epsilon(\boldsymbol{W}^{\star}))$ is a minimiser of $\mathcal{R}_\epsilon(\cdot,\cdot)$. Thus, we have shown that any convergent subsequence of $\{\boldsymbol{W}^{(k_l)}\}_{l=1}^{\infty}$ converges to a minimiser of $\mathcal{R}_\epsilon(\cdot,\cdot)$. Since the original sequence $\{\boldsymbol{W}^{(k)}\}_{k=1}^{\infty}$ is monotone and bounded, it is convergent [19, sec. 3.4.1], thus completing the proof.

■

Theorem 3.5.

Let the sequence $\{\epsilon_l>0\}_{l=1}^{\infty}$ converge towards zero, and let $\left(\boldsymbol{W}_l,\boldsymbol{D}_{\epsilon_l}(\boldsymbol{W}_l)\right)$ be a minimiser of

$$\begin{aligned}\min_{\boldsymbol{W},\boldsymbol{D}}\quad&\mathcal{R}_{\epsilon_l}(\boldsymbol{W},\boldsymbol{D})=\textstyle\sum_{t=1}^{T}\sum_{n=1}^{N_t}\mathcal{L}_t\left(\boldsymbol{w}_t^{\intercal}\boldsymbol{x}_{n,t},\ y_{n,t}\right)+\lambda\operatorname{tr}\left(\boldsymbol{D}^{-1}\left(\boldsymbol{W}\boldsymbol{W}^{\intercal}+\epsilon_l\boldsymbol{I}\right)\right)\\ \text{s.t.}\quad&\operatorname{tr}(\boldsymbol{D})\leq 1,\quad\boldsymbol{D}\succ\boldsymbol{0},\end{aligned}$$

for every $l\in\mathbb{N}$. Then any limiting point of $(\boldsymbol{W}_l,\boldsymbol{D}_{\epsilon_l}(\boldsymbol{W}_l))$ is a minimiser of

$$\begin{aligned}\min_{\boldsymbol{W},\boldsymbol{D}}\quad&\textstyle\sum_{t=1}^{T}\sum_{n=1}^{N_t}\mathcal{L}_t\left(\boldsymbol{w}_t^{\intercal}\boldsymbol{x}_{n,t},\ y_{n,t}\right)+\lambda\operatorname{tr}(\boldsymbol{W}^{\intercal}\boldsymbol{D}^{-1}\boldsymbol{W})\\ \text{s.t.}\quad&\operatorname{tr}(\boldsymbol{D})\leq 1,\quad\boldsymbol{D}\succ\boldsymbol{0}.\end{aligned}$$

Proof

Let the sequence $\{\epsilon_l>0\}_{l=1}^{\infty}$ converge towards zero. Let $\left\{\left(\boldsymbol{W}_{l_k},\boldsymbol{D}_{\epsilon_{l_k}}\left(\boldsymbol{W}_{l_k}\right)\right)\right\}_{k=1}^{\infty}$ be a sequence of limiting points of the sequence $\left\{\left(\boldsymbol{W}_l,\boldsymbol{D}_{\epsilon_l}(\boldsymbol{W}_l)\right)\right\}_{l=1}^{\infty}$. Finally, let $\lim_{k\to\infty}\left(\boldsymbol{W}_{l_k},\boldsymbol{D}_{\epsilon_{l_k}}\left(\boldsymbol{W}_{l_k}\right)\right)=\left(\boldsymbol{W}^{\star},\boldsymbol{D}^{\star}\right)$. It is clear that $\min_{\boldsymbol{W},\boldsymbol{D}}\mathcal{R}_\epsilon(\boldsymbol{W},\boldsymbol{D})$ is decreasing, as a function of ϵ. Fix both $\boldsymbol{W}$ and $\boldsymbol{D}$, and move ϵ outside the trace operator, to see that this statement is true. Next, let $\lim_{\epsilon\to 0}\min_{\boldsymbol{W},\boldsymbol{D}}\mathcal{R}_\epsilon(\boldsymbol{W},\boldsymbol{D})=\mathcal{R}^{\star}$. This means that, if we use $\left\{\left(\boldsymbol{W}_{l_k},\boldsymbol{D}_{\epsilon_{l_k}}\left(\boldsymbol{W}_{l_k}\right)\right)\right\}_{k=1}^{\infty}$ as input to $\mathcal{R}_\epsilon(\boldsymbol{W},\boldsymbol{D})$, we must have

$$\lim_{k\to\infty}\mathcal{R}_{\epsilon_{l_k}}\left(\boldsymbol{W}_{l_k},\boldsymbol{D}_{\epsilon_{l_k}}\right)=\mathcal{R}^{\star}.$$

By proposition 3.1, $\mathcal{R}(\boldsymbol{W},\boldsymbol{D})$ is convex, and so is $\mathcal{R}_\epsilon(\boldsymbol{W},\boldsymbol{D}_\epsilon)$. $\mathcal{R}_\epsilon(\boldsymbol{W},\boldsymbol{D})$, as a function of ϵ, is defined on the open interval $]0,\infty[$, and thus it is continuous. Note that $\mathcal{R}_\epsilon(\boldsymbol{W},\boldsymbol{D})$ is also continuous in $\boldsymbol{W}$, as $\boldsymbol{W}\in\mathbb{R}^{m\times T}$, which is an open interval. By this continuity in both ϵ and $\boldsymbol{W}$, we conclude that $\mathcal{R}_\epsilon(\boldsymbol{W}^{\star},\boldsymbol{D}^{\star})=\mathcal{R}^{\star}$ for $\epsilon=0$, completing the proof. ■

We have in this section discussed the concept of multi-task learning, and under which circumstances one should apply a multi-task learning scheme. We have in (3.27) seen an intuitive formulation of a multi-task learning minimisation problem. We have discussed some undesirable properties of (3.27), when it comes to the method of solving this. However, we have presented a result in theorem 3.1 which states that we may rewrite (3.27), such the solving should prove easier. The MTFL algorithm has been introduced, and we have shown a multitude of results regarding the convergence of the algorithm. In the next chapter we aim to combine the section on federated learning and this section on multi-task learning to describe federated multi-task learning.

4 Proposed Solution: Federated Multi-Task Learning

We shall in this section make a stride to combine the two previously described concepts: Federated Learning and Multi-Task Learning. Thus, finding a way of solving multi-task learning problems in settings where the data is split into disjoint subset, which are physically distributed, further restricted by the fact that we are neither able nor allowed to communicate the data it self.

We have previously discussed the learning of a linear feature space, and proposed the inclusion of a non-linear transformation of the data in the model. With this inclusion, we shall in this section investigate the effects of the data being distributed in a multi-task setting. The aim of this section is to build a federated multi-task learning algorithm with low communication cost and computational complexity, that has the ability to perform binary classification.

There already exist algorithms that can solve problem of this kind. One of these is [11], which both utilises a kernel matrix, granting a great measure for task relation within the data. However, this requires that all the data be transmitted to the central server before beginning the training, to determine said kernel matrix, followed by the distribution of the kernel matrix, to all of the nodes. The solution from [3], likewise, utilises a kernel matrix. This solution even includes the distribution of data during the training between the nodes. Another solution is the MOCHA algorithm [4]. It does not require any transmission of data, and solely communicates local weights. However, it does so to not only the central server, but also to all of the other nodes. Common for all of the mentioned solutions is the similar primal problem, and the use of the stochastic dual coordinate ascent (abbr. SDCA) algorithm. This algorithm is commonly used throughout the literature of federated multi-task learning to solve quadratic approximations of the dual problem. This means, that all of the mentioned literature utilises this SDCA algorithm on the nodes.

The shall in this section aim to create an algorithm that takes ideas from [3, 4, 11, 13], to create an algorithm that does not require sharing of data, that has both limited communication requirements and limited computational complexity.

First we state the optimisation problem, which we aim to solve.

$$\begin{aligned} \min_{\boldsymbol{W},\boldsymbol{D}} \quad & \sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_t\left(\boldsymbol{w}_t^\top \phi\left(\boldsymbol{x}_{n,t}\right),\ y_{n,t}\right) + \lambda \operatorname{tr}\left(\boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{W}^\top\right) \\ \text{s.t.} \quad & \operatorname{tr}(\boldsymbol{D}) \leq 1, \quad \boldsymbol{D} \succ \boldsymbol{0}. \end{aligned} \tag{4.1}$$

It is important to note that in the regularisation term, we have changed the transpose. This means that the $\boldsymbol{D}$-matrix will now learn the relations between the tasks, rather than the relation between the features. As we are interested in introducing different types of machinery into the learning, the features these different type of machinery might not be the same, thus it makes more sense the compare tasks rather than features. With that said, many of the results from the previous chapter still applies. Namely, theorem 3.2, which means that we may still update $\boldsymbol{D}$ using a closed form solution:

$$\boldsymbol{D} = \frac{\left(\boldsymbol{W}^\top\boldsymbol{W} + \epsilon\boldsymbol{I}\right)^{1/2}}{\operatorname{tr}\left(\left(\boldsymbol{W}^\top\boldsymbol{W} + \epsilon\boldsymbol{I}\right)^{1/2}\right)} \tag{4.2}$$

This is the case as we may simply define $\boldsymbol{C} = \boldsymbol{W}^\top\boldsymbol{W} + \epsilon\boldsymbol{I}$ in the proof of theorem 3.2. This decision is supported by [11] and [4], as they do the same. Also note the introduction of the feature mapping: $\phi(\cdot)$, which maps each data point into the feature space.

We aim to build an algorithm that, like in the previous chapter, alternates between optimising over $\boldsymbol{W}$ with fixed $\boldsymbol{D}$, and vice versa. As mentioned above we can use the closed form solution for $\boldsymbol{D}$ from (4.2). This means that we only need to determine a way of updating the weights $\boldsymbol{w}_i$ for $i = 1, \ldots, T$.

4.1 Updating the weights

Recall that for fixed $\boldsymbol{D}$ in (4.1) the constraints are removed. According to [10], (4.1) is convex, and therefore the derivative of the Lagrangian will be zero at the optimal point. However, the concept of constructing the Lagrangian, only makes sense, if the original optimisation problem has constraints. We, therefore, rewrite (4.1) as follows

$$\begin{aligned} \min_{\boldsymbol{W}} \quad & \sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_{n,t}\left(z_{n,t}\right) + \lambda \operatorname{tr}\left(\boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{W}^\top\right) \\ \text{s.t.} \quad & \boldsymbol{w}_t^\top \phi\left(\boldsymbol{x}_{n,t}\right) - z_{n,t} = 0 \ \ \forall t, n. \end{aligned} \tag{4.3}$$

Note the n index on the loss function, which allows us the omit the label. It can be seen that (4.1) for fixed $\boldsymbol{D}$, and (4.3) are identical. Now we are able to form a meaningful Lagrangian, by introducing the Lagrangian multipliers $\boldsymbol{\alpha}$:

$$\mathrm{L}(\boldsymbol{W}, \boldsymbol{z}, \boldsymbol{\alpha}) = \sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_{n,t}\left(z_{n,t}\right) + \lambda \operatorname{tr}\left(\boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{W}^\top\right) + \sum_{t=1}^{T}\sum_{n=1}^{N_t} \alpha_{n,t}\left(\boldsymbol{w}_t^\top \phi\left(\boldsymbol{x}_{n,t}\right) - z_{n,t}\right) \tag{4.4}$$

To clarify: The structure of the Lagrangian multipliers is as follows $\boldsymbol{\alpha} = \left[\boldsymbol{\alpha}_1^\top, \boldsymbol{\alpha}_2^\top, \ldots, \boldsymbol{\alpha}_T^\top\right]^\top$, meaning that $\alpha_{n,t}$ is the nth entry of $\boldsymbol{\alpha}_t$. Likewise, for $\boldsymbol{z}$.

Recall, that the data is distributed, thus we cannot use (4.4) directly, as it contains all of the data. Therefore, we consider the influence that the tth column of $\boldsymbol{W}$ has on the Lagrangian from (4.4). Due to convexity: $\frac{\partial \mathrm{L}(\boldsymbol{w}_t^\star)}{\partial \boldsymbol{w}_t} = 0$ for the optimal weights $\boldsymbol{w}_t^\star$, thus

$$\begin{aligned}
\frac{\partial \mathrm{L}}{\partial \boldsymbol{w}_t} &= \frac{\partial}{\partial \boldsymbol{w}_t} \lambda \operatorname{tr}\left(\boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{W}^\intercal\right) + \sum_{n=1}^{n_t} \alpha_{n,t}\phi(\boldsymbol{x}_{n,t}) \\
&= \frac{\partial}{\partial \boldsymbol{w}_t} \lambda \sum_{k=1}^{T}\sum_{l=1}^{T} \boldsymbol{D}_{l,k}^{-1}\boldsymbol{w}_l^\intercal \boldsymbol{w}_k + \sum_{n=1}^{n_t} \alpha_{n,t}\phi(\boldsymbol{x}_{n,t}) \\
&= 2\lambda \sum_{l=1}^{T} \boldsymbol{D}_{l,t}^{-1}\boldsymbol{w}_l - \sum_{n=1}^{n_t} \alpha_{n,t}\phi(\boldsymbol{x}_{n,t}) \\
&= 2\lambda \sum_{l\neq t}^{T} \boldsymbol{D}_{l,t}^{-1}\boldsymbol{w}_l + 2\lambda \boldsymbol{D}_{t,t}^{-1}\boldsymbol{w}_t + \sum_{n=1}^{n_t} \alpha_{n,t}\phi(\boldsymbol{x}_{n,t}) \\
&= 0 \\
\implies \quad \boldsymbol{w}_t^\star &= \frac{-\boldsymbol{b}_t - 2\lambda \sum_{l\neq t}^{T} \boldsymbol{D}_{l,t}^{-1}\boldsymbol{w}_l}{2\lambda \boldsymbol{D}_{t,t}^{-1}},
\end{aligned} \tag{4.5}$$

where $\boldsymbol{b}_t = \sum_{n=1}^{n_t} \alpha_{n,t}\phi(\boldsymbol{x}_{n,t})$. Note how only $\boldsymbol{b}_t$ depends on the data, and that this dependency is only on the locally accessible data. This means that each node is able to locally compute $\boldsymbol{b}_t$. However, to do so, $\boldsymbol{\alpha}_t$ is necessary. To determine $\boldsymbol{\alpha}_t$ we turn to the dual function. Recall that the dual function is given as the infimum of the Lagrangian, hence

$$\begin{aligned}
g(\boldsymbol{\alpha}) &= \inf_{\boldsymbol{W},\boldsymbol{z}} \mathrm{L}(\boldsymbol{W},\boldsymbol{z},\boldsymbol{\alpha}) \\
&= \inf_{\boldsymbol{W},\boldsymbol{z}} \left(\sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_{n,t}\left(z_{n,t}\right) + \lambda \operatorname{tr}\left(\boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{W}^\intercal\right) + \sum_{t=1}^{T}\sum_{n=1}^{N_t} + \alpha_{n,t}\left(\boldsymbol{w}_t^\intercal \phi\left(\boldsymbol{x}_{n,t}\right) - z_{n,t}\right) \right) \\
&\overset{(a)}{=} \sum_{t=1}^{T}\sum_{n=1}^{N_t} \inf_{z}\left(\mathcal{L}_{n,t}\left(z_{n,t}\right) - \alpha_{n,t} z_{n,t}\right) + \inf_{\boldsymbol{W}} \left(\lambda \operatorname{tr}\left(\boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{W}^\intercal\right) + \sum_{t=1}^{T}\sum_{n=1}^{N_t} \alpha_{n,t}\boldsymbol{w}_t^\intercal \phi\left(\boldsymbol{x}_{n,t}\right) \right) \\
&\overset{(b)}{=} \sum_{t=1}^{T}\sum_{n=1}^{N_t} -\mathcal{L}_{n,t}^*(\alpha_{n,t}) + \inf_{\boldsymbol{W}} \left(\lambda \operatorname{tr}\left(\boldsymbol{W}\boldsymbol{D}^{-1}\boldsymbol{W}^\intercal\right) + \sum_{t=1}^{T}\sum_{n=1}^{N_t} \alpha_{n,t}\boldsymbol{w}_t^\intercal \phi\left(\boldsymbol{x}_{n,t}\right) \right)
\end{aligned} \tag{4.6}$$

where (a) comes from splitting up the Lagrangian into terms dependent on $\boldsymbol{z}$ and $\boldsymbol{W}$ and (b) comes from the definition of the convex conjugate (also referred to as the Fenchel conjugate).

It can be seen that maximisation of (4.6) is equivalent to the minimisation problem

$$\min_{\boldsymbol{\alpha}} \quad g_G(\boldsymbol{\alpha}) = \textstyle\sum_{t=1}^{T}\sum_{n=1}^{N_t} \mathcal{L}_{n,t}^*(\alpha_{n,t}) - \lambda \operatorname{tr}\left(\boldsymbol{W}^\star \boldsymbol{D}^{-1}\boldsymbol{W}^{\star\intercal}\right) - \sum_{t=1}^{T}\sum_{n=1}^{N_t} \alpha_{n,t}\boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right). \tag{4.7}$$

Note that the maximum of (4.6) and the minimum of (4.7) is not guaranteed to have the same value, but will have the same argument, and recall that we are interested in the Lagrangian multipliers that minimise (4.7), and not the minimum value of $g(\boldsymbol{\alpha})$. We refer to $g_G(\boldsymbol{\alpha})$ as the dual function of the global problem or simply the global dual function. Using the same argument as before, we investigate the impact that $\boldsymbol{\alpha}_t$ has on (4.7), as seen below.

$$\min_{\boldsymbol{\alpha}_t} \quad \textstyle\sum_{n=1}^{N_t} \mathcal{L}_{n,t}^*(\alpha_{n,t}) - \sum_{n=1}^{N_t} \alpha_{n,t}\boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right). \tag{4.8}$$

We can do this as we are interested in the argument and not the value. Notice, that (4.8) may be rewritten as follows

$$\min_{\boldsymbol{\alpha}_t} \quad \sum_{n=1}^{N_t} \mathcal{L}^*_{n,t}(\alpha_{n,t}) - \boldsymbol{w}_t^{\star\intercal} \boldsymbol{X}_t \boldsymbol{\alpha}_t, \tag{4.9}$$

where $\boldsymbol{X}_t = \left[\phi\left(\boldsymbol{x}_{1,t}\right), \ldots, \phi\left(\boldsymbol{x}_{n,t}\right)\right]$. As proclaimed in chapter 2 the dual function is always concave regardless of whether the original problem is convex. This means that (4.9) is convex as it is the minimisation of a negated concave problem. Next, we can find the optimal Lagrangian multipliers by equating the derivative of (4.9) to zero. However, before determining the derivative of (4.9) we need to determine an expression for the convex conjugated of the loss function.

Recall the definition of the convex conjugated:

$$\mathcal{L}^*_{n,t}(\alpha_{n,t}) = \sup_{z \in \mathbb{R}} \left\{ \alpha_{n,t} z - \mathcal{L}_{n,t}(z) \right\}. \tag{4.10}$$

To move forward, a loss function must be chosen. All of the tasks in this project will be binary classification problems, thus choices such as cross-entropy loss(from logistic regression) or Hinge-loss are fitting.

Simply by examining (4.10), we realise that - dependent on the choice of loss function - there might not be a choice of $\alpha_{n,t}$ such (4.10) is finite. To ensure that (4.9) gives a meaningful result, we next examine (4.10) for different choices of loss function.

We start by examining the hinge-loss function. Thus, (4.10) becomes,

$$\mathcal{L}^*_{n,t}(\alpha_{n,t}) = \sup_{z \in \mathbb{R}} \left\{ \alpha_{n,t} z - \max\left(0, 1 - y_{n,t} z\right) \right\}. \tag{4.11}$$

As there are two labels, it is natural to examine (4.11) for one label at a time.

In the following we assume $y_{n,t} = 1$, even when referring to (4.11). It is clear from (4.11), that for $\alpha_{n,t} > 0$, (4.11) will be ∞, as we can let $z \to \infty$. The same goes for $\alpha_{n,t} < -1$, as we can let $z \to -\infty$. Thus, we must limit our search to $\alpha_{n,t} \in [-1, 0]$. Equation (4.11) is essentially a piecewise function given as follows:

$$\mathcal{L}^*_{n,t}(\alpha_{n,t}) = \begin{cases} \alpha_{n,t} z - (1 - z) & z < 1 \\ \alpha_{n,t} z & z \geq 1, \end{cases} \tag{4.12}$$

for $\alpha_{n,t} \in [-1, 0]$. Note that (4.12) is strictly increasing for $z < 1$ and decreasing for $z \geq 1$. Thus, the maximum will be found where they meet, at $z = 1$. At $z = 1$, (4.12) takes the value $\alpha_{n,t}$. This means, that we now have an expression for (4.11) as a function of $\alpha_{n,t}$ for $y_{n,t} = 1$.

We can make the same analysis for $y_{n,t} = -1$. Using the same argument, we realise that (4.11) is finite for $\alpha_{n,t} \in [0, 1]$, and likewise for $y_{n,t} = 1$, it takes its maximum value where the functions meet, at $z = -1$. Thus, (4.11) take the value of $-\alpha_{n,t}$, for $\alpha_{n,t} \in [0, 1]$.

Next we examine what these findings mean for the original problem, we investigated, (4.9). Note that we may consider (4.9) as N_t separate optimisation problems, formulated as follows

$$\min_{\alpha_{n,t}} \quad \mathcal{L}^*_{n,t}(\alpha_{n,t}) - \alpha_{n,t} \boldsymbol{w}_t^{\star\intercal} \phi\left(\boldsymbol{x}_{n,t}\right). \tag{4.13}$$

This is the case as we are minimising over $\boldsymbol{\alpha}_t$ in (4.9), and it can be noticed that each entry in $\boldsymbol{\alpha}_t$ - that is $\alpha_{n,t}$ for $n = 1,\dots,N_t$ - is only present in one of the terms in the sum, likewise is the same entry only multiplied by its associated data point ($\boldsymbol{x}_{n,t}$).

Assuming that $y_{n,t} = 1$, then (4.13) becomes,

$$\begin{aligned} \min_{\alpha_{n,t}} \quad & \alpha_{n,t}\left(1 - \boldsymbol{w}_t^{\star\mathsf{T}}\phi\left(\boldsymbol{x}_{n,t}\right)\right) \\ \text{s.t.} \quad & \alpha_{n,t} \in [-1,0]. \end{aligned} \tag{4.14}$$

We may rewrite (4.14) as follows:

$$\begin{aligned} \min_{\alpha_{n,t}} \quad & \alpha_{n,t}\left(-1 + \boldsymbol{w}_t^{\star\mathsf{T}}\phi\left(\boldsymbol{x}_{n,t}\right)\right) \\ \text{s.t.} \quad & \alpha_{n,t} \in [0,1]. \end{aligned} \tag{4.15}$$

Note that the equality can be seen by examining the minima of (4.14) and (4.15) for values of e.g. $\boldsymbol{w}_t^{\star\mathsf{T}}\phi\left(\boldsymbol{x}_{n,t}\right) = 0.5, 1, 2$.

It is clear to see that in this case $\alpha_{n,t}$ will always take on the value 1 or 0, dependent on the sign of $-1 + \boldsymbol{w}_t^{\star\mathsf{T}}\phi\left(\boldsymbol{x}_{n,t}\right)$. If we try to interpret this result: For a given data point with a ground truth label of 1, the product $\boldsymbol{w}_t^{\star\mathsf{T}}\phi\left(\boldsymbol{x}_{n,t}\right)$ should be larger than one, yielding a correct classification. Now if that is the case, then $\alpha_{n,t} = 0$ is chosen in (4.15). Recall from (4.5) that the weight $\boldsymbol{w}_t$ is - amongst others - determine by $\boldsymbol{b}_t = \boldsymbol{X}_t\boldsymbol{\alpha}_t$. This means that if the data point is correctly classified using the current version of the weights, it will not have any influence on the updated version of the weights. This is the case as the entry in corresponding to said data point will be zero.

To clarify: Any correctly classified data point, using the current version of the weights, will not influence the updated version. We update the weights based solely upon the wrongly classified data points. However, this means that any data point will either have full influence or non at all. Perhaps it could be beneficial to make more subtle changes in weights.

It should be mentioned that the cross-entropy suffers from a similar problem. For any data point with label $y = 1$ the the associated α must be chosen within the interval $[-1,0]$ and likewise $[0,1]$ for $y = -1$. Otherwise, the convex conjugate will converge towards infinity.

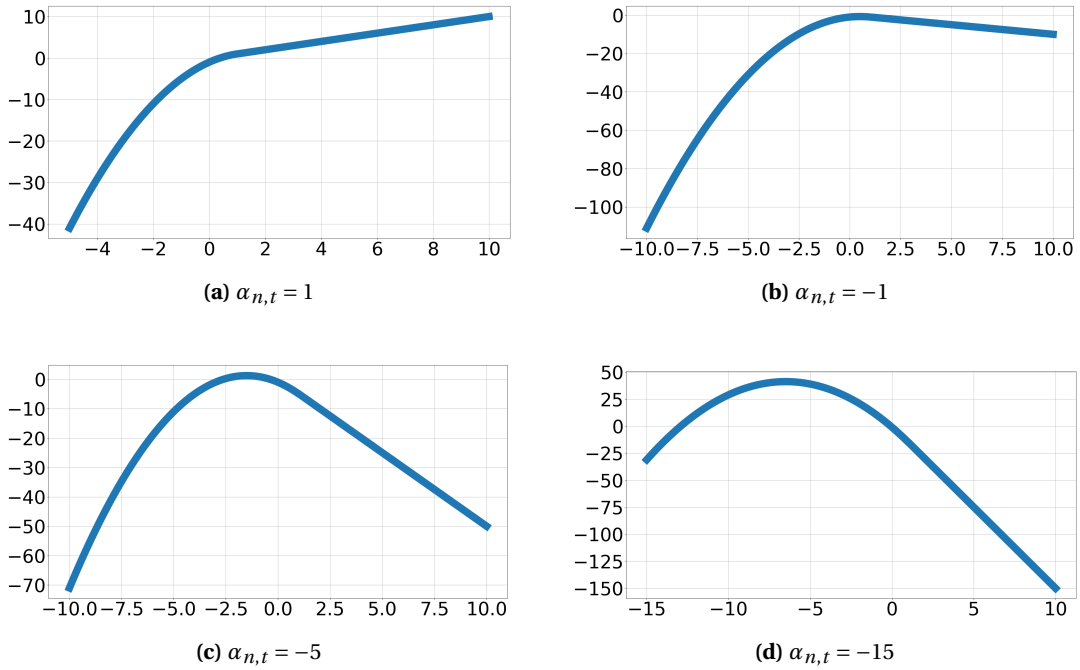
To investigate the opportunity of giving data point a smoother influence, without moving away from the idea of hinge-loss, we examine (4.10) for squared hinge-loss, which is simply hinge-loss squared. Thus, (4.10) becomes,

$$\mathcal{L}^*_{n,t}(\alpha_{n,t}) = \sup_{z\in\mathbb{R}}\left\{\alpha_{n,t}z - \max\left(0, 1 - y_{n,t}z\right)^2\right\}. \tag{4.16}$$

We next do the same analysis as for hinge-loss. Like previous, we examine (4.16) one label at a time. Assuming $y_{n,t} = 1$, we notice the following equality:

$$\alpha_{n,t}z - \max(0, 1-z)^2 = \begin{cases} \alpha_{n,t}z & z \geq 1 \\ -z^2 + (\alpha_{n,t}+2)z - 1 & z < 1. \end{cases} \tag{4.17}$$

Notice that for $z < 1$ the function is concave, meaning it has a unique maximum. Likewise notice two obvious observation that for $\alpha_{n,t} > 0$, (4.16) will be ∞ as we can let $z \to \infty$. Therefore, we must restrict $\alpha_{n,t} \leq 0$. Figure figure 4.1 depicts (4.17) for four different choices of $\alpha_{n,t}$, namely $1, -1, -5$ and -15. Notice how for $\alpha_{n,t} = 1$ the function is non-decreasing, just as we argued earlier.



(a) $\alpha_{n,t} = 1$ **(b)** $\alpha_{n,t} = -1$

(c) $\alpha_{n,t} = -5$ **(d)** $\alpha_{n,t} = -15$

Figure 4.1. Equation (4.17) for various choices of $\alpha_{n,t}$

Like previously, with $\alpha_{n,t} \leq 0$, the maximum value is found on the bound, i.e. at $z = 1$ or on the left hand side i.e. $z < 1$. We can quickly verify that (4.16) is zero for $\alpha_{n,t} = 0$: Obviously $0z = 0, \forall z \geq 1$, and secondly, $-z^2 + 2z - 1 > 0$ has no solutions, as its extremum is 0 with $z = 1$. This means that for $\alpha_{n,t} < 0$ the maximum of (4.17) is to be found when $z < 1$.

For a second-order polynomial on the form $az^2 + bz + c$, the extremum is given as: $-(b^2 - 4ac)/4a$. This means, that for $\alpha_{n,t} < 0$ and $y_{n,t} = 1$, (4.16) is given as,

$$\begin{aligned} \frac{-\left((\alpha_{n,t}+2)^2 - (4\cdot(-1)\cdot(-1))\right)}{-4} &= \frac{-\left(\alpha_{n,t}^2 + 4 + 4\alpha_{n,t} - 4\right)}{-4} \\ &= \frac{\alpha_{n,t}^2 + 4\alpha_{n,t}}{4} \\ &= \frac{\alpha_{n,t}^2}{4} + \alpha_{n,t}. \end{aligned}$$

W now have an expression for (4.16) when $y_{n,t} = 1$.

The same can be done for when $y_{n,t} = -1$. In that case we notice the following equality:

$$\alpha_{n,t} z - \max(0, 1+z)^2 = \begin{cases} \alpha_{n,t} z & z \leq -1 \\ -z^2 + (\alpha_{n,t} - 2)z - 1 & z > -1. \end{cases} \tag{4.18}$$

We notice that for $\alpha_{n,t} < 0$ (4.16) is ∞. Thus, we limit $\alpha_{n,t} \geq 0$, as we recall the goal is to minimise (4.16) w.r.t $\alpha_{n,t}$. We again argue that (4.16) is zero for $\alpha_{n,t} = 0$, using the same argument as previously. Next we examine the extremum of (4.18) for $\alpha_{n,t} > 0$:

$$\begin{aligned}\frac{-\left((\alpha_{n,t}-2)^2-4\right)}{-4} &= \frac{\alpha_{n,t}^2+4-4\alpha_{n,t}-4}{4} \\ &= \frac{\alpha_{n,t}^2}{4} - \alpha_{n,t}.\end{aligned}$$

With the two expressions for (4.16) - one for each of the labels - we are now ready to examine (4.13).

Assume $y_{n,t} = 1$. Then (4.13) becomes

$$\begin{aligned}\min_{\alpha_{n,t}} \quad & \frac{\alpha_{n,t}^2}{4} + \alpha_{n,t}\left(1 - \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right)\right) \\ \text{s.t.} \quad & \alpha_{n,t} \leq 0.\end{aligned} \tag{4.19}$$

Now since the objective function of (4.19) is convex, the minimum can be found by equating the derivative to zero. Thus,

$$\begin{aligned}\frac{1}{2}\alpha_{n,t} + \left(1 - \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right)\right) &= 0 \\ \implies \alpha_{n,t} &= -2\left(1 - \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right)\right).\end{aligned}$$

From this it can clearly be seen that in a solution of (4.19), different from $\alpha_{n,t} = 0$, we must have

$$\begin{aligned}1 - \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right) &> 0 \\ \implies 1 &> \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right),\end{aligned}$$

which means a misclassification. Thus, we again see that the data point which are correctly classified, will have no influence on the updated weights. However, we have constructed a setting where data points which are misclassified have influence based upon the severity of the misclassification. Let examine an example. Let the label of $\phi\left(\boldsymbol{x}_{n,t}\right)$ be 1, but $\boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right) = -5$. This, in turn, implies that $\alpha_{n,t} = -12$, making it very important in the next weight update. We shall not be conducting the same analysis for the cases when $y_{n,t} = -1$, as it will be very similar. Next, we give an overview of how $\alpha_{n,t}$ is chosen:

$$\alpha_{n,t} = \begin{cases} -2\left(1 - \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right)\right) & 1 > \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right) \text{ and } y_{n,t} = 1 \\ 0 & 1 \leq \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right) \text{ and } y_{n,t} = 1 \\ 2\left(1 + \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right)\right) & \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right) > -1 \text{ and } y_{n,t} = -1 \\ 0 & \boldsymbol{w}_t^{\star\intercal}\phi\left(\boldsymbol{x}_{n,t}\right) \leq -1 \text{ and } y_{n,t} = -1. \end{cases} \tag{4.20}$$

To summarise: We have determined a way of updating the weights in (4.5), these updates requires knowledge of the Lagrangian multipliers, which we have determined values for based on the label and the prediction of a given data point in (4.20). Finally, we are updating the relation matrix using (4.2).

Next we are able to paint a picture of how the federated multi-task learning should function. Locally, the Lagrangian multipliers are being determined, such in turn we can determine the $\boldsymbol{b}_t$ vector at each node. These $\boldsymbol{b}_t$ vectors are then transmitted from each node to the central server, where the weights, $\boldsymbol{w}_t$ are update, followed by the update of $\boldsymbol{D}$. The way, that we have structured this, we are able to run the scheme utilising asynchronous updates. This means, that when the tth node finished updating the Lagrangian multipliers, it transmits its $\boldsymbol{b}_t$ vector to the central server, which in turn updates the weights $\boldsymbol{w}_t$ and the $\boldsymbol{D}$ matrix. The central server then transmits the updated weights back to the tth node.

Now, we have not assumed anything about the computing power at each node, nor the number of data point at each node. This means that some nodes may determine $\boldsymbol{b}$ faster than others. This is the incentive to allow asynchronous updates in the first place. However, this might also result in cases where no other node has requested an update at the central server between two consecutive requests from a particular node. Thus, the second update request is somewhat irrelevant.

First, we argue that two consecutive update request, without any other nodes updates in between is not pointless. This is not the case, since the weight of this particular node has been updated. Thus, the predictions of the local data points will most likely be different, and thus in turn the Lagrangian multipliers will be different. However, said node will probably experience diminishing returns if such consecutive updates happen multiple times in a row. Imagine a particular node, having requested 10 updates in row, without any other node having requested updates in between. Most likely the 10th update will not yield as much of a change in the prediction of the local data points as the first did.

This problem encourages the ability of a node making multiple local updates, before requesting an global update at the central server. This is made available by making approximate updates to the wights on potentially each of the nodes. This approximate update simply consists of letting vector $\boldsymbol{b}_t$ be the updated version of $\boldsymbol{w}_t$. That is, instead of requesting an update an the central server, we let the node make the approximate update simply using $\boldsymbol{b}_t$ instead. Note, that is only something that should be employed if a particular node has extremely high computation time compare the rest of the nodes. As the number of nodes increase this will most likely not become necessary.

We are now ready to present the algorithm.

Algorithm 5 Federated Multi-Task Task Learning Algorithm

Input: T training sets $\mathcal{D}_t$ for $t = 1, \dots T$, the regularisation constant λ and a tolerances ϵ and *tol*.

Output: The weight matrix $\boldsymbol{W} \in \mathbb{R}^{m \times T}$, and the relation matrix $\boldsymbol{D} \in \mathbb{R}^{T \times T}$.

Initialise: $\boldsymbol{D} = \frac{1}{T}\boldsymbol{I}$.
while $\|\boldsymbol{W} - \boldsymbol{W}_{\text{prev}}\| > tol$ **do**
  for each task $t = 1, 2, \dots T$ in parallel **do**
    $\boldsymbol{\alpha}_t = \text{NODE\_UPDATE}(\boldsymbol{w}_t, \boldsymbol{X}_t, \boldsymbol{y}_t)$
    $\boldsymbol{b}_t = \boldsymbol{X}_t \boldsymbol{\alpha}_t$
    Transmit $\boldsymbol{b}_t$ to the central server
  end for

  The central server update for node t:
  Receive $\boldsymbol{b}_t$ from the tth node
  $\boldsymbol{w}_t = \dfrac{-\boldsymbol{b}_t - 2\lambda \sum_{l \neq t}^{T} \boldsymbol{D}_{l,t}^{-1} \boldsymbol{w}_l}{2\lambda \boldsymbol{D}_{t,t}^{-1}}$
  set $\boldsymbol{D} = \dfrac{(\boldsymbol{W}^\intercal \boldsymbol{W} + \epsilon \boldsymbol{I})^{1/2}}{\text{tr}\left((\boldsymbol{W}^\intercal \boldsymbol{W} + \epsilon \boldsymbol{I})^{1/2}\right)}$
  Transmit $\boldsymbol{w}_t$ to the tth node.
end while

Below we present the NODE_UPDATE algorithm.

Algorithm 6 NODE_UPDATE

Input: The current version of the weights $\boldsymbol{w}_t \in \mathbb{R}^m$, the data $\boldsymbol{X}_t \in \mathbb{R}^{m \times N_t}$ and the associated labels $\boldsymbol{y}_t \in \mathbb{R}^{N_t}$.

Output: The optimal Lagrangian multipliers $\boldsymbol{\alpha}_t \in \mathbb{R}^{N_t}$.

Initialise $\boldsymbol{\alpha}_t = \boldsymbol{0}$.
for $i = 1, 2, \dots, N_t$ **do**
  if $y_i = 1$ **then**
    temp$= -2 + 2\boldsymbol{w}_t^\intercal \boldsymbol{x}_i$
    if temp< 0 **then**
      $\alpha_i =$ temp
    end if
  else if $y_i = -1$ **then**
    temp$= 2 + 2\boldsymbol{w}_t^\intercal \boldsymbol{x}_i$
    if temp > 0 **then**
      $\alpha_i =$ temp
    end if
  end if
end for

It is atypical to be this specific with algorithms, but it shall in the next subsection be made perfectly clear, why exactly we have been this specific.

4.2 Computational Complexity

In this subsection, we shall investigate the computational complexity of the proposed solution. In most cases, the accuracy of a solution is the most important parameter, however, not the only one. Another parameter is computational complexity. This gives an expression of how hard a processing unit needs to work to obtain the aforementioned accuracy. Computational complexity in algorithms is measured in floating point operations (abbr. FLOPs), which is simply the total number of operations performed during the algorithm. If a particular algorithm is required to add two scalars, we say that the particular algorithm has a FLOPs count of one.

We have in chapter 1 mentioned that we are not concerned with the computations on the central server. And if we were to compare algorithm 5 to a local traditional solution, then the computations on the central server are disregarded. Thus, we solely count the number of FLOPs that occurs on the node.

We shall do this by going through steps 4 and 5 in algorithm 5. The goal is to find an expression for the number of FLOPs performed as a function of the number of data points. We shall then compare this number to the traditional method of the support vector machines.

We start be examining step 4 in algorithm 5, which is the same as the entire algorithm 6. We shall examine this, by evaluating a single run through of the for-loop in step 2 of algorithm 6. This means that there will be a single FLOP to check whether $y_i = 1$. Step 8 is formulated as an else if, but could for all intents and purposes have been an else, thus, we do not attribute a FLOP to this. Assume that $y_i = 1$, we reach step 4, which holds the majority of the FLOPs of the algorithm. An inner product between two vectors of same size - in this case size m - can be perform using $2m-1$ FLOPs. Then there is another FLOP used to multiply the result of the inner product by 2, and finally another FLOP to subtract 2. Then step 5 has a single FLOP in the if-statement. This totals to $2m+3$ FLOPs if $y_i = 1$. However, it will be the exact same number of FLOPs if $y_i = -1$, and since only one of these cases will be the case, we conclude that the algorithm needs $2m+3$ FLOPs per run through the for-loop. The for-loop is performed a total of N_t times as it runs once per data point. Thus, algorithm 6 need a total of $N_t(2m+3)$ FLOPs to complete.

Finally we need to include step 5 from algorithm 5 to have our final answer. A matrix-vector product will need $2N_t - 1$ FLOPs per number of rows in $\boldsymbol{X}_t$, which has m rows, thus a total of: $m(2N_t - 1)$ FLOPs to comple step 5 in algorithm 5. This brings the total FLOPs need to perform a single round of processing on the node to a total of: $N_t(4m+3) - m$. From this number we can see that the number of FLOPs needed scales linearly with the number data points.

We choose to compare the proposed solution with an SVM solution. We have chosen a python implementation for this project, and one library that has a SVM solution is SKlearn. It can be found in the documentation for said SVM solution that the implementation scales at leased quadratic with the number of data points [15]. Another research - [20] - has investigated the

number of FLOPs needed to perform sequential minimal optimisation algorithms, which are algorithms that - amongst others - are used in SVM solutions. The research conducted reports a count of FLOPs needed of upwards of 10^9, for a data set of size 1000. This number is enormous compared to the number of FLOPs required for algorithm 5 per node. Let $N_t = 1000$ and let $m = 20$, then we expect algorithm 5 to need $N_t(4m+3) - m = 829.890$ FLOPs, which is approximately $0.8 \cdot 10^5$. Even if we assume the quadratic scaling with the number of data points, then the number of FLOPs needed to perform SVM is 10^6, which is still about 12 times the amount of FLOPs. This sounds like a lot, but algorithm 5 performs the 829.890 FLOPs at every round. Thus, we conclude that the number of communication rounds must be smaller than 12 for a set up with 20 features. The number of communication round are obviously dependant on both the number of data points and the number of features.

When comparing the number of FLOPs needed in algorithm 5, to the local SVM, we realise that the number of FLOPs needed in the local SVM will too depend on the number of features. However, as the number of features will often be much smaller than the number of data points, it is usually disregarded.

4.3 Additional nodes

An important part of such a solution we are after is the ability of said solution to adapt to the number of tasks. That is, we need a solution that is robust in terms of adding new tasks. When we say robust, we refer to the ability to easily include a new task in the learning without requiring to start the entire learning process over.

Imagine that algorithm 5 is running and assume that it currently has T nodes in the scheme. At some point, before the algorithm has converged, a new node is suppose to be included in the learning, thus bringing the total number of nodes to $T+1$. Once the new node is fully installed, it begins steps 4 and 5 of algorithm 5, with an initialisation of $\boldsymbol{w}_{T+1} = \mathbf{0}$. This way it can calculate the $\boldsymbol{b}_{T+1}$ vector. With the first transmission of the $\boldsymbol{b}_{T+1}$ vector to the central server, it includes the fact that is it a new node, such the central server can act accordingly. The only thing that the central server is changing, with the inclusion of the new node, is a change of dimension of the relation matrix. The dimension of $\boldsymbol{D}$ is expanded to $\boldsymbol{D} \in \mathbb{R}^{(T+1)\times(T+1)}$, and the new entries in $\boldsymbol{D}$ are given as follows:

$$\boldsymbol{D}_{T+1} = \begin{bmatrix} \frac{T}{T+1}\boldsymbol{D} & \mathbf{0} \\ \mathbf{0}^\intercal & \frac{1}{T+1} \end{bmatrix}.$$

This way we assume no relations between the new task and any of the existing tasks. This way of handling the addition of new nodes is suggested by [13].

4.4 Federated solution

The final point we need to argue, is whether or not a federated solution makes sense. Recall the arguments in chapter 1 about limited processing power etc. However, what if the number of communication rounds needed for algorithm 5 to converge exceeds the number of transmissions needed to move the entire data set to the central server? In that case, it would be preferable to simply transmit all the data and conduct training on the central server.

In order to, perform such an analysis, without going into too much detail, we define a single data point as 1 unit of communication. Since $\boldsymbol{w}_t$ and $\boldsymbol{b}_t$ for any t have the same dimension as a single data point, we consider a single transmission of $\boldsymbol{w}_t$ or $\boldsymbol{b}_t$ as one unit of communication as well. This means that every time a node needs to communicate with the central server, 2 units of communication is used, as the node transmits $\boldsymbol{b}_t$ to the central server, and receives $\boldsymbol{w}_t$ in return. If algorithm 5 needs more than half the number of data point on a single node to converge, it is not communications efficient. Algorithm 5 would have to use substantially less rounds of communication than the number of data points, since a solution where the data is transmitted will most likely lead to a more accurate solution, due to the ability of running more complex machine learning schemes.

5 Simulations

We shall in this chapter investigate the capabilities of the proposed solution compared to those of SVM solutions. We investigate whether it would be advantageous to transmit all of the data sets to the central server, by comparing the proposed solution to a "global" SVM. That is, a single SVM, that has access the all of the data. We investigate whether a purely local solution would be better. This way, no communication between the nodes and the central server would be necessary. We do this, by comparing the proposed solution to local SVMs, one for each of the node, having access to the local data only. Lastly we compare with the MOCHA algorithm [4], which is a state of the arc federated multi-task solution, that is in many ways similar to the proposed solution.

We begin by examining the proposed solution on a synthetic data. Later, we shall investigate in what capacity the proposed solution can perform predictive maintenance on the MIMII data, which will be described later.

5.1 Task Relation Learning on Synthetic Data

We shall in this section investigate, via a simulation study, the capabilities of algorithm 5. In this section we solely focus on synthetic data, that is, data which is generated for this simulation study. Unless otherwise stated, the simulation studies below were conducted using the convergence tolerance $tol = 10^{-4}$ from algorithm 5. We shall run multiple experiments, all designed to test different aspects of algorithm 5.

Firstly, we explain how the synthetic data is generated. The data is regenerated before every experiment, and every experiment is run multiple times. Firstly, a total number of tasks is chosen e.g. $T = 16$, then a number of "original" tasks is chosen e.g. $T_{og} = 4$. Is shall be clear why these are referred to as original tasks. For each original task a Gaussian vector is drawn as follows:

$$\tilde{\boldsymbol{w}}_o \sim \mathcal{N}(\boldsymbol{0}, \boldsymbol{I}), \quad \text{for } o = 1, \ldots, T_{og}.$$

It is worth noting that the elements of $\tilde{\boldsymbol{w}}_o \in \mathbb{R}^m$ are uncorrelated. The remaining $T - T_{og}$ tasks are randomly selected from the set $\{\pm\tilde{\boldsymbol{w}}_o | o \in \mathbb{N} : o \leq T_{og}\}$. The negative sign is included to simulate negative relations, and is applied with a probability of 0.5. We refer to these T vectors as the "creator" weights, as their sole purpose is to determine the labels of the data points.

The data points, $\boldsymbol{x}_{n,t}$ are each drawn uniformly from the interval $[-1,1]$ with a dimension of m. There are independence both between data points, but also between entries in a single data point. The number of data points is (unless otherwise stated) drawn uniformly from the interval $[900,1100]$, and rounded to nearest integer. This way, the amount of data points will most likely be different across task, while still maintaining a substantial amount of data points. This is done accommodate the assumption of different frequency of data generation acrois the nodes. Lastly, the labels are generated as follows:

$$y_{n,t} = \text{sign}\left(\boldsymbol{w}_t^\intercal \boldsymbol{x}_{n,t}\right), \tag{5.1}$$

where $\text{sign}(\cdot)$ is the sign function that returns -1 or 1 depending on the sign of the input. With the data creation complete, we are now ready to describe the first experiment.

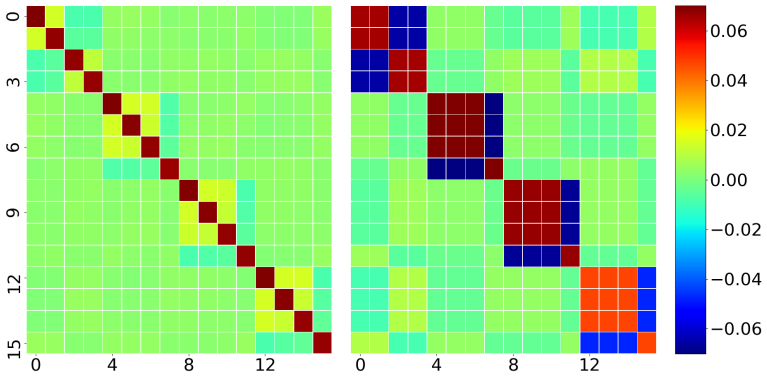
5.1.1 Learning Tasks Relations

In this subsection we investigate whether algorithm 5 can actually learn the relations between the individual tasks. That is, can it find randomly selected positive or negative relations that all of the non-original creator weights have with one of the original creator weights. This experiment is conducted, by simply running the algorithm on the synthetic data, generated above, and then comparing the learned $\boldsymbol{D}$ matrix, line 10 in algorithm 5, with the estimated trace normalised covariance matrix of the creator weights. We trace normalise the covariance matrix, such we may compare the values of the matrix entries. For this experiment we chose $T = 16$ and $T_{og} = 4$, and placed the 4 original creator weights at positions: 1, 5, 9 and 13. Throughout this experiment we are using $\lambda = 0.6$, from algorithm 5, as was found to yield good results. We found this value through numerous tests. This parameter will be application specific, and should undergo thorough testing before settling on a specific value. Further more, we let the number of features be $m = 20$. Figure 5.1 depicts heat maps of the learned $\boldsymbol{D}$ matrix (left) and the estimated trace normalised covariance matrix of the creator weights (right), for two different synthetic data sets. If the algorithm does, what it claims to do, we expect the two heat maps to be similar.

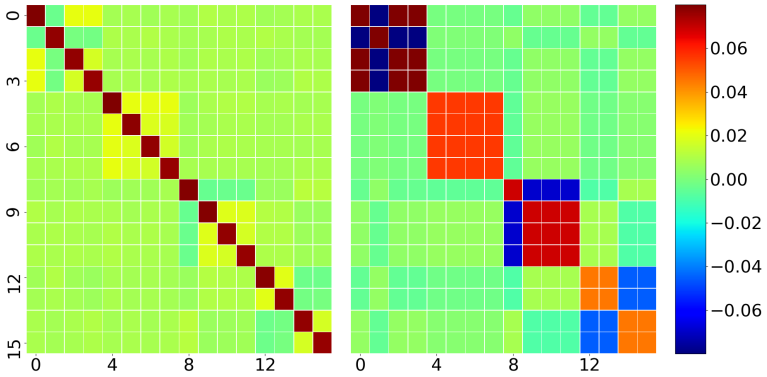
From Figure 5.1 it is clearly seen that algorithm 5 can detect the relations between the tasks that are actually related (referring to the four squares). However, it is also clearly seen that the algorithm does not capture the scale of these relations. This could be a worrying sign, if tasks are only slightly related, these might be interpreted by algorithm 5 as being unrelated, and thus, some of the advantages of the solution are lost.

5.1.2 Accuracy of the Proposed Solution

In this subsection, we examine the obtainable accuracy of the classification, based the weights learned from algorithm 5. The data is generated in the exact same way as in the previous subsection, except in this subsection we use $m = 2$. We simply run the algorithm and examine



(a) Heat map comparison for comparison between the learned relations matrix (left) and the estimated covariance matrix of the creator weights (right)



(b) Heat map comparison for comparison between the learned relations matrix (left) and the estimated covariance matrix of the creator weights (right)

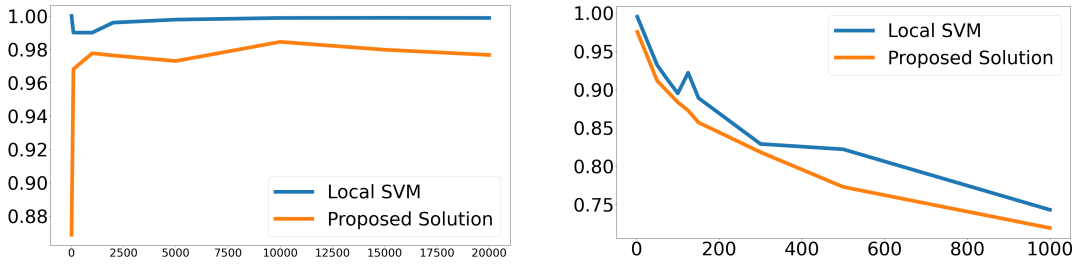
Figure 5.1. Depicts heat maps for comparison between the learned relations matrix (left) and the estimated covariance matrix of the creator weights (right), for two different synthetic data sets.

the results. While the SVM implementation used in this project, from [15], comes with scoring extension, the proposed solution does not. Thus, we predict the labels of data points as done in (5.1). The accuracy of the four methods used can be found and compared in table 5.1. From table 5.1, it can be seen that in general, the proposed solution performs worse than the local SVM solution, however, much better than the global SVM solution. One thing to notice is the indifference in the accuracy between the two types of kernels used in the SVMs. The most likely reason for this is the way the data and labels are generated. It is generated in a way such that - in theory - a SVM solution with a linear kernel should be able to perfectly classify the data. Thus, the added complexity from the Gaussian kernel does not bring any additional information.

Next we examine the behaviour in term of accuracy of the proposed solution against the local SVM alternative, as the number of data points and features increases. We do, however, examine these separately. That is, while one is increasing, the other is fixed. The results are shown in figure 5.2.

Table 5.1. Shows the accuracy of the 4 methods compared.

Task id	Accuracy of proposed solution	Local SVM accuracy(Gaussian)	Local SVM accuracy(linear)	Global SVM accuracy
1	0.966	985	0.998	
2	0.98	0.998	0.991	
3	0.996	0.994	0.997	
4	0.989	1	0.993	
5	0.988	0.996	0.998	
6	0.989	0.997	0.997	
7	0.994	0.998	0.999	
8	0.987	0.997	0.993	
9	0.943	0.998	0.994	
10	0.942	0.996	0.995	
11	0.953	1	0.997	
12	0.958	0.995	0.99	
13	0.953	0.994	0.998	
14	0.958	0.995	0.997	
15	0.985	0.997	0.999	
16	0.955	0.999	0.992	
Average:	0.971	0.996	0.996	0.604



(a) Accuracy vs. number of data points per node, N_t

(b) Accuracy vs. number of features, m

Figure 5.2. Depicts the accuracy obtained from the proposed solution and a local SVM solution.

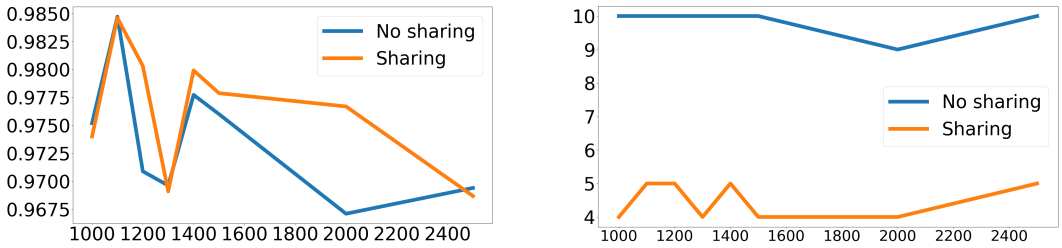
From figure 5.2 it can be seen that algorithm 5 consistently performs almost as good at the local SVM alternative. Though the accuracy of algorithm 5 decreases when the number of features increase, so does the accuracy of the local SVM. From the way that the data is generated, the number of features (m) is sort of a metric for the difficulty of the problem. As the number of features grow, more parameters have to be correctly tuned, thus, making the tasks more difficult. Therefore, it is expected to see a decrease in accuracy with an increase in features.

5.1.3 Enhanced Learning From Learning Task Relations

We shall in this subsection investigate, whether it is worth it, to include the sharing of weights and letting them influence each other. In this experiment, we examine the differences in ac-

curacy and number of local rounds needed to converge, between sharing and not sharing the weights with the central server. This would be the same as fixing $\boldsymbol{D} = \boldsymbol{I}$, this way when the weights are updated, no weight will have any influence on any other weight. Thus, for the "no sharing" version, we simply only allow local computations to take place, and in the "sharing" version, we let the algorithm run as described in algorithm 5.

In this experiment we use the synthetic data previously described, unless otherwise stated. First, we examine the accuracy vs. the number of data points at each node. For this we keep the number of data points constant across the tasks, thus all task have the same number of data points. We found a need to increase the value of tol to 0.1, as the convergence of the "no sharing" version is very slow. Figure 5.3 depicts both the accuracy and the number of rounds needed to converge as a function of the number of data points per task.



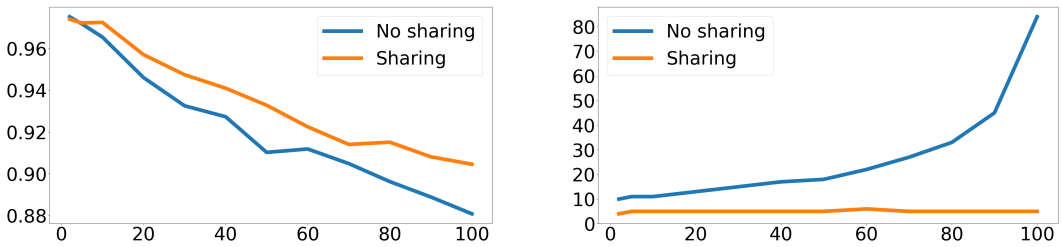
(a) Accuracy vs. N_t

(b) # of local rounds needed to converge vs. N_t

Figure 5.3. Depicts the accuracy of and the number of rounds needed for algorithm 5 to converge.

From Figure 5.3 it can be seen that sharing the weights with the central server, has very little to no improvement in the accuracy, and in some cases it even worsens the accuracy. However, as can be seen from figure 5.3b, it does drastically reduce on the number of rounds needed to converge.

Next, we conducted a similar experiment, however, this time, letting the accuracy and the number of rounds needed to converge be a function of the number of features, m. Figure 5.4 depicts these results.



(a) Accuracy vs. m

(b) # of local rounds needed to converge vs. m

Figure 5.4. Depicts a comparison of the accuracy and the number of rounds to convergence, for algorithm 5 with and with out sharing of the weights.

From Figure 5.4 it can - once again - be seen that the gain in accuracy is small (if any), however, we do see an immense improvement in the convergence speed. This means the algorithm is greatly enhanced by the sharing of the weights in terms of convergence.

As a final note; experiments were conducted in order to determine the number of rounds needed to converge, for the version of algorithm 5 without sharing, with a number of features of 125. However, after 40.000 iterations, the algorithm had yet to terminate. Thus, the experiment was cut.

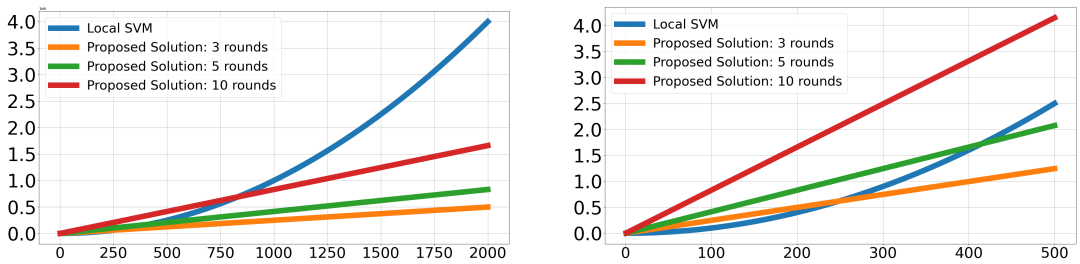
5.1.4 Communication Rounds

Recall our discussion in chapter 4 about the number of communication rounds. We discussed two aspects of concern, being: The number of FLOPs and whether it is worth it to construct a federated solution. We begin by investigating the number of rounds needed to converge. The number of rounds needed to converge for algorithm 5, along with the obtained accuracy as a function of the number of data points is seen in table 5.2. It should be noted that $tol = 0.1$ in this experiment.

Table 5.2. Comparison of the number of data points per task and the rounds needed to converge, together with the obtained accuracy.

Number of data points:	10^4	$1.5 \cdot 10^4$	$2 \cdot 10^4$	$4 \cdot 10^4$	$7 \cdot 10^4$	10^5	$5 \cdot 10^5$	10^6
Rounds to converge	5	4	5	4	5	4	4	4
Accuracy	0.976	0.972	0.975	0.977	0.982	0.992	0.983	0.976

From table 5.2 it is seen that the number of rounds needed to converge seems more or less unaffected by the number of data points per task. Therefore, we move onward in this investigation with a selection of numbers of rounds, as follows: 3, 5, 10. We note that the accuracy in table 5.2 is included solely to inform that the low number of rounds is not affecting the accuracy.



(a) Number of rounds [10^6] vs. the number of data points N_t

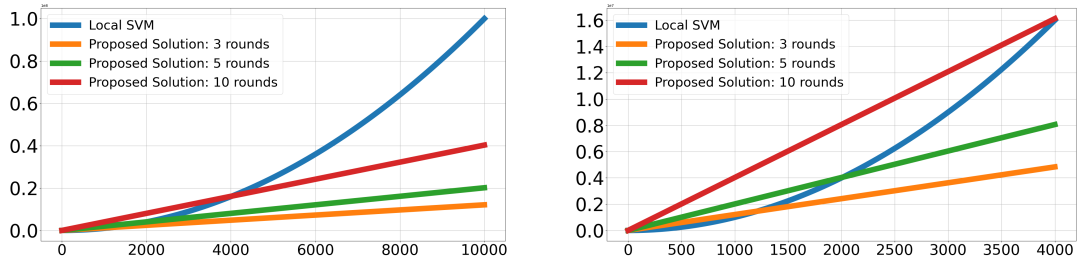
(b) Number of rounds [10^5] vs. the number of data points N_t

Figure 5.5. Depicts the number of rounds vs. the number of data points N_t, for fixed $m = 20$.

Recall, that algorithm 5 needs, $N_t(4m+3) - m$ FLOPs per round of communication per node. However, as previously argued, we simply compare one node to one local SVM solution. In the

figures 5.5 and 5.6, we see the results of this experiment. Note that we compare to the minimum number of FLOPs needed to complete a SVM algorithm, which is the square of the number of data points (blue curve on the figures 5.5 and 5.6).

During these two experiments we fix the number of features to 20 and 100, to test the differences. The goal of this experiment is to determine whether the loss in accuracy is made of for in the reduced number of FLOPs needed to complete algorithm 5. As can be seen from the figures 5.5 and 5.6, the answer to that question is dependent on the number of data points. As the SVM scales at least quadratic and algorithm 5 scales linearly, the proposed solution will always be favoured for large amounts of data points.



(a) Number of rounds [10^8] vs. the number of data points N_t **(b)** Number of rounds [10^7] vs. the number of data points N_t

Figure 5.6. Depicts the number of rounds vs. the number of data points N_t, for fixed $m = 100$.

Next we discuss whether it is worth it to construct a federated solution at all. Luckily - as can be seen from table 5.2 - algorithm 5 converges very quickly in terms of number of communication rounds. The number of rounds never exceeds 5, even as the number of data points increase. We have in all of the experiments - using synthetic data - used the same precision (32 bits) for data points and weights. This means that algorithm 5 uses at most 10 units of communication, which is well under the amount that would otherwise be needed to transmit all the data points. We are examining this on a per node basis, since it has to communication efficient for all the nodes. Otherwise, if a node has far fewer data points than the rest, it would probably be more efficient to simply transmit the data points of said node and take the node out of the federated learning.

5.2 Predictive Maintenance on the MIMII Data Set

We shall in this section investigate the capabilities of algorithm 5, in terms of ability to perform predictive maintenance, on the MIMII data set [21].

5.2.1 Data Description and Preprocessing

The data set used in this project consists of 4 different types of machines: Valves, pumps, fans, and slide rails. The data consists of a large amount of 10 seconds long acoustic recordings, collected using a TAMAGO-03 microphone, which is a circular microphone array consisting of 8 microphones. The microphone array was in each case placed 50 cm from the machines, except for the valves where the array was placed 10 cm away.

Within each type of machinery (valves, pumps etc.), there are recordings of 7 individual machines of the same type, however, they may come from different manufactures. For each individual machine, there are recordings under both normal and abnormal operations. These conditions are described in table 5.3. The exact number of recording for each individual machine across machine type, can be found in table 5.4. For a more detailed description of the data, we refer to [21], from which the data is taken. Unfortunately, only some of the recordings are available, those being: 00, 02, 04 and 06 individuals for all types for machinery. The authors promise the full data set in the future. However, each individual recording is available in three distinct signal-to-noise ratios (abbr. SNR), those being: 6 dB, 0 dB and -6 dB. That is, a random background noise is added to each "clean" recording. The background noise is recorded in several real-life production factories, and is added to each individual recording at random. Thus, two recordings from the same individual machine, might have two different background noises added to them.

Table 5.3. Shows the functionalities and the abnormal conditions of each of the machinery types in the data set.

Machine type	Functionality	Abnormal conditions
Valve	Opening / closing, repeating at different time intervals	multiple types of contamination
Pump	Suction from / discharge to, a water pool	Leakage, contamination, clogging
Fan	Continuous flow of gas / air	Unbalanced, voltage changes, clogging
Slide rail	Sliding back and forth, repeating	Rail damage, insufficient grease

5.2.1.1 Preprocessing

Thought the goal was to conduct experiments on all of the four types of machinery accessible in the MIMMI data set, we have - due to time restrictions - had to limit the amount of experiments included. Since we are collaborating with Grundfos A/S, it is natural to focus on the water pump data.

It was found - through visual inspection - that the abnormal data in general had a more volatile frequency content in the lower frequencies across time, in the time-frequency spectrum. This

Table 5.4. Shows the number of recording/data points associated with each of the individual machines, both under normal and abnormal conditions.

Machine type / model ID		# of normal segments	# of abnormal segments
Valves	00	991	119
	01	869	120
	02	708	120
	03	963	120
	04	1000	120
	05	999	400
	06	992	120
Pump	00	1006	143
	01	1003	116
	02	1005	111
	03	706	113
	04	702	100
	05	1008	248
	06	1036	102
Fan	00	1011	407
	01	1034	407
	02	1016	359
	03	1012	358
	04	1033	348
	05	1109	349
	06	1015	361
Slide rail	00	1068	356
	01	1068	178
	02	1068	267
	03	1068	178
	04	534	178
	05	534	178
	06	534	89
Total		26092	6065

observation led to the choice of feature space, which is the variance of the tiles across time, in the time-frequency spectrum. Furthermore, we ensure zero-mean and unit-variance across the features within a task.

5.2.2 Proposed Solution vs. Local SVM

We shall in this subsection conduct experiments comparing the proposed solution to a local SVM. As seen in the in section 5.1, the global SVM solution performs poorly, and we have, therefore, chosen not to continue to consider it.

As mentioned, each recording of the pumps (in the MIMII data set) comes in three SNR levels. To test whether the proposed solution can detect the relations between the task, we consider each level of SNR as a seperate task. This way we experiment with 9 task, three for each of the

individuals: 00, 02 and 06. That is, task 1 is the task of classifying the data from pump individual 00 with a SNR of -6, task 2 is the same, but for SNR 0. Task 3 is again the same, but for SNR 6. Tasks 4-6 is likewise formulated for pump individual 02 and tasks 7-9 as for 06.

For each of the nine tasks, we construct two data sets: A training set and a testing set. The test set always contains exactly 200 normal data points and 10 abnormal, regardless of task. The training set will contain the remaining data points. This means that the training set will have different size across tasks.

During this experiment, we used $\lambda = 0.03$, $tol = 0.1$, $\epsilon = 0.01$, and the algorithm converged in 11 rounds.

Due to task setup, we expect a strong positive relation between tasks 1-3, 4-6 and 7-9. Additionally, as all the task are of the same type of pump, we expect a positive relation between all the tasks. Examining figure 5.7 we see exactly what we expect.

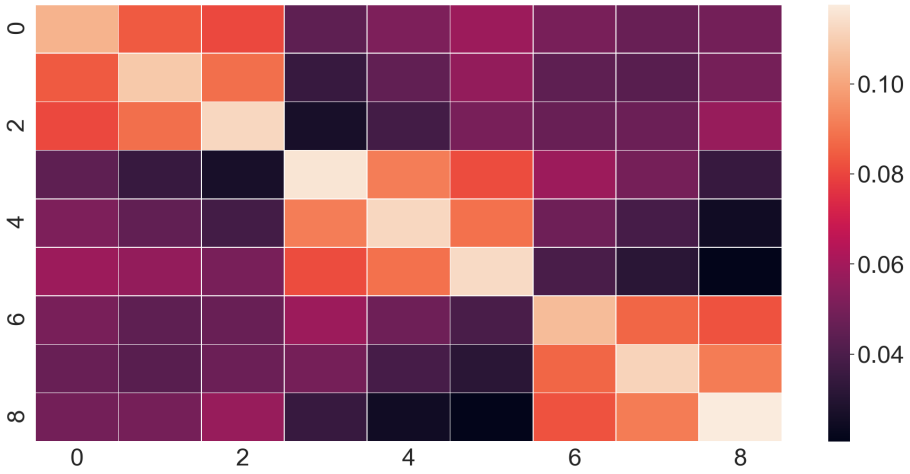


Figure 5.7. A heat map of the relation matrix, $\boldsymbol{D}$.

It is clear to see - from figure 5.7 - that the algorithm recognises that some of the tasks are more related than others. Simultaneously, we see that all the tasks have a positive relation with all of the other tasks, as expected.

The problem, that we are trying to solve is a classification problem, but in reality it is a predictive maintenance problem. This means, that we a particular interested in detecting the abnormal data points. Therefore, it is to a degree acceptable to classify normal data points as abnormal data points, if we make sure that all of the abnormal data points are correctly classified. A tool to observe to what degree we need to allow misclassification of normal data points, to obtain a certain accuracy of the abnormal data points, is the receiver operating characteristic (abbr. ROC) curve.

The ROC curve is a visualisation of the confusion matrix as the decision bound changes. In the

case of the proposed solution the decision is the cutoff for which data points are being classified. The confusion matrix is an expression of how the relation between the ground truth label and the predicted label. In the binary case, that is the detection of abnormal pumps, we define:

- True Positive: An abnormal Pump classified as an abnormal Pump
- True Negative: A normal Pump classified as a normal Pump
- False Positive: A normal Pump classified as an abnormal Pump
- False Negative: An abnormal Pump classified as a normal Pump

In this case the confusion matrix is as follow:

$$\begin{bmatrix} \text{True Negative} & \text{False Positive} \\ \text{False Negative} & \text{True Positive} \end{bmatrix},$$

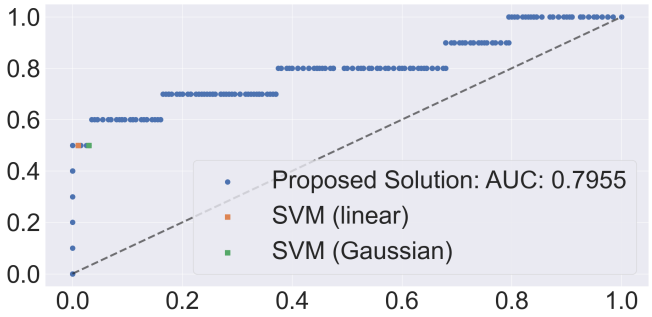
Ideally, we would like to see the number of false positives and false negatives be zero. We can compare the ROC curve the confusion matrix of the local SVM solution, by plotting if as a single point on the ROC curve.

We start by examining the task 1 through 3, for which the ROC curves may be found depicted on figure 5.8. The y-axis of the ROC curve should be interpreted as the ratio of abnormal pumps being classified as abnormal pumps (True Positives), while the x-axis represents the ratio of normal pumps being classified as abnormal pumps (False Positives). This means, that is if it one wanted to correctly classify every single abnormal data point in task 1, it would result in ca. 80 % of the normal pumps too being classified as abnormal pumps.

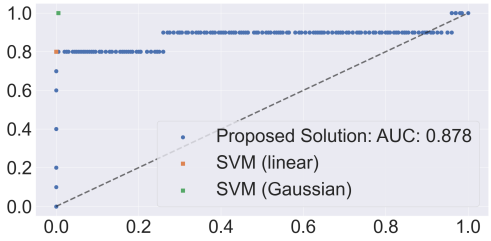
The area under the curve (abbr. AUC) is a measure of comparing the algorithms or in this case tasks. The closer the AUC is to 1, the better the algorithm/solution is.

Despite the lower SNR, it seems that the proposed solution performs better on the data set at 0 SNR, compared to the data set at 6 SNR. It can be seen that the two local SVM solution perform almost as good as the proposed solution in task 1, and as good or better in tasks 2 and 3.

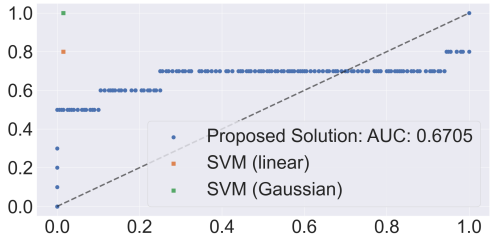
Next we examine the tasks 4 through 6, depicted on figure 5.9. From which it can be seen that the two local SVM solution truly struggle, e.g. the linear SVM classifies all data points as normal pumps. The poor performance seen in the tasks with SNR = -6 is far from unexpected. With a SNR of -6, the power of the noise in the signal is about 4 times that of the actual signals from the pump. In the tasks 5 and 6 we see that the proposed solution and the SVMs perform about similar.



(a) ROC curve for task 1 (SNR: -6).

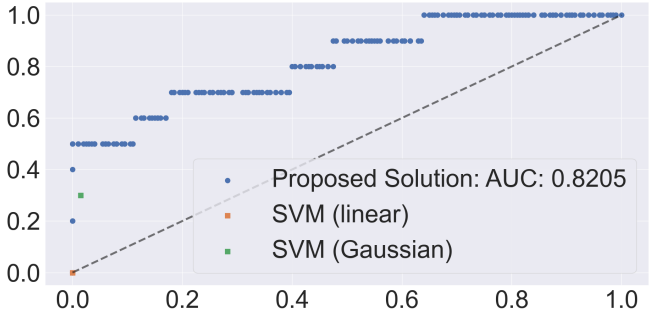


(b) ROC curve for task 2 (SNR: 0).

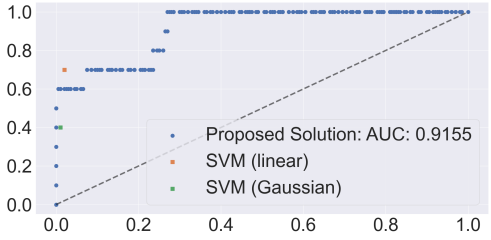


(c) ROC curve for task 3 (SNR: 6).

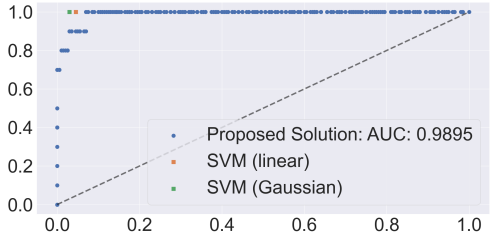
Figure 5.8. Depicts the ROC curves of the tasks 1-3 and the associated SVM confusion matrix points.



(a) ROC curve for task 4 (SNR: -6).

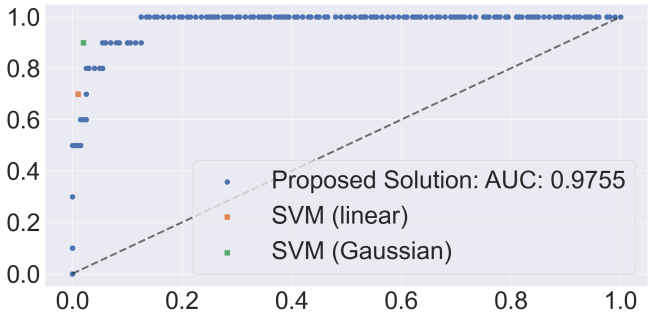


(b) ROC curve for task 5 (SNR: 0).

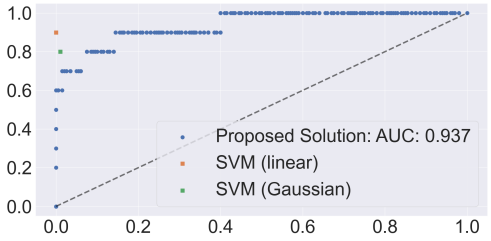


(c) ROC curve for task 6 (SNR: 6).

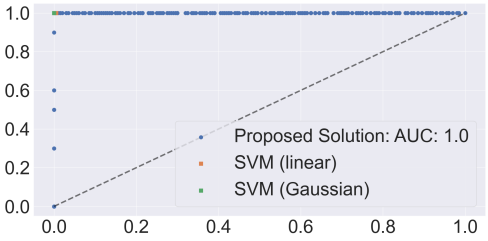
Figure 5.9. Depicts the ROC curves of the tasks 4-6 and the associated SVM confusion matrix points.



(a) ROC curve for task 7 (SNR: -6).



(b) ROC curve for task 8 (SNR: 0).



(c) ROC curve for task 9 (SNR: 6).

Figure 5.10. Depicts the ROC curves of the tasks 7-9 and the associated SVM confusion matrix points.

Lastly we comment on the results depicted on figure 5.10. As can be seen from figure 5.10c, the proposed solution actually correctly classifies every single data point in the test set, as can be seen from the AUC = 1. For the other tasks, it can be seen that the two local SVM solutions perform better in both cases.

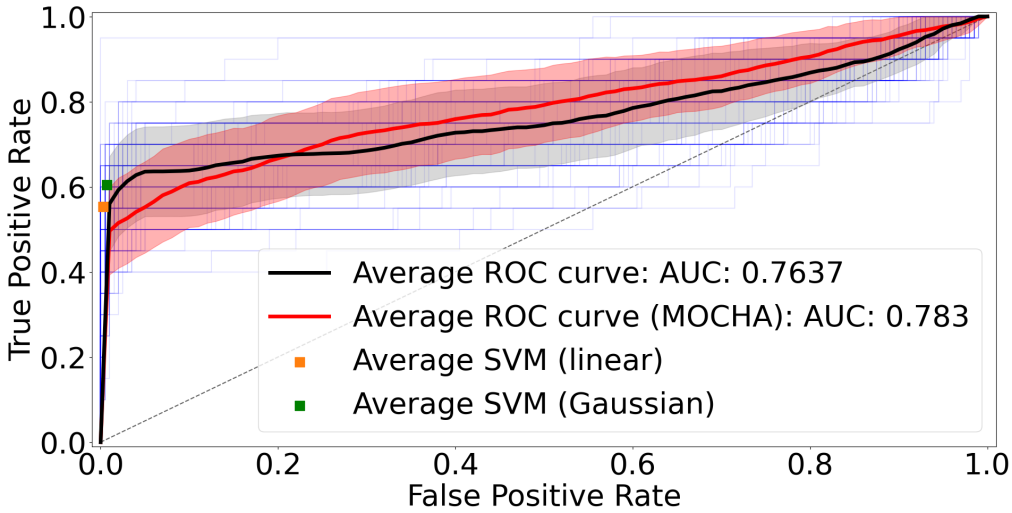


Figure 5.11. Average ROC curves of 100 runs of the proposed solution and the MOCHA algorithm (task 1), along with the average confusion matrix of both linear and Gaussian SVMs.

So far we have examined the results of a single run of the proposed algorithm. Next we exam-

ine what happens in general. We have run the proposed algorithm 100 times, and - for each task - determined the average ROC curve of the proposed solution and the average confusion matrix for the two local SVM solutions. We, likewise, run the MOCHA algorithm 100 times and average the ROC curves. We have selected the best three tasks in terms of AUC, within each of the three task blocks (those being tasks: 1-3, 4-6 and 7-9), to present, which may be found on the figures 5.11 to 5.13. The ROC curves of the remaining 6 tasks may be found in appendix A The grey area defines ± the standard deviation of the ROC curves of the proposed solution. The red area, defines the same, but for the MOCHA algorithm.

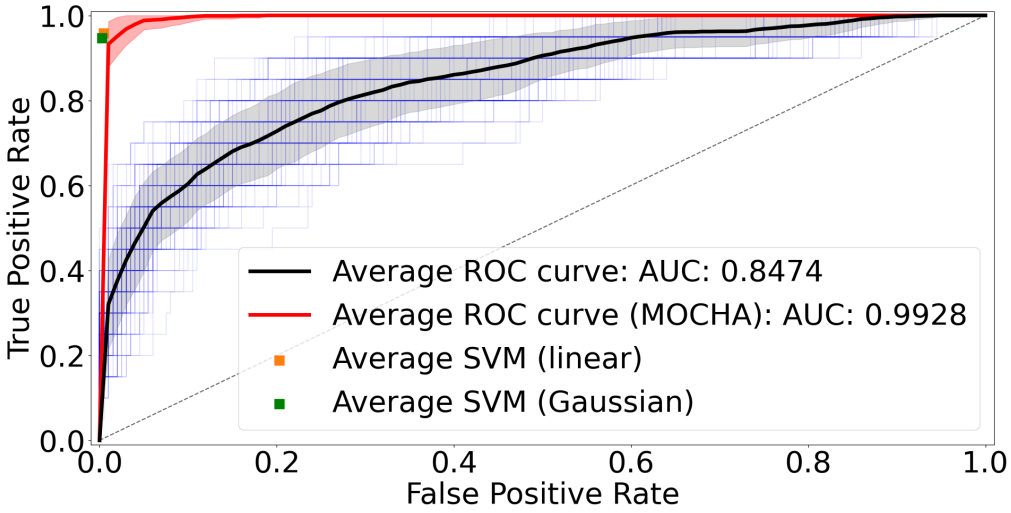


Figure 5.12. Average ROC curves of 100 runs of the proposed solution and the MOCHA algorithm (task 6), along with the average confusion matrix of both linear and Gaussian SVMs.

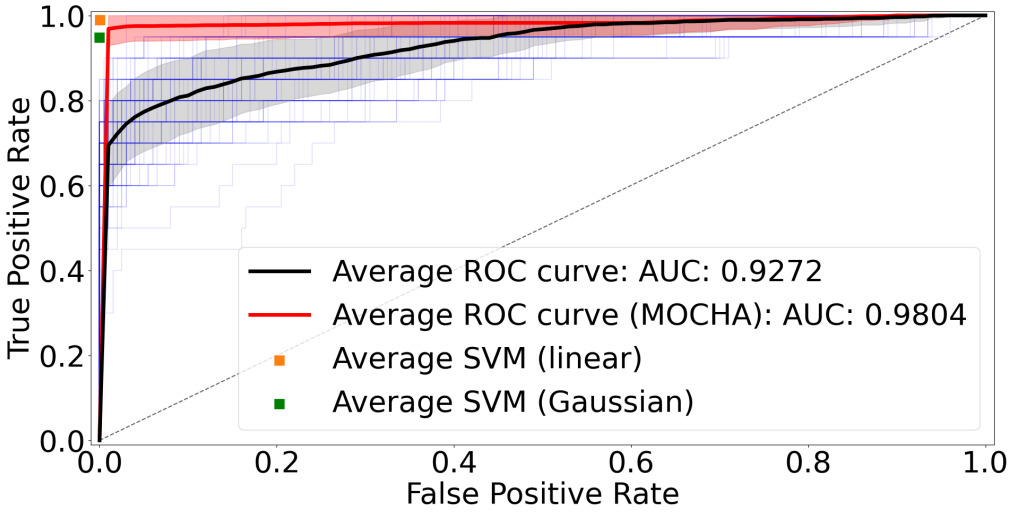


Figure 5.13. Average ROC curves of 100 runs of the proposed solution and the MOCHA algorithm (task 9), along with the average confusion matrix of both linear and Gaussian SVMs.

Lastly we present table 5.5, which shows the AUCs of both the proposed solution and the MOCHA algorithm, along with the signed difference between them. From table 5.5 it can be

Table 5.5. Contains the AUCs of both the proposed solution and the MOCHA algorithm along with the difference for each task.

Task no.	AUC of Proposed Solution	AUC of the MOCHA algorithm	Difference
1	0.7637	0.783	0.0193
2	0.7025	0.8475	0.145
3	0.6749	0.8692	0.1943
4	0.6944	0.7516	0.0572
5	0.7344	0.8844	0.15
6	0.8474	0.9828	0.1354
7	0.8186	0.832	0.0134
8	0.8656	0.9524	0.0868
9	0.9272	0.9804	0.0532
Average	0.7810	0.8760	0.095

seen that the MOCHA algorithm performs better in every single task, however, on average, the difference is small.

5.2.3 Convergence

The results presented above we found that the algorithm converged in 11 rounds. However, this is unfortunately not something that we found in general. occasionally, we found a timely convergence, that is when the number of rounds we less than 15, however in most cases the number of rounds needed to converge was far larger than 15.

Obviously there is a relationship between the tol, λ and the number of rounds needed to converge. We chose not to alter the $tol = 0.1$, as changing this parameter makes little sense. Thus, the only parameter that could be experimented with is λ. λ describes the ration of importance between collaboration of the tasks and the individuality of the tasks. That is, the smaller the λ, the less emphasis the algorithm puts on collaboration. In theory for larger λ the algorithm should converge faster, as the algorithm forces the weights to be similar meaning less changes in the weights, thus the convergence criteria is met faster.

We conducted a small experiment to investigate how large we needed to select λ to force convergence within 15 rounds. We found that the average λ needed was 0.5304 with a variance of 0.3874. This experiment was conducted from 100 runs of the algorithm, each time randomly choosing the training set.

6 Discussion

We shall in this chapter discuss the results presented in chapter 5. Furthermore, we shall discuss the proposed solution it self, presented in chapter 4.

6.1 The Proposed Solution

We have in chapter 4 presented the proposed solution. The goal was to construct an algorithm with low computational complexity and low requirements on communications. These goals have been fulfilled, as we have shown in section 4.2 that the number of FLOPS is well under the number of FLOPS for a SVM solution, and we have shown that the algorithm solely needs to communicate with the central server.

In this section, we shall discuss the proposed solution in depth, as from chapter 5, it has become clear that there are issues with the proposed solution. Firstly we shall go through the logic of the path chosen to derive the proposed solution. As mentioned in the introduction of chapter 4, common for the existing similar solutions is the use of the quadratic approximation of the dual problem. To draw parallels, that would be the same had we made a quadratic approximation of (4.8). However, it was decided that this project should take another approach.

Since the convex conjugate of any convex function, is also convex and the fact that the second term in (4.8) is linear, we knew that choosing a convex loss function would result in (4.8) being convex. Any minimum in a convex optimisation problem is a global minimum, which can be obtained from equating the derivative to zero. Thus, began the search for the derivative of the convex conjugate of the loss function.

Though the mathematical analysis in section 4.1 leading to the proposed solution is mathematically sound, the result in a way be counter intuitive. If a data point is correctly classified, we should strive not to change the weights in the upcoming iteration. However, this is not exactly what is happening. For every correctly classified data point the associated Lagrangian multiplier α is set to zero. This means that only wrongly classified data points have influence on the $\boldsymbol{b}$ vector from (4.5). In the current version of the proposed solution, there are no incentives to keep the current version of the weights. There is no reward for correctly classifying e.g. 90% of the data point, only great penalty for wrongly classifying 10% of the data points. Note

that we could take these numbers to the extremes: Imagine that all data points are correctly classified. This implies that $\boldsymbol{\alpha} = \mathbf{0}$, which in turn implies $\boldsymbol{b} = \mathbf{0}$. Thus, that the weight from this particular task will - in the upcoming weight update using (4.5) - solely be defined by the weights from the other tasks based on the relationship with those tasks. Then in the extreme case; imagine that the particular task in question has no relation with any of the other tasks. In that case the weights of the particular task is set to $\boldsymbol{w} = \mathbf{0}$, starting all over with the algorithm. In one iteration, we went from a perfect classifier, to a classifier that classifies every single data point as the same class.

The scenario describe, is the reason we need the large values of λ to see convergence. If we compare with similar literature, such as [11], the choice of λ range from 10^{-6} to 10^{-5}. The average obtained in the experiments from section 5.2.3 is about 0.5, which is orders of magnitude larger than 10^{-5}. This means that the proposed solution puts a huge emphasis in the collaboration of tasks, and could in some cases even force a relationship between two tasks, where there might not be one.

6.2 Performance of the Proposed Solution

We have in chapter 5 presented the results obtained from multiple experiments using the proposed solution. We have in he previous section discuss how the proposed solution, intuitively makes little to no sense, besides the fact that is solve the optimisation problem (4.9). Despite this, the proposed solution performs surprisingly well. From the figures 5.11 to 5.13 in chapter 5 and the figures A.1 to A.6 in appendix A, we observe that the proposed solution perform in general as well as or worse than the three methods of comparison, those being a local SVM (linear), a local SVM (Gaussian) and the MOCHA algorithm. We make these claims based on the AUCs and the placements of the SVM confusion matrices. We notice that gap seems to be larger for tasks with higher SNR (those being tasks: 3,6,9), while there is almost no difference for the tasks with low SNR (those being tasks: 1,4,7). The results on the aforementioned figures are as expected, in terms of the differences between the proposed solution and the MOCHA algorithm. The requirements of the MOCHA algorithm are more demanding, as it requires approximately 150 times the number of FLOPs, and requires communication between the nodes. That is, after a completed round on a node, said node will communicate its results to all of the other nodes. This way, the communication requirements for MOCHA scales quadratic with the number of nodes, while the communication requirements of the proposed solution only scales linearly. Reducing the number of FLOPs by 150 times, is probably not going to be enough of an argument to choose the proposed solution over the MOCHA algorithm. The only scenario, where one might make the choice in is low SNR environment, where the performance of the proposed solution seems to be the same as the MOCHA algorithm. These claims are based entirely on the experiments run in this thesis, as we might experience completely different results, with a different data set.

6.3 Predictive Maintenance as Binary Classification

In this section, we address the binary classification setting for predictive maintenance in this scenario described in chapter 1. Recall that the motivation for multi-task learning, was that the nodes (pumps) could potentially be placed in different noise environments. This also means that in practice the data would have to be recorded post installation of the product (pump). Here lies the problem: This would mean that abnormal data would have to be recorded post installation. It is not unreasonable to assume that this is quite difficult. As the operator installing the pump, would have to forcefully break the pump to achieve these data, possible completely destroying the pump in the process. In that case a new pump would be needed to be installed, completely undermining the point of learning a model of each pump. Thus, the choice of binary classification for predictive maintenance is probably not the best choice.

While it is impossible to do binary classification with only normal data points, it is, however, totally possible to do anomaly detection using only normal data points. Anomaly detection is the act of detecting when a new data point is different from all of the training data. An intuitive way of thinking of this is to build a transformation such all the training data of transformed in a circle or an oval. This way all data point that fall outside the oval are considered abnormal. [22]

7 Conclusion

We have in this thesis investigated the possibility of constructing a federated multi-task learning algorithm with a focus on minimal communications and computational power, for predictive maintenance use. The overall idea of the thesis was to take inspiration from already existing work, such as [3, 4, 11], to create an algorithm that was suited for both the problem and the specification of the problem, described in chapter 1.

The derivation of the proposed solution is based upon a primal problem similar to those of [3, 4, 11]. Even, the update of the relation matrix $\boldsymbol{D}$ (the matrix that describes the relations between the tasks), is equivalent. The main difference between the proposed solution and similar literature ([3, 4, 11]) is the formulation and solution method of the local dual problem, the problem that is solved on the nodes (pumps). Where the proposed solution takes a naive and direct approach, the likes of [3, 4, 11] all conduct quadratic approximation and solve via the stochastic dual coordinate ascend method. Through appropriate choice of loss function (squared hinge-loss) and the realisation of the convexity of the local dual problem, led to an analytical solution of the local dual problem. These observation gave rise to the proposed algorithm found in chapter 4, algorithm 5.

We have in this thesis succeeded in developing an algorithm that performs binary classification, with a computational complexity that is much smaller than that of a support vector machine (abbr. SVM) solution, in terms of number of FLOPs. We have succeeded in developing an algorithm, that allows for asynchronous updates of the weights, as each node is free to contact the central server at any given time. This is contrary to existing literature mentioned, which all - in some way - require synchronous updates of the weights and the relation matrix. However, as can be found in chapter 5, the accuracy of the proposed solution and the receiver operating characteristics are inferior to those of the two local SVM solutions with linear and Gaussian kernels, and the MOCHA algorithm.

In order to obtain the necessary knowledge to develop a federated multi-task learning algorithm, we have investigated and documented, how machine learning functions in a federated setting. More specifically we have - in section 3.2 - studied federated learning, which is a technique that allows for nodes to - via communication with a central server - collaborate on learning a single global model. We have in section 3.3 introduced the concept of multi-task learning, which is a technique that allows for multiple machine learning schemes to collaborated

and help each other in achieving their individual goals, of each learning a model that fits their respective data. Included in section 3.3 is a multitude of theorems and results that are essential in the combination of federated learning and multi-task learning.

Lastly, we give the final verdict over the proposed solution. The main two arguments for deploying the proposed solution are; if there are strict requirements on the communication in the network and limited processing power and memory available at the nodes. The main argument against the proposed solution is the inferior accuracy. While the MOCHA algorithm and the SVM solutions would require more memory and processing power to complete in the same amount of time, they have - in some cases - a vastly better accuracy. Thus, serious considerations are needed, in regards to the importance of limiting the computational complexity at the cost of loosing accuracy, before deploying the proposed solution.

8 Further Development

Unfortunate, we ran out of time during the process of completing this master's thesis. In this chapter we discuss ideas and concepts that could improve upon the proposed solution from chapter 4. With more time, we would have investigated this.

We have discussed how the intuitive understanding of the proposed solution is contradicting to the overall goal. We have described how correctly classified data points have zero influence on how the future weights are shaped. In an attempt to combat this, we see a potential solutions. We have discussed how, when most of the data points are being correctly classified only a very few select data point decides how the next iteration of the weights are shaped. We propose a solution to this problem, by letting the $\boldsymbol{b}_t$ vector from eq. (4.5) be define by a weighted average. Note that $\boldsymbol{b}_t$ - in the thesis - is defined as follows,

$$\boldsymbol{b}_t = \boldsymbol{X}_t \boldsymbol{\alpha}_t, \tag{8.1}$$

where the columns of $\boldsymbol{X}_t$ contains the data points from the tth task, and α_t the variable we optimise over in the local dual problems. The nth entry of α_t is set to zero if the nth data point in task t is correctly classified. As can be imagined, the current version of $\boldsymbol{b}_t$ can potentially become very unstable, get stuck in a loop, or even be set to zero, as discussed in chapter 6.

The idea is, instead of giving the misclassified data points sole jurisdiction over $\boldsymbol{b}_t$, we propose to let the correctly classified data points to have influence as well. This is done by introducing a convex sum between $\boldsymbol{X}_t \boldsymbol{\alpha}_t$ and the current set of weights. That is, at the ith iteration at the tth node, we define

$$\boldsymbol{b}_t^{(i)} = \left(1 - \frac{\left|\mathcal{N}_t^{(i)}\right|}{N_t}\right) \boldsymbol{X}_t \boldsymbol{\alpha}_t^{(i)} + \frac{\left|\mathcal{N}_t^{(i)}\right|}{N_t} \boldsymbol{w}_t^{(i-1)}, \tag{8.2}$$

where $\mathcal{N}^{(i)} = \left\{ n \in \mathbb{N} \mid \alpha_{n,t}^{(i)} = 0 \right\}$, $\left|\mathcal{N}^{(i)}\right|$ denotes the cardinality of $\mathcal{N}^{(i)}$, N_t is the number of data points in the tth task, and the use of superscript, (i), denotes the ith iteration.

The intuitive interpretation of (8.2) is that, we are still updating $\boldsymbol{b}_t$ as in the thesis, but we weight it be the number of points that are wrongly classified. Simultaneously, we weigh the weights from the previous iteration by the number of correctly classified data points. This way we have build a sort of democratic voting system on, not if there should be change, but to what degree the weights should change.

As mentioned, if ever single data point in a task is correctly classified at iteration i, then by (8.1), $\boldsymbol{b}_t^{(i)} = \mathbf{0}$, which is possibly the worst choice of weights. If we, however, use (8.2), and ever single data point in a task is correctly classified at iteration i, then $\boldsymbol{b}_t^{(i)} = \boldsymbol{w}_t^{(i-1)}$.

If we were to put this in the context of predictive maintenance, one could assign different weights to the individual data points. By this, we refer to the possibility of giving data points with the "abnormal" label more influence than those with a label of "normal". This could potentially combat an imbalanced data set. In (8.2) we assume that all data points have identical influence.

Bibliography

[1] Faisal Zaman. *Instilling Responsible and Reliable AI Development with Federated Learning.* medium.com, 2020.

[2] Amaury Bouchra Pilet, Davide Frey, and Taïani François. *Simple, Efficient and Convenient Decentralized Multi-Task Learning for Neural Networks.* hal-02373338v4, 2020.

[3] Virginia Smith, Sebastian Caldas, and Ameet Talwalkar. *Federated Kernelized Multi-Task Learning.* SYSML, 2018.

[4] Virginia Smith, Chao-Kai Chiang, Maziar Sanjabi, and Ameet Talwalkar. *Federated Multi-Task Learning.* ArXiv, 2018.

[5] Omkar Motaghare, Anju S Pillai, and K.I. Ramachandran. *Predictive Maintenance Architecture.* IEEE, 2018.

[6] *grundfos.dk.* Last visited the 3rd of may 2021.

[7] Filip Hanzely and Peter Richtárik. *Federated Learning of a Mixture of Global and Local Models.* arXiv, 2020.

[8] Zheng Wang, Fan Xiaoliang, Jianzhong Qi, Chenglu Wen, and Rongshan Yu. *Federated Learning with Fair Averaging.* arXiv, 2021.

[9] Andreas Argyriou, Theodoros Evgeniou, and Massimiliano Pontil. *Convex Multi-Task Feature Learning.* Springer, 2008.

[10] Yu Zhang and Dit-Yan Yeung. *A Convex Formulation for Learning Task Relationships in Multi-Task Learning.* arXiv, 2012.

[11] Sulin Liu, Sinno Jialin Pan, and Qirong Ho. *Distributed Multi-Task Relationship Learning.* arXiv, 2017.

[12] Stephan Boyd and Lieven Vandenberge. *Convex Optimization.* Cambridge University Press, 2004. ISBN 978-0-521-83378-3.

[13] Rui Li, Fenglong Ma, Wenjun Jiang, and Jing Gao. *Online Federated Multitask Learning.* IEEE, 2019.

[14] Christopher M. Bishop. *Pattern Regocnition and Machine Learning*. Springer Science, 2006.

[15] *scikit-learn.org/stable/modules/generated/sklearn.svm.SVC.html*. Last visited the 4rd of may 2021.

[16] H. Brendan McMahan, Daniel Ramage, and Blaise Aguera y Arcas. *Federated Learning of Deep Networks using Model Averaging*. arXiv, 2016.

[17] H. Brendan McMahan, Seth Hampson, Daniel Ramage, and Blaise Aguera y Arcas. *Communication-Efficient Learning of Deep Networks from Decentralized Data*. arXiv, 2017.

[18] Yu Zhang and Qiang Yang. *A Survey on Multi-Task Learning*. arXiv, 2018.

[19] Rob Haggarty. *Fundamentals of Mathematical Analysis*. Addison-Wesley, second edition, 1993.

[20] Alistair Shilton, Daniel Ralph, and Marimuthu Palaniswami. *Incremental Training of Support Vector Machines*. IEEE, 2005.

[21] Harsh Purohit, Ryo Tanabe, Kenji Ichige, Takashi Endo, Yuki Nikaido, Kaori Suefusa, and Yohei Kawaguchi. *MIMII DATASET: SOUND DATASET FOR MALFUNCTIONING INDUSTRIAL MACHINE INVESTIGATION AND INSPECTION*. arXiv, 2019.

[22] Sangamesh Ragate, Ndim Hmeidat, and Mustafa Aljumaily. *Pattern Rcognition "Anomaly Detection Challenges"*. ResearchGate, 2015.

A Remaining ROC curves for tasks: 2-4,7,8

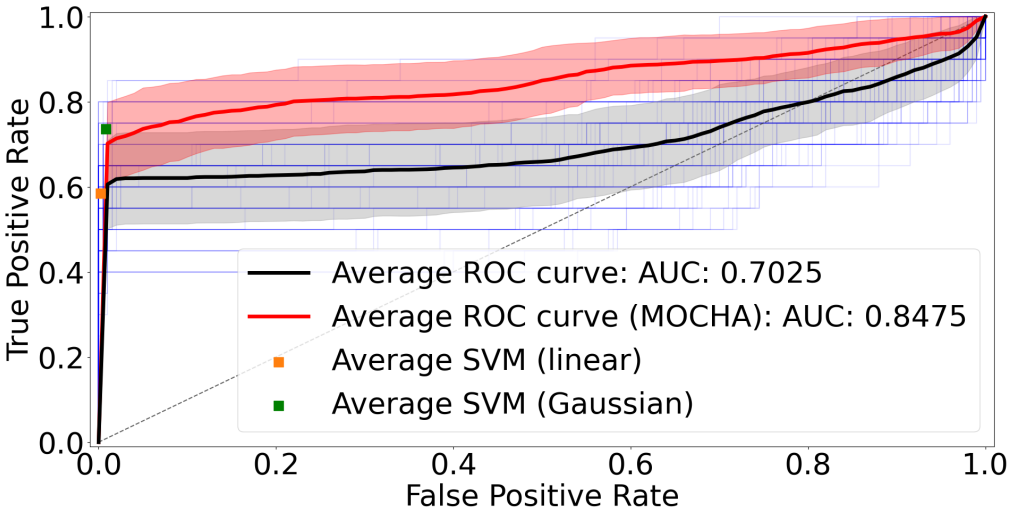


Figure A.1. Average ROC curves of 100 runs of the proposed solution and the MOCHA algorithm (task 2), along with the average confusion matrix of both linear and Gaussian SVMs.

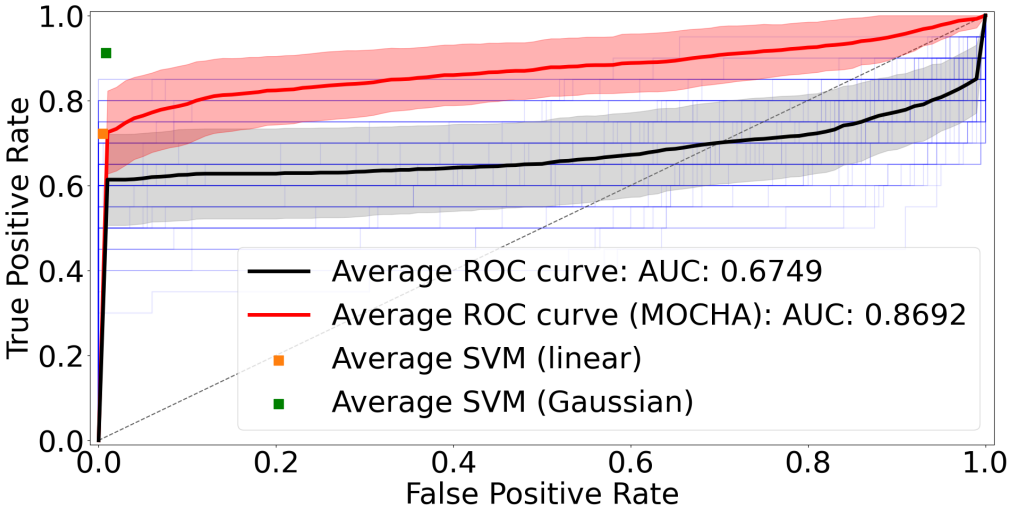


Figure A.2. Average ROC curves of 100 runs of the proposed solution and the MOCHA algorithm (task 3), along with the average confusion matrix of both linear and Gaussian SVMs.

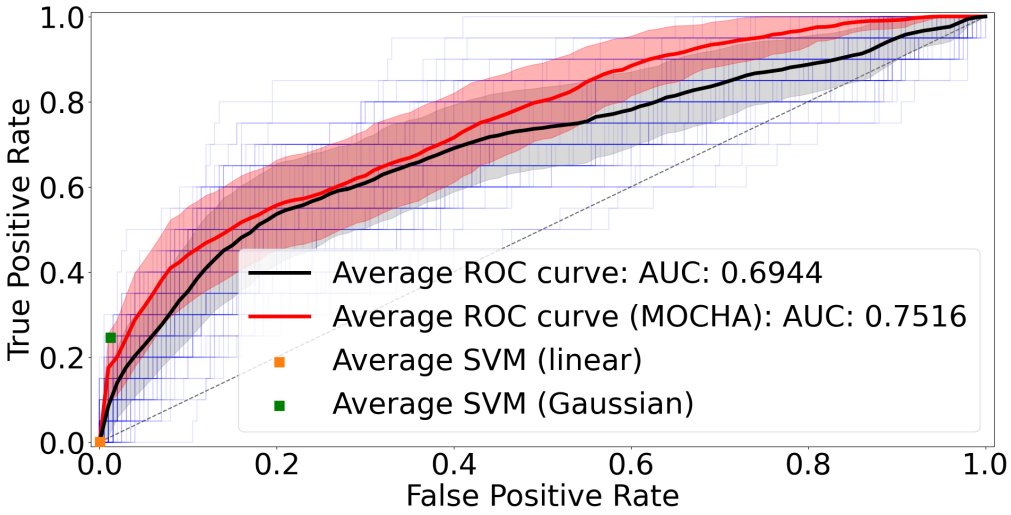


Figure A.3. Average ROC curves of 100 runs of the proposed solution and the MOCHA algorithm (task 4), along with the average confusion matrix of both linear and Gaussian SVMs.

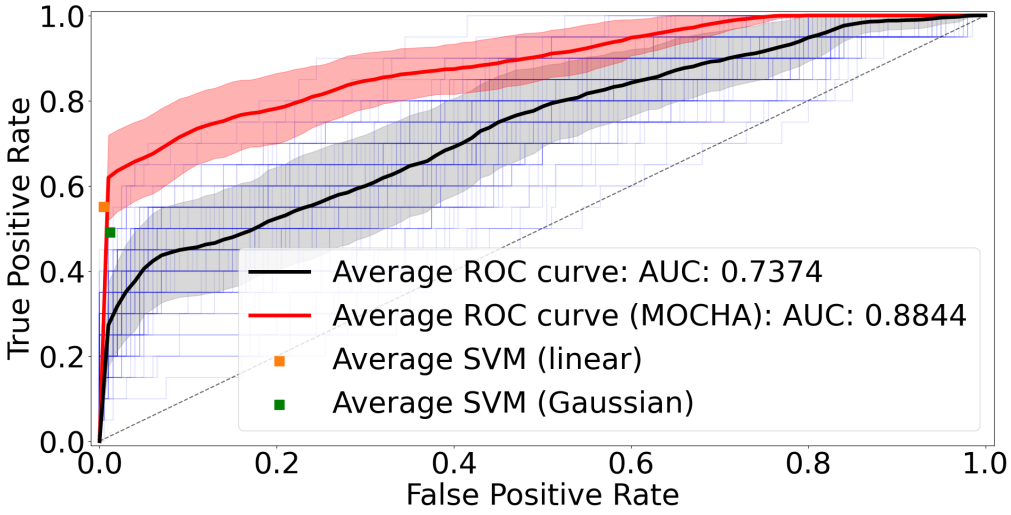


Figure A.4. Average ROC curves of 100 runs of the proposed solution and the MOCHA algorithm (task 5), along with the average confusion matrix of both linear and Gaussian SVMs.

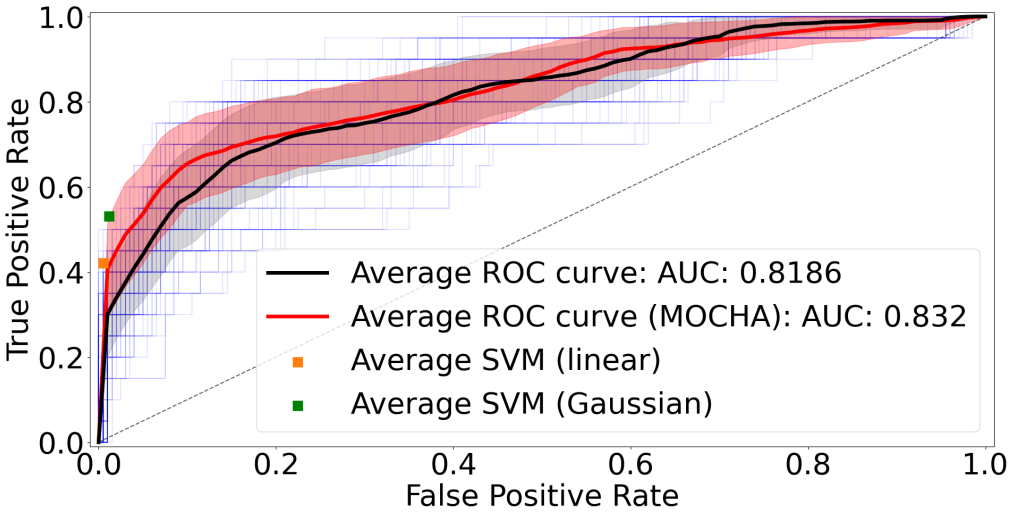


Figure A.5. Average ROC curves of 100 runs of the proposed solution and the MOCHA algorithm (task 7), along with the average confusion matrix of both linear and Gaussian SVMs.

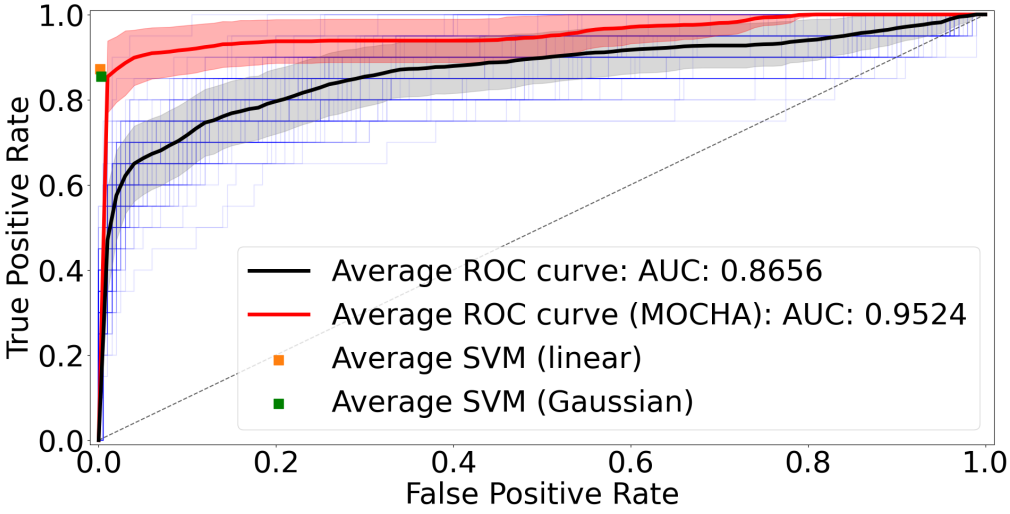


Figure A.6. Average ROC curves of 100 runs of the proposed solution and the MOCHA algorithm (task 8), along with the average confusion matrix of both linear and Gaussian SVMs.

B Accompanying Python Code

Accompanying the master's thesis, is a ROC_MANY.PY file. Running the file, with the MIMII data set and the VAGT.MAT file in the appropriate location, will result in the averaged ROC curves seen in appendix A. Note that the VAGT.MAT contains the weights obtained from running the MOCHA algorithm 100 times in matlab. The VAGT.MAT is as well as the ROC_MANY.PY file is accompanying the master's thesis. However, the matlab source code for the MOCHA algorithm is not. We do, however, refer the reader to [4], for the code. The code created for this master's thesis is written in spyder 3.8.