Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

School of Computer Science and Statistics

Federated Meta-Learning: A Novel Approach for Algorithm Selection

Mukesh Arambakam
[redacted]

Supervisor: Dr. Joeran Beel

A Dissertation submitted to the University of Dublin,
in partial fulfilment of the requirements for the degree of
Master of Science in Computer Science (Data Science)

2020

Declaration

I hereby declare that the work described in this dissertation is, except where otherwise stated, entirely my own work, and has not been submitted as an exercise for a degree at this or any other university

I have read and I understand the plagiarism provisions in the General Regulations of the University Calendar for the current year, found at `http://www.tcd.ie/calendar`.

I have also completed the Online Tutorial on avoiding plagiarism 'Ready Steady Write', located at `http://tcd-ie.libguides.com/plagiarism/ready-steady-write`.

Signed: [redacted] Date: 07-09-2020

. Mukesh Arambakam

Permission to lend and/or copy

I agree that the Trinity College Library may lend or copy this dissertation upon request.

Signed: [redacted] Date: 07-09-2020
. Mukesh Arambakam

Acknowledgements

Throughout the course of this dissertation I have received a great deal of support and assistance from a lot of people.

First and foremost, I would like to thank my supervisor, Dr. Joeran Beel, for the research idea and for his continuous support throughout the project. I also would like to thank him for believing in me and giving me the autonomy to work on the project and develop each stage of the application as I wanted.

Secondly, I would like thank my friends Manasi, Aishwarya and Jagadish, who provided me with advice and proofread my report. I would also like to thank them for their efforts in acting as alpha testers of the application and providing me with valuable feedback in improving the user-experience of the application.

Finally, I would like to thank my family for their constant support and patience during these unforeseen circumstances, especially my mother for her constant moral support without whom this would not have been possible. Thanks Mum!

Abstract

"Federated Meta-Learning" (FML), a concept that allows everyone to benefit from the data that is generated through software libraries including machine learning and data science libraries. It focuses on learning the algorithm performance measures and making recommendations based on the model created. I introduce FMLearn, an application developed using the client-server model, which allows the exchange of meta-data about machine learning models and data in itself, for the purpose of meta-learned algorithm selection and configuration. The input to FMLearn is a dataset and the output is a recommendation for the potentially best performing algorithm(s) and it's hyper-parameters to solve the task. This recommendation is made by a model built using the Meta-Features obtained from the dataset description, using the K-Nearest Neighbours algorithm and historic performance data. Scikit-Learn's toy datasets along with other datasets from UCI Machine Learning repository were used and evaluated against various machine learning algorithms using GridSearchCV and Cross-Validation for which the execution time was measured. In the case previously seen datasets like scikit-learn's breast cancer dataset, an execution time of approximately 94.24min was recorded to find the best algorithm by performing Grid Search and Cross-Validation. Whereas, when FMLearn was asked to recommend the best algorithm, the application only took 3secs to recommend the best performing algorithm along with its model parameters. For a large dataset like the skin segmentation, traditional means for finding the best performing algorithms takes approximately 869.74 minutes and when FMLearn was asked to recommend the same, it took only 3secs. In the case of previously unseen but similar datasets, the recommended algorithm is accurate, but the hyper-parameters required re-optimisation to suite the dataset. For a previously unseen and highly dissimilar dataset, FMLearn recommends a list of algorithms which it thinks are best suited for that dataset based on its prior knowledge. In this case, FMLearn recommends the best performing algorithm about 60% of the time. Overall, the use of this application allows the user to scale down an average of 86.718% and 95.762% of time and electricity for small and large datasets respectively by eliminating the repetitive and time consuming task of algorithm selection and configuration from the Machine Learning Workflow.

Keywords

Federated Meta-Learning, FMLearn, AutoML, Machine Learning, RecSys, Dataset Meta-Features, Algorithm Selection, Algorithm Configuration

Contents

List of Figures

List of Tables

Nomenclature

Auto*	Automated
AutoML	Automated Machine Learning
API	Application Programming Interface
DAA	Distributed Application Architecture
DML	Distributed Machine Learning
FML	Federated Meta-Learning
FedML	Federated Machine Learning
HTTP	Hypertext Transfer Protocol
JSON	JavaScript Object Notation
KNN	K-Nearest Neighbours
ML	Machine Learning
MtL	Meta-Learning
NF	Normal Form
NP	Non-Deterministic Polynomial Time
PaaS	Platform as a Service
SAT	propositional satisfiability problem
SQL	Structured Query Language
SVM	Support Vector Machines
i	Single Instance from a Set
m	Cost Metric
$\mathcal{A}$	Algorithm
$\mathcal{D}$	Dataset
$\mathcal{I}$	Set of Instances
$\mathcal{P}$	Portfolio of Algorithms
$\mathcal{R}$	Problem Statement

All measurements of time used in the report are in **minutes** unless specified otherwise.

All measurements of power specified in this report are in **watts** unless specified otherwise.

1 Introduction

An ever-growing number of algorithms are used to solve machine learning and data science tasks, the challenge of algorithm selection and configuration is subjected to intensive research. (1, 2, 3, 4, 5, 6). The **algorithm selection problem** characterises the challenge of finding an effective algorithm for a given dataset or task on an *instance-by-instance basis* from a set of algorithms. This problem is motivated by the fact that different algorithm performs differently on different datasets. When put simply, algorithm A performs poorly on a dataset X, while algorithm B performs well, choosing algorithm B over algorithm A for the given dataset X, without having to run both algorithm A and B to find out which of these would be a better performer is the challenge of algorithm selection. There are *various techniques* to find out which algorithm works better, few examples of such techniques would be winner-takes-all, heuristic approximation, feature analysis, meta-learning and many more. In this research paper, the focus is on the Meta-Learning approach.

Meta-learning is one of the most promising techniques to warm start the algorithm selection and configuration process (7). With meta-learning, a machine learning model is trained to predict how algorithms perform on various tasks. The meta-learning model is built based on the past performance of algorithms over a large number of tasks or datasets, which are described through meta-features. It does this by learning the relationships between the different meta-features of data, and performance of the algorithm on the data. For unseen tasks, the best performing algorithms can be predicted through the meta-learner (and subsequently be optimized using techniques such as Bayesian Hyper-Parameter Optimization). These meta-features are also known as the characterisation measures and there are no standard set of meta-features which can be used to describe the dataset accurately, but there are several approaches to characterise the dataset as described in (8), (9) and (10). These meta-features are used to describe and characterise the dataset, which is then used to provide an assessment of algorithm's performance.

1.1 AutoML

Automated Machine Learning or AutoML, is the process of automating the time consuming task of Algorithm Selection and Optimizing the Hyper-Parameters. The task of Algorithm Selection is usually done on an instance by instance basis. There are many tools which offer this feature like Auto-Sklearn, OpenML, etc., however, these tools select the best performing algorithm by predicting the performance of each algorithm for a given task by running various algorithms on a subset of the entire dataset and then choosing the best performing algorithm on that subset for the task. Though this is just a over-simplified explanation of what the libraries do internally, this is a time consuming task, and in-turn electricity and computational power before it could suggest the best performing algorithm for the task. Even if the task is to find the best algorithm for a previously seen dataset the entire process is repeated.

The same can be said for Hyper-Parameter Optimisation / Tuning as well, this is a process of selecting a set of optimal hyper-parameters for an algorithm. A hyper-parameter is a parameter whose value is used to control how a machine learning algorithm solves a given problem. Hyper-Parameter Optimisation is also a task that takes a great deal of time, electricity and computational power, before an optimal set of hyper-parameter can be chosen for an algorithm-dataset pair. Hyper-Parameters are selected by building models based on a combination of parameters specified and evaluating the models to find the best set of parameters for which the selected algorithm performs the best.

1.2 Motivation

A challenge in algorithm selection and configuration is the (non) availability of data in some disciplines to build the meta-learning model, which is due to the workflow of machine learning, data science or other projects. Typically, software libraries – be it machine learning libraries like Auto-sklearn (8), Auto-Weka (11) or ML-Plan (12), recommender system libraries like LibRec-Auto (13), Auto-Suprise (14) are used in isolation, either locally or in the cloud. The term "in isolation" implies that, the information regarding how algorithms perform on a particular dataset, is neither published nor shared. Consequently, computationally expensive algorithm selection and hyper parameter optimization is performed by each machine learning engineer over and over again for the same datasets, which is a huge waste of Time, Electricity, Computational Power and Money.

1.3 Research Problem

The resources such as time spent for the task of algorithm selection and hyper-parameter optimisation is justifiable for the first time search of the best algorithm is performed. On the other hand, with the increasing popularity and use of Machine Learning for various tasks, time, electricity and computational power spent in finding the best algorithm for a given a dataset, along with it's optimal hyper-parameters becomes a repetitive task.

Considering a small and simple dataset like the Breast Cancer Dataset with 569 records and 30 features, spending 94.24 minutes to find the best performing algorithm is a very long time for which the developer has to wait before proceeding with further computational tasks. This also means an average of about 125.65W of power is being consumed by the computer (check Appendix A1.2 for power consumption on the tested machine). For a large dataset like the Skin Segmentation dataset with 245057 records and 4 features, it takes approximately 869.74 minutes to find the best performing algorithm and this task consumes approximately 1159.65W of energy. Even if this task is repeated a couple of hundred times, the electricity wasted performing the same task repeatedly is huge!

The problem here is the workflow of machine learning and data science libraries, these libraries usually work in isolation and the information regarding how an algorithm performs on a particular dataset is neither published nor shared. This leads to computationally expensive and time consuming task like algorithm selection and hyper-parameter optimisation being performed repeatedly for the same dataset.

1.4 Federated Meta Learning

Beel proposed "Federated Meta-Learning" (15), a concept that allows everyone to benefit from the data which is generated through software libraries including standard machine learning and data science libraries as well as the auto* tools.

Federated Meta Learning focuses on learning the algorithm performance measures for arbitrary tasks across devices. He envisioned federated meta learning as an ecosystem where the raw data is kept on the original devices and the meta data, algorithm names, combined with performance metrics of the algorithm on the tasks would be stored on a central FML server (though a peer-to-peer architecture might also be possible). Using this historic performance data, a model is trained and is used to predict the best performing algorithm along with its hyper-parameters for a previously seen or unseen task.

1.5 Research Goal

The goal of this research is to facilitate the algorithm selection and configuration process. This can be done by making it faster with the use of historic performance data, produced on various devices and by various machine learning algorithms and libraries. This data can be used to improve the performance of algorithm selection, thus saving time, electricity, computational power and money which will otherwise be required for finding the best algorithm and its optimal hyper-parameters for the task.

To improve the algorithm selection and configuration process, a prototype of Federated Meta Learning named **FMLearn** is presented. The potential benefits of this research include the development of FMLearn a tool that applies the concepts of Federated Meta-Learning to suggest users with the best algorithm along with it's hyper-parameters for a previously seen or unseen task.

The goal is to develop an application, **FMLearn** as a simple proof-of-concept for Federated Meta-Learning and it should allow everyone to benefit from the data that is generated through machine learning and data science libraries. When put simply, the input to FMLearn is a dataset and the output is a recommendation for the potentially best performing algorithm(s) and it's hyper-parameters to solve that task.

FMLearn also provides an additional benefit by acting as a publicly available knowledge base or a directory of algorithms-data performance measures with an ability to improve and/or add data to this knowledge base. Additionally it also showcases an ability to use this knowledge via the API's provided by the FMLearn application for other similar tasks. Another important benefit is the ability for any Machine Learning or Data Science tool to use FMLearn irrespective of the programming language used to build them to get a recommendation for the potentially best performing algorithm(s) and it's hyper-parameters to solve it's task without being constrained by the need to use the client that is supported by this paper (modified scikit-learn).

1.6 Contributions

This research utilises the novel concept of Federated Meta-Learning in it's core, applies it's principle and proposes the first prototype of the application FMLearn. The application will ultimately facilitate the algorithm selection and configuration process in finding the best performing algorithm for a given task. Apart from this, the research also makes the following contributions to the community:

- FMLearn acts as a knowledge base or directory of algorithms-data performance measures.
- Provides a publicly available API server to facilitate access of the performance measures and also the algorithm selection and configuration process.
- Provides a client implementation in python to access the API's provided by the server.
- This research acts a proof of concept for Federated Meta-Learning and how it helps in saving time and hence electricity, computational power and money for the user.

This research was also published at the *7th ICML Workshop on Automated Machine Learning (AutoML)*, under the name *Federated Meta-Learning: Democratizing Algorithm Selection Across Disciplines and Software Libraries* (16).

2 Background Details

This chapter provides insights about multiple areas related to Federated Meta-Learning. It takes a deeper look into the fundamental concepts related to this research and also discusses other related work and existing research in the fields of Distributed (Machine) Learning, Federated Learning, Meta-Learning, Algorithm Selection and Automated Machine Learning. This chapter also provides an understanding of the current state of research in the areas of Algorithm Selection, AutoML and Meta-Feature abstraction.

2.1 Meta-Learning

Meta-Learning in the context of Machine Learning is learning how to learn. It can also be attributed to the study of principal methods that use meta-knowledge to create efficient models by using machine learning techniques. Meta-Learning algorithm use experience to learn how to perform a certain task, these modified learner are better than the original learner as they have gained additional experience. Each algorithm works on a set of assumptions about the data, that it is **inductively biased**, meaning that the algorithm will learn well (or work as expected) given that the bias matches the learning problem at hand. This introduces restrictions about the type of data and the techniques used to collect this data to perform the meta-learning. By using different types of meta-data, like in this case meta-features of the dataset, it is possible to learn to effectively solve a learning problem. There are three approaches to meta-learning algorithm model-based, metrics-based and optimisation-based.

- **Model-Based**: This type of meta-learning models updates its model-parameters rapidly with training over a dataset.
- **Metric-Based**: The core idea in type of models is similar to that of nearest neighbours algorithms and a relationship between tasks are established.
- **Optimization-Based**: In this case the algorithms plays around with the optimisation algorithms which is used to train the model.

2.2 Automated Machine Learning

Automated Machine Learning or AutoML, is the process of automating the time consuming task of Algorithm Selection and Optimizing the Hyper-Parameters. Traditional machine learning model creation is highly resource-intensive and requires significant domain knowledge and time to create and compare dozens of models. With AutoML, the user will be able to accelerate the time it takes to get ML models with great ease and efficiency. There are many tools that offer to perform this complicated and time consuming task like Auto-Sklearn, Auto-Weka, Auto-Keras, etc. Apart from these tasks AutoML tools also perform feature selection, featuring engineering, data pre-processing, analyse and tune the results. Among the various tasks performed by the AutoML Tools, the most time consuming tasks are Algorithm Selection and Hpyer-Parameter Optimisation.

2.2.1 Algorithm Selection

Algorithm Selection is usually done on an instance by instance basis, meaning different algorithm have to be chosen for different datasets. An algorithm which performs exceptionally well for one dataset may perform poorly for another. The task of choosing an algorithm is dependent on what data we have and how we choose to use it, meaning how the data is cleaned, encoded, imputed, distributed and also how they co-relate with other features or variables within the dataset. The manual task of algorithm selection involves studying and understanding the relationship between different features of the dataset, trying various types of pre-processing steps to bring out the inter-feature relationship which best explains the data.

The tools which perform Algorithm Selection choose the best performing algorithm by predicting the performance of each algorithm for a given task by running various algorithms on a subset of the entire dataset repeatedly after performing different pre-processing steps and then choose the best performing algorithm on that subset for the task. Though this is just a over-simplified explanation of what the libraries do internally, this task is more of a straight forward approach which reduces the effort that a developer needs to put in to select an algorithm. But this process comes with a huge trade-off, it takes up a lot of time, electricity and computational power before it could suggest the best suited algorithm for the task. Even if the task is to find the best algorithm for a previously seen dataset the entire process is repeated.

2.2.2 Algorithm Configuration

The task of Algorithm Configuration can be attributed to Hyper-Parameter Optimisation / Tuning, which is a process of selecting a set of optimal parameters for an algorithm for which it performs best on a dataset. A hyper-parameter is a parameter whose value is used to control how a machine learning algorithm solves or learns to solve a given problem. Hyper-Parameter Optimisation of an Algorithm is a type of Optimisation problem. And a typical Optimisation problem consists of minimising or maximizing the set of parameters along with the metric. This task takes a lot of time to solve by an expert with an in-depth knowledge, and is not something that can be performed with ease by a developer and hence the alternate approach of automating the task is adopted.

There are various optimisation strategies, among them, the commonly used strategies are: Grid search, Random search, Hill Climbing and Bayesian optimization. These techniques are used in combination with cross-validation for evaluating how the results of a statistical analysis will generalise irrespective of the dataset. The working of these strategies are out of scope of this paper, but to give an brief understanding, Grid search is a hyper-parameter tuning approach that will build and evaluate various models, where each model is build on a combination of algorithm parameters specified in a grid. The important thing to note is that these strategies takes up a great deal of time, electricity and computational power, before a optimal set of hyper-parameter can be chosen for an algorithm-dataset pair.

2.3 Workflow of AutoML Libraries

In this section, the workflow of AutoML libraries will be discussed without getting to the details regarding the internal workings of the algorithms or the libraries. The figure 2.1 represents the workflow of a typical AutoML project for a given task or dataset. The data is split into training and testing datasets, the training dataset is first sent to the library for the purpose of model creation. Here, the dataset is pre-processed using various techniques like data-cleaning, data imputation, data encoding, feature selection, feature exploration, feature engineering, feature co-relation, etc..

Once, the data has been pre-processed, the dataset is split into a smaller subsets of the training dataset. One such subset is randomly chosen on which multiple algorithms are run to train and create a model for comparison and evaluation. Once the training and model creation of all chosen algorithms are completed, these models are evaluated based on multiple parameters, one of which is their performance of an unseen data. When the performance metrics are calculated these models are evaluated and compared between each other and the

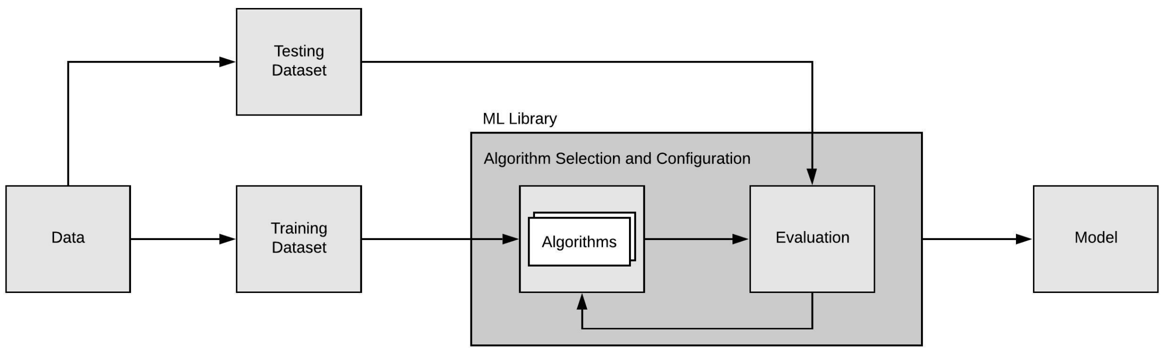


Figure 2.1: AutoML Workflow Diagram

best performing model is chosen for further training. Depending on the internal evaluation strategy of the AutoML tool used, more than one algorithm can be chosen for further training. Once these model(s) are chosen they are trained on various configurations of hyper-parameters using one of these search strategies; Grid Search, Randomized Search, Bayesian Optimization, etc. Once these models are trained and their hyper-parameters are optimised on the complete training dataset, they are evaluated against the testing dataset and the final best performing model is returned to the user.

2.4 Meta-Features

In the notion of Meta-Learning (MtL), Meta-Features are measures used to describe datasets and their relationship with algorithm bias. Meta-Features are used in Meta-Learning and AutoML tasks to describe the underlying dataset to create a machine learning, recommender systems and other models. The Meta-Features can be categorised to various groups, namely: General, Statistical, Information-theoretic, Model-based, landmarking, Relative Landmarking, Clustering, Complexity, etc. (9) (10) (17). The most frequently adopted meta-features in three of the important categories used in this research are:

- **General Meta-features**: Number of observations, Number of attributes, Number of output values, Dataset dimensionality.
- **Statistical Meta-features**: Standard deviation, Coefficient of variation, Covariance, Linear correlation coefficient, Skewness, Kurtosis
- **Information-Theoretic Meta-features**: Normalized class entropy, Normalized attribute entropy, Joint entropy of class and attribute, Mutual information of class and attribute, Equivalent number of attributes, Noise-signal ratio.

The Meta-Features discussed above are just a small sample of various possible characterisation features (9) (10) (17) , this is not an exhaustive list and details about these features are out of scope of this research.

2.5 Related Work

2.5.1 Federated Meta-Learning

Chen in (18) has used the term **Federated Meta-Learning** once before, but in a completely different context. The terms, Distributed Machine Learning (19) and Federated Machine Learning (20) are related to Federated Meta-Learning, in the sense that they share a few underlying concepts but yet totally different.

Distributed (Machine) Learning

Automatic Learning has become popular and important in the recent years due to exponential growth of data being collected and the need to analyse data to create machine learning models. Peteiro in his survey of methods for Distributed Machine Learning (19), points out that these needs introduced a new type of challenge with respect to efficiency and scalability of machine learning algorithms related to computational and memory resources. To solve this challenge researchers proposed two approaches, one is the concept of distributed machine learning algorithm which posses various limitations and the other strategy is to combine the outputs of various algorithms in a fashion similar to ensembling. On doing so large datasets can be addressed using a distributed clustering environment where both the techniques are used in combination to attain best possible results.

Federated Learning

Federated Machine Learning (20) is a collaborative learning strategy where a machine learning algorithm is trained across multiple devices which hold the data without exchanging it, i.e, the users remain the owners of the data. This approach is different to traditional approach of having a centralised repositories of data similar to OpenML (21) and also the approach of distributed machine learning, it is assumed that the data is identically distributed across devices. Federated Learning allows the end users to build a common, robust model across devices without sharing data, thus addressing important issues regarding data security and privacy.

Federated Learning vs Federated Meta-Learning

Federated Machine Learning enables devices to collaboratively learn a shared prediction model while keeping all the training data on device, decoupling the ability to do machine learning from the need to store the data in a centralised server. Whereas Federated Meta-Learning provides an ecosystem where the learning happens locally on the users machine based on the dataset's meta-data that the user provides. This meta-data is used to make a recommendation of an algorithm that is used to create a model. Here, both the learning and data is kept on the users machine.

2.5.2 Other Related Concepts and Research

Algorithm Selection

The field of algorithm selection and configuration is an area of intense research, according to Wikipedia, as stated by Rice in (22), the process of algorithm selection is defined as:

Definition 1: Given a portfolio $\mathcal{P}$ of algorithms $\mathcal{A} \in \mathcal{P}$, a set of instances $i \in \mathcal{I}$ and a cost metric $m : \mathcal{P} \times \mathcal{I} \to \mathcal{R}$, the algorithm selection problem consists of finding a mapping $s : \mathcal{I} \to \mathcal{P}$ from instances $\mathcal{I}$ to algorithms $\mathcal{P}$ such that the cost $\sum_{i \in \mathcal{I}} m(s(i), i)$ across all instances is optimized.

Rice, went into great detail about explaining about what he thought were the four major criteria in the selection process. The four primary criteria are as follows: Best Selection, Best Selection for a Subclass of Problems, Best Selection from a Subclass of Mappings and Best Selection from a Subclass of Mappings and Problems. He also proposed five major steps for analysis and solution of the algorithm selection process. The steps being: Formulation, Existence, Uniqueness, Characterization and Computation. He also explained in great detail about these criteria and steps using various example problem and formulating the best algorithm selection process for these criteria along with models. By doing this Rice in (22) proposed 15 questions that he suggested to be asked before an algorithm is chosen.

In (23), Kerschke and others talks about algorithm selection problem by giving importance to *per-instance algorithm selection problem* that is; given a computational problem, a set of algorithms for it, and a specific instance that needs to be solved, the problem is to determine which algorithm(s) can be expected to perform best on that instance. The authors also relates this type of problem to per-set algorithm selection, algorithm configuration, algorithm schedules and parallel algorithm portfolios and discusses about them in detail. By doing so in (23), the authors also compare the results, discuss the problems faced and propose solutions for algorithm selection in discrete and continuous problems. They also provide an informative overview of problem specific feature set about which they discuss and provide a strong basis of why these characteristics are used in algorithm selection. They also shed some light on various other applications and contributions based on their impact in this ever growing and evolving field of algorithm selection.

The propositional satisfiability problem or SAT is an NP-Complete problem, and in (24) the authors propose an automated approach for constructing per-instance algorithm portfolios for SAT and proposed a online platform called SATzilla. Here the authors propose a new approach to algorithm selection based on the idea of building an approximate run-time predictor compared to the previous "winner-takes-all" approach. In the winner-takes-all approach where the algorithms run-time is measured on a representative set of the problem and which ever algorithm performs better takes the top spot. The approximate run-time predictor is an heuristic approximation to the perfect solution and the authors built an empirical hardness model, which is a computationally inexpensive predictor of an algorithm's run-time and it based on features of the instance and past performance of the algorithm.

ASLib (1) is a benchmark library for algorithm selection, which focuses on (not a limitation to) constraint satisfaction problems. The authors introduced 12 algorithm selection scenarios from six different areas, discussed the formats and showed examples for automated exploratory data analysis that will run for each new scenario which has been submitted to their ASLib platform. Their platform also facilitates algorithm selection methods by providing a common set of benchmarks and tools. The authors built on top of Rice's (22) formalisation of algorithm selection and also took a different approach than that of SATzilla (24). SATzilla approach was to select a single algorithm for solving the problem instance, ASLib, tries to find a schedule where ordering and time budget where all or a subset of the all the algorithms can be executed in a to reflect the expected performance of the given algorithm.

AutoML

Though the above mentioned researchers laid the foundation for algorithm selection, these researches either discussed about the problem in general or focused on one particular field like the propositional satisfiability or SAT problem(s). Due to the growing demand in commercial use of Machine Learning, various enterprises have aimed to satisfy the AutoML problem, this has lead to the increase in research and development of tools which help novices to use machine learning without the expertise required.

According to (8) and (11), new methods for increasing efficiency and robustness of AutoML are the current trend and focus in this area of research. The authors proposed tools like Auto-Sklearn and Auto-WEKA respectively to solve the problem of algorithm selection by improving on the existing AutoML methods. In Auto-Sklearn (8), the authors made improvements to the AutoML approach by introducing Meta-learning, in which they used to find good instantiations of machine learning algorithms, it is a complementary approach to that of Bayesian optimization techniques. In another approach the authors of Auto-Sklearn used automated ensemble construction, where they used the models created during the evaluation process, instead of discarding these models they used the models in a post-processing technique to create an ensemble which is evaluated during optimization and from there results they found that the ensemble almost always outperformed individual models.

In Auto-WEKA (11), the authors built on the Bayesian Optimisation approach. They used the Tree-structure Parzen Estimator (TPE) approach which is a Sequential Model-Based Optimization (SMBO) algorithm, they also used Sequential model-based algorithm configuration (SMAC) models, used to create probabilistic models. These two types of models gave robust performance for algorithm selection and probabilistic estimators for hyper-parameters respectively, which were used to demonstrate the feasibility of an automatic approach to learning algorithm and hyper-parameter selection. The result of this research was a tool called Auto-WEKA, which has a list of 47 WEKA classification algorithms that were used as a single learning algorithm, which selects a single base classifier and a meta- or ensemble-classifiers. It was one of the first tools to perform a fully automated algorithm selection and hyper-parameter optimization for a large set of candidates.

Meta-Features

Rivolli in (17) surveys a comprehensive list of meta-features and how they are used in classification problems, they authors also analyse and organise the meta-features by highlighting their positive and negative attributes of each meta-feature. They also defined what a meta-feature is, and it is as follows:

Definition 2: Let $D \in \mathcal{D}$ be a Dataset, $m : \mathcal{D} \rightarrow \mathcal{R}^{k'}$ be a characterisation measure, and $\sigma : \mathcal{R}^{k'} \rightarrow \mathcal{R}^{k}$ be a summarisation function. Both m and σ have also hyper-parameters associated, h_m and h_σ respectively. Thus, a meta-feature $f : \mathcal{D} \rightarrow \mathcal{R}^{k}$ for a given dataset $\mathcal{D}$ is given by:

$$f(\mathcal{D}) = \sigma(m(\mathcal{D}, h_m), h_\sigma)$$

The measure m can extract more than one value from each data set, i.e., k' can vary according to $\mathcal{D}$, which can be mapped to a vector of fixed length k using a summarisation function σ.

The authors also present a tool called Meta-Feature Extractor, which can be used to measure the meta-features proposed and discussed in this paper and made the tool publicly available to the users as a package in Python and R.

The authors in (9) and (10) introduce concepts about meta-features characterisation and selection algorithm respectively. Castiello in (9) analyses most commonly used meta-features and discusses their properties in an inherent manner. He also introduces new features by transforming the existing features as a result of their analysis. The author(s) also suggest a set of measures that can be used to describe a dataset using the proposed meta-features. The suggested meta-features can be used as a bias for other base-learning in for their respective tasks.

Filchenkov in (10) creates an optimal meta-feature suggestion algorithm for different cases. The authors do this by creating a new approach for meta-feature engineering, which has been proved useful by them for feature selection algorithm's recommendation. They also conduct an almost complete analysis of most of the popular meta-features and use the results so obtained to suggest optimal meta-features for different tasks. One of the major contributions by this paper is the analysis they have done in relation to classification algorithms. They have used various popular classification algorithms for which the author(s) engineered new meta-features and recommended existing meta-features for most of the popular algorithms and they categorised this specific type of measures as model-based meta-features.

3 Design and Implementation

This chapter provides details regarding the FMLearn application, implementation of the proposed concept, that is, Federated meta Learning and it's working mechanisms. Each component of the application is discussed in detail including the components that were considered and tested, but are not a part of the final experimentation. Explanation of all the design decisions are provided in detail so as it make it easier for the reader to understand how the application developed over-time and reached it's final state. The FMLearn application attempts to solve the issue of redundant work put in by developers to find the best algorithm and it's hyper-parameters, repeatedly for a previously solved / optimised dataset, and possibly suggests algorithms for a previously unseen dataset as well. If this is achieved, the users of this application and the community in general will be greatly benefited by saving time spent waiting for program to complete, saving electricity consumed on the running the machines and cooling them, the computational power expended over a repetitive task and money spent in all of the above is saved in finding the best algorithm and it's hyper-parameter.

The design and implementation choices led to the development of a working prototype of the concept Federated Meta-Learning and the application FMLearn. The server was developed in python and is used to portray the capabilities of the application. The concept of Federated Meta-Learning discussed in 3.1 and the changes in the workflow of a typical Machine Learning project caused due to the introduction of Federated Meta-Learning is discussed in section 3.2. Then we discuss in detail the prototype FMLearn which was built to demonstrate this concept under section 3.3, while discussing it's architecture design in section 3.3.2 along with some design decisions in section 3.3.1 and why these were taken. We will also be discussing in detail about the Model built by FMLearn to predict / recommend the potentially best algorithm(s) for a given task along with it's workflow. In section 3.4 we will be discussing in detail about the modifications made to scikit-learn so that it can act as a client to FMLearn along with discussing the data description used.

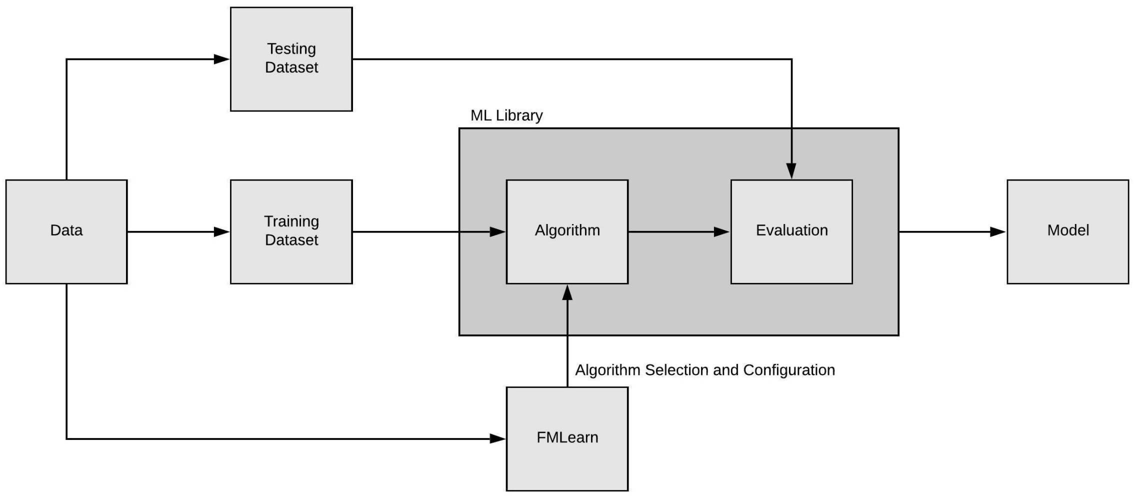


Figure 3.1: Federated Meta-Learning Workflow Diagram

3.1 Federated Meta-Learning

Federated Meta-Learning focuses on learning algorithm performance measures for arbitrary tasks. Essentially federated meta learning is an ecosystem where the raw data is kept on the original devices and the meta data and performance metrics of the algorithm on the tasks would be stored on a central FML server. Using this historic performance data, predict the best performing algorithm along with its hyper-parameters for a previously seen or unseen task.

The input to Federated Meta-Learning is a description of the task, and the output is a recommendation for the potentially best performing algorithm(s) to solve that task. This recommendation could consist simply of a list of the best algorithms, or their predicted performance values. The list could also consist of multiple sub-lists created with different meta-learners.

In its simplest form, federated meta-learning would simply be a knowledge base or directory of algorithm-dataset performance measures. Ultimately, Federated Meta-Learning would be able to predict algorithm performance for unseen tasks.

3.2 Workflow of Federated Meta-Learning

From section 2.3, we know that a typical algorithm selection and configuration task using the AutoML tools result in a repetitive and time consuming process which in turn eats up a lot of computing resources, electricity and money. To tackle this problem the concept of Federated Meta-Learning as seen in section 3.1 can be applied. When we take this concept and apply it to the current ecosystem of Machine Learning tasks we get a workflow as show in Figure 3.1.

Let's go into detail about the workflow of Federated Meta-Learning, Initial steps of the workflow remain similar to the working of AutoML (as seen in section 2.3). The data is split into training and testing datasets, the training dataset in first sent to the library for the purpose of model creation. Here, the dataset is pre-processed using various techniques like data-cleaning, data imputation, data encoding, feature selection, feature exploration, feature engineering, feature co-relation, etc.

The difference between the 2 processes arise at the point when the data is obtained. The data apart from being sent to the library for pre-processing it is also sent to the Federated Meta-Learning application after finding the meta-features of the dataset. Once these meta-features are obtained Federated Meta-Learning predicts / recommends the potentially best performing algorithm(s) along with it's hyper-parameters for the given dataset. Once this recommendation is received by the Machine Learning library it re-optimises the algorithms and builds the model based on the training dataset which was received after pre-processing and this model is then evaluated against the testing dataset and then the final model is returned to the user.

The major difference with the workflows described in Figure 2.1 (as seen in the Section 2.3 - Workflow of AutoML Libraries) and Figure 3.1 is the introduction of an external application which applies the concept of Federated Meta-Learning. This application results in the elimination of the repetitive nature of algorithm selection and configuration task, which would drastically reduce the time spent on such a task along with the computational resources, electricity and money spent here.

3.3 FMLearn

FMLearn is an application which is a simple proof of concept of Federated Meta-Learning. FMLearn allows everyone to benefit from the data that is generated through machine learning and data science libraries. FMLearn is built using the Distributed Application Architecture (DAA), following the Client-Server Model. This allows the users to access as well as to exchange information and services with others. Though a Peer-To-Peer Model would be possible, this design was avoided due to the limitation in time as well as favouring the ease of development using the client-server model.

FMLearn consists of a client which in our case a modified version of the popular Machine Learning library Scikit-Learn, but it could be any machine learning library. The Server is a Python Flask Application which handles all API calls and Data Store requests made by the user. FMLearn also acts as a knowledge base, storing all the algorithms-data performance measure collected though the machine learning library. The server also provides publicly available

API's which can be accessed by any Machine Learning or Data Science tool to use FMLearn irrespective of the programming language used to build them to get a recommendation for the potentially best performing algorithm(s) and it's hyper-parameters to solve it's task without being constrained by the need to use the client that is supported by this paper (modified scikit-learn).

3.3.1 Design Decisions

A few important Design Decisions that needs to be addressed before proceeding with further discussions are as follows:

- Why Client-Server Model?

  The biggest design decision was taken in the early stages of the dissertation. A decision was taken to proceed with the Client-Server model rather than a Peer-To-Peer model with respect to the architecture design of the application. This was an "Experience Based Design" as I have prior experience working with Client-Server Application model and more importantly due to the limited availability of time and ease of development which cannot be achieved when a Peer-To-Peer architecture is followed.

- Why Scikit-Learn?

  Scikit-learn is a free software machine learning library for the Python programming language (25). It is also among the popular and easy to use python libraries available in the market. Though other libraries with similar capabilities are available in the internet scikit-learn was chosen as an "Intuition Based Design" due to familiarity with the library and it's easy to use nature.

- Why a Public API Server?

  The concept of Federated Meta-Learning (15), was envisioned as an ecosystem where everyone would benefit from the data that is generated though Machine Learning libraries and the prediction of the best performing algorithm along with it's hyper-parameters be available to all. A public API server ensures that even if the developers aren't using the supported client - modified scikit-learn - they can still benefit from the recommendations made by FMLearn.

- Why not a stand-alone client?

  A stand alone client was a possible aim for this dissertation, as said the client doesn't highly depend on the features provided by scikit-learn, though a few dependencies exists. But this idea was dropped due to limited availability of time to concentrate on building and improving the server where the recommendations were to be made.

- Why Flask?

  Flask is a popular, extensible web micro-framework for building web applications with Python (26) and is among the most used web applications frameworks in python. It was a choice between Django and Flask, and Flask was chosen because of ease of use in terms of quick development when compared to Django. This was a "Reference Based Design" decision and when trying out both frameworks it was easier to get things started with Flask as opposed to Django. It is also easy to deploy Flask on to a free hosting services like Heroku, for the application to go live.

- Why PostgreSQL?

  PostgreSQL is a free and open-source relational database management system. It is easy to use when compared to other options and most importantly as it is a widely used database it was also available as a database deployment option in various online hosting platforms such as Heroku.

These are among the major design decisions made in the early stages of the dissertation which needed to be addressed before proceeding. Though a few other design decisions were made during the process of development they will be explained as and when the appropriate section of the application is discussed.

3.3.2 Architecture Design

The figure 3.2 describes the client-server architecture of the FMLearn application as a whole, along with a few other components that haven't been discussed yet, these components will be introduced here and will be expanded upon later sections of this report.

From the Figure 3.2 we can see that the entire application is divided into two major chunks the client and the server. The client here is the modified version of Scikit-Learn which is explained in Section 3.4 in detail. To which I have introduced an additional package called '`fmlearn`'. This package provides the user with the capability to interact with the server. This package also functions as an important link to the intermediary step where the data description is obtained for a given dataset or task i.e., the dataset is converted into it's meta-features. The conversion of dataset to it's Meta-Features is handled by a 3rd party library called Auto-Sklearn (8). Auto-Sklearn is an external library, a dependency is introduced here, details about it will be further explained in Section 3.4.1.

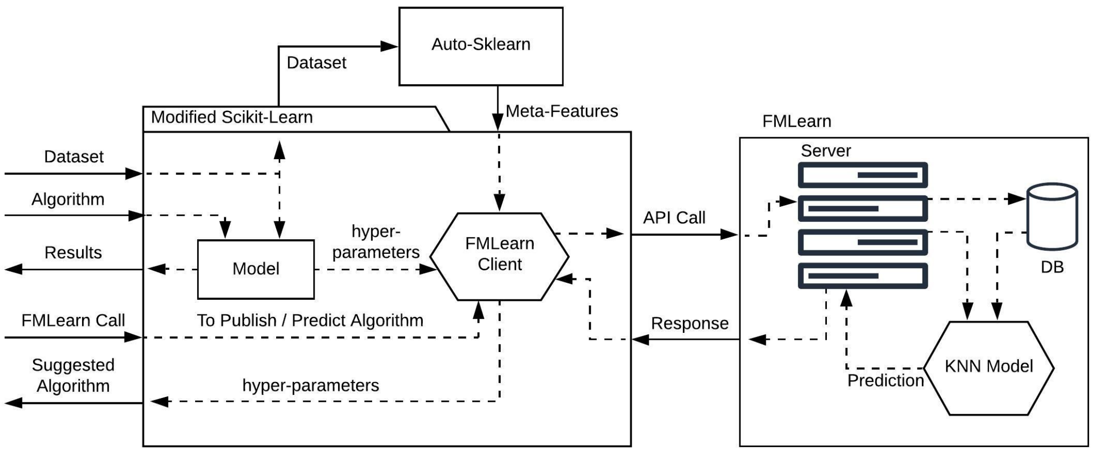


Figure 3.2: Architecture Diagram

FMLearn Client

The client majorly performs 2 tasks, publishing the data to the server and displaying the results obtained for algorithm recommendation from the server to the user. Irrespective of the task, the `fmlearn` module obtains the dataset from the user and it is converted to it's Meta-Features with the help of Auto-Sklearn. Depending on the task the user might want a recommendation of the best performing algorithm or might want to contribute his findings to the betterment of the FMLearn in general. Depending on the use case either an API call is made to obtain the recommendation for the user or the model details along with the performance metrics which is obtained from the user and the model directly, then an API call is made to publish these details to the server respectively.

FMLearn Server

The server performs various roles depending on the task and is explained in Section 3.5 in detail. The server primarily acts a recommender system, recommending algorithm(s) for a given task. The server also acts a knowledge base or directory of algorithm-dataset performance measures, apart from this is also provides API's to expand or build this knowledge base. This knowledge base is stored in a PostgreSQL Database and is also used to build a Machine Learning model. The model was built using the K-Nearest Neighbors algorithm. The details about the model and algorithm in general will be discussed in the Section 3.5.1. This model is used to make recommendations to the users for previously seen or unseen task or dataset. The model's ability to recommend the best performing algorithm(s) have been exposed to the public via API's.

3.4 The Client: modified Scikit-Learn

In this Client-Server Architecture, the stable release of the client i.e., modified version of scikit-learn is available on GitHub via the following link:

`https://github.com/mukeshmk/scikit-learn/releases/latest`.

The complete code repository is available at: `https://github.com/mukeshmk/scikit-learn/` which is forked from: `https://github.com/scikit-learn/scikit-learn`.

After downloading the stable release and installing the library by following the instructions specified in the `README.md` file, we can use it normally as we use scikit-learn, since it's a fork of the original release it has all the features of the stable release plus it has the features of FMLearn.

Initialisation of FMLearn

To initialise `fmlearn`'s client we have to import the `FMLClient` package from `sklearn.fmlearn` and then create an object of the class `FMLClient` as follows:

```python
# imports the FMLearn client into the program
from sklearn.fmlearn import FMLClient
# initialises the client
fmlearn = FMLClient()
```

Obtaining the Meta-Features

Once the client as been initialised irrespective of the task the first thing to do would be do tell `FMLearn` what dataset we will be using for the task this can be done as follows:

```python
# assuming that the import for train_test_split has been done
# and the data has been loaded and split into 'X' and 'y'
x_train, x_test, y_train, y_test = train_test_split(X, y, test_size=0.2,
    random_state=12)

# this is a function introduced by the Client to let FMLearn know
# what dataset is being used for the task.
fmlearn.set_dataset(x_train, y_train, x_test, y_test)
```

Setting the dataset using the `sets_dataset()` method triggers a background call to an external 3rd party library called Auto-Sklearn which is used to describe the Dataset and obtain the Meta-Features for that dataset. This is further explained in Section 3.4.1. These Meta-Features thus collected is then sent to the server to be processed as required.

Available Methods

Depending on the objective of the user, the user can proceed with his program in one of these two ways:

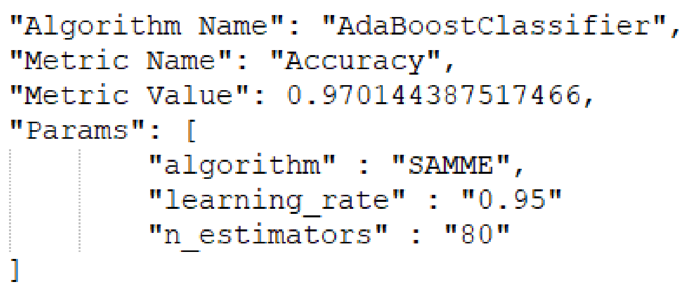
1. Get recommendation for a task.
2. Publish performance data for a task.

Recommendation for a Task

If the user wants to get a recommendation for a task all they would have to do is to call the method `predict_metric()` as follows:

```
# this method returns the recommendation of the
# best performing algorithm along with its hyper-parameters
# for the dataset set using the set_dataset() method
fmlearn.predict_metric()
```

This method provided by the client internally makes an API call to the server, sending the server the Meta-Features of the dataset which was obtained from `auto-sklearn`. Upon reaching the server, the server processes the request, responding with the appropriate algorithm recommendation which is then sent to the user as shown in Figure 3.3. The recommendation can then be used by the user to create a model with the specified algorithm and the hyper-parameter values. The created model can then be used by user to proceed with the task after re-optimising the parameters to suite the users dataset.



```
"Algorithm Name": "AdaBoostClassifier",
"Metric Name": "Accuracy",
"Metric Value": 0.970144387517466,
"Params": [
        "algorithm" : "SAMME",
        "learning_rate" : "0.95"
        "n_estimators" : "80"
]
```

Figure 3.3: Sample Output

Publish Metrics for a Task

If the user wants to publish a performance metric for a task which they performed, the user can do so by using the `publish()` method by passing the model used, scoring metric and the score as it's parameters as follows:

```
# the ‘model‘ parameter contains the model trained using the dataset
# ‘fmlearn.ACCURACY‘ specifies the evaluation metric used
# the ‘score‘ parameter contains the evaluation results.
fmlearn.publish(model, fmlearn.ACCURACY, score)
```

Upon calling the client's `publish` method the meta-features of the dataset, model details including the algorithm name, hyper-parameters, along with the evaluation metric and the evaluation scores are sent to the FMLearn server. This data is processed and stored in the database, which is later be used to build a model.

3.4.1 Data Description: Auto-Sklearn

As discussed in section 2.4, Meta-Features describe the dataset, and there are various dedicated tools which provide the required features. One such tool is Auto-Sklearn (8). It is an open-source AutoML tool written in Python that automatically determines effective machine learning pipelines for classification and regression datasets. Though this is an AutoML tool, it was used here for it's **meta-learning** step in the AutoML pipeline, which it uses to warm start the Bayesian optimization procedure in it's core. This meta-learning step. i.e., obtaining meta-features is abstracted inside all the features provided by the tool and is not easily available to the users to use outside the tool.

Meta-features provided by Auto-Sklearn fit the following meta-features categories: General, Statistical and Information-Theoretic meta-features, though a few meta-features exists outside these categories, but majority of them fall into these. An utility package was introduced inside the FMLearn's client (modified Scikit-Learn) to make use of the functionality provided by auto-sklearn to obtain the meta-feature of a dataset. This utility function performs the input checks, transforms the data and performs the required validations before passing on the dataframe to Auto-Sklearn's calculate meta-features method. It is available in this package via the imported method.

```
from autosklearn.metalearning.metafeatures.metafeatures
import calculate_all_metafeatures_with_labels
```

The input to the `calculate_all_metafeatures_with_labels()` method is the dataset and the output is a set of meta-features. These meta-features are then converted into the required format and then forwarded to server via API calls. A few of the meta-features thus obtained are: `ClassEntropy`, `ClassProbabilityMean`, `NumberOfMissingValues`, `NumberOfCategoricalFeatures`, `NumberOfNumericFeatures`, `etc`. Depending on the dataset there can be about 24-30 meta-features obtained for a given dataset. Refer appendix A1.3, for the complete list of meta-features and to see a an example of values obtained via the Auto-Sklearn.

The end user is abstracted away from existence of this feature, the call to auto-sklearn is handled internally by the client upon letting the client know about the dataset in use.

```
# this is a method provided by the FMLearn client that internally calls
# ‘calculate_all_metafeatures_with_labels()‘ method provided by auto-sklearn
# which provides the meta-features required.
fmlearn.set_dataset(x_train, y_train, x_test, y_test)
```

Design Decision

Before finalising with `auto-sklearn` another library by Hadi called `dataset2vec` (27) was explored and evaluated for it's effectiveness in model building. This library represents a tabular dataset in a hierarchical fashion by defining a dataset as a set of features, where each feature is a set of instance values. Working with this library proved to be quite challenging due to the lack of documentation and lack of pre-trained model which converts the dataset to it's meta-features and thus making it necessary for the user to train a model. Though this paper claimed to out-perform the current state-of-the-art, a design decision was made to not use `dataset2vec` and instead proceed with `auto-sklearn`, as a model was available and also a support by the community to improve the meta-feature extraction model.

3.5 The Server: FMLearn

The stable release of the server - FMLearn in this Client-Server Architecture is available on GitHub via the following link:

`https://github.com/mukeshmk/fm-learn/releases/latest.`

The complete code repository is availabe at: https://github.com/mukeshmk/fm-learn. The application is also deployed on Heroku. Heroku is a platform as a service (PaaS) that enables FMLearn to available on the cloud to be accessed by everyone. The application is deployed and available via the link mentioned below, though this is an API server a basic user-interface has been developed so as to provide information about Federated Meta-Learning and API documentation.

https://fmlearn.herokuapp.com/.

API's Provided

The server provides several API's among which two of the important API's are the `publish` and `predict` API's. These API's represent the two important functionalities of the proposed prototype.

Publish API

The `publish` API solves the problem with the workflow of machine learning and data science libraries. These libraries usually work in isolation and the information regarding how an algorithm performs on a particular dataset is neither publisher nor shared.

```
# the input JSON object to publish API
{
   "algorithm_name": algorithm_name,
   "dataset_hash": dataset_hash,
   "meta_features": [
      {
         "feat_name": feature_name,
         "feat_value": feature_value
      }
   ],
   "metric_name": metric_name,
   "metric_value": metric_value,
   "params": [
      {
         "param_name": hyper_parameter_name,
         "param_value": hyper_parameter_value
      }
   ],
   "target_type": target_type
}
# NOTE: "meta_features and params" are a list of objects
# but for illustration purpose only 1 item has been shown.
```

The publish API can be called from within these libraries as done in the modified scikit-learn presented and thus enabling the users to share the algorithm performance data. The API is a `POST` method which can be accessed via this endpoint `/metric` and takes a JSON object as input in the format as described above.

The Underlying Database

The input data is validated, preprocessed and stored in the database. A PostgreSQL database used in this application which contains 3 tables to store the information received from the client. These tables are structured as follows as shown in Figure 3.4. The three tables are `Metric`, `Meta Feature` and `Params`, the `Metric` table has a one to many relation ship with both the `Meta Feature` and `Params` tables and this relationship puts the database in it's 3rd Normal Form.

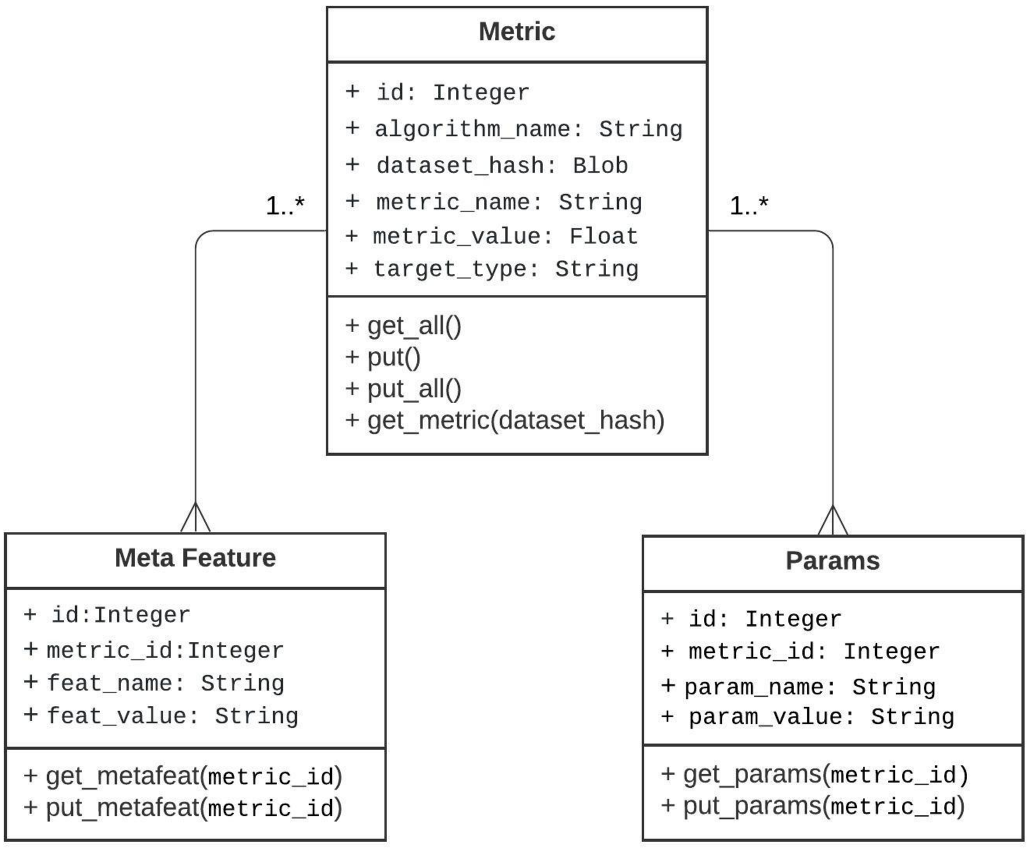


Figure 3.4: Class Diagram

This database thus created acts as a knowledge base which contains information about the algorithm-data performance metrics. FMLearn also provides others API's to access this data which enables sharing of performance details of various algorithms among software libraries. The response of this API when there are no internal errors is a JSON object similar to the request object with a HTTP status code of 200.

Predict API

The `predict` API solves the problem with the repetitive nature of the workflow of machine learning libraries, where the algorithm selection and hyper-parameter optimization tasks are to be performed repeatedly. The `predict` API can be called by the machine learning libraries, as done in the modified scikit-learn presented in this paper, and thus breaking the cycle and saving time, electricity, computation power and money. The API is a `GET` method which can be accessed via this endpoint `/metric/predict` and the API accepts a JSON object as input which is similar to the below format:

```
# the input JSON object to predict API
{
   "dataset_hash": dataset_hash,
   "meta_features": [
      {
         "feat_name": feature_name,
         "feat_value": feature_value
      }
   ],
   "target_type": target_type
}
# NOTE: the key "meta_features" accepts a list of objects
# but for illustration purpose only 1 item has been shown.
```

This input is validated, preprocessed and then sent to the model for the best algorithm prediction (this is discussed in detail in Section 3.5.1) and the result of the prediction is sent to the user as the best algorithm recommendation for the given task. This response is a JSON object in the following format:

```
# the output JSON object of predict API
{
  "recommendations": [
    {
      "algorithm_name": algorithm_name,
      "metric_name": metric_name,
      "metric_value": metric_value,
      "params": [
        {
          "param_name": hyper_parameter_name,
          "param_value": hyper_parameter_value
        }
      ],
      "target_type": target_type
    }
  ]
}
# NOTE: "recommendations and params" are a list of objects
# but for illustration purpose only 1 item has been shown.
```

3.5.1 KNN Algorithm

The goal is to use FMLearn to try and find the closest existing dataset which is very similar to the dataset for which an algorithm is to be recommended. Then recommending best performing algorithm for the requested dataset is based on the closest existing dataset. FMLearn doesn't try to predict the best performing algorithm rather tries to predict the closest dataset, this is an important point to be noted in the algorithm recommendation process. Prediction of an algorithm along with their hyper-parameters using a machine learning model might not be feasible as there are various types of machine learning problems like classification, regression and clustering just to name a few. These problem types employ various algorithms within each type, which have varying performances and the resulting model will be a multi-class output problem. This problem will be to try and predict algorithm name, metric used, metric value, and not to mention the parameters of an algorithm. This will result in a lot of mismatch with the hyper-parameter suggestion and the output recommended will just be a random set of values. This might become feasible given the existence of a huge dataset and a complicated neural network, but gathering this data is out of scope of this dissertation. Here, in this dissertation we are building the first proof of concept for Federated Meta-Learning. Thus predicting the closest existing dataset for the input dataset and then finding the best algorithm for that dataset is the best possible approach in the given scenario.

The algorithm used to make prediction or recommendation of the best performing algorithm for a given task is the **K-Nearest Neighbours (KNN)** algorithm. KNN is a supervised, non parametric learning algorithm, meaning, the target variable is known and that it does not make an assumption about the underlying data distribution pattern. In the case of FMLearn, a KNN classification algorithm has been used with k=1 and the distance between points is calculated using Euclidean distance. The "K" in KNN is the number of nearest neighbors to which the given input point must be the closest to before labeling the input point. In terms of FMLearn, "K" represents the number of datasets that the input dataset has to be closest to. Since we want to find one dataset to which the input dataset is closest to we are setting the value of "k=1".

Design Decision

To guarantee reliable recommendations of algorithms based on historic performances of algorithms-data pairs, K-Nearest Neighbours algorithm and a Support Vector Machine were ideal choices to being with. Both KNN and SVM were tested with FMLearn. Though in it's current state FMLearn has a very small dataset with a low dimensionality or small number of features, both algorithms perform equally well. A decision was made to go with KNN over SVM because on the long run, when the number of instances or points in the dataset increase KNN begins to outperform in a low-dimensional space, whereas as SVM might face difficulty is finding a linear separation in the dataset when the number of points increase. Though KNN is very sensitive to bad features and outliers, in this case we know that all the features being used to build the model are required to describe the dataset and are obtained from auto-sklearn and as for the case of outliers, since K value being used in KNN is one, outliers will not be a major factor and will not affect predictions or algorithm recommendations. Whereas, in the case of SVM, which is generally considered to be a better performer than KNN, the drawbacks arise when the number of points increase, even if a smart kernel is used, the number of output classes that get created increases as the number of new datasets or variations of datasets seen by FMLearn increase. This puts SVM in a huge disadvantage as the need arises to create separating planes for each class. Thus for these reasons KNN was chosen over SVM as the algorithm of choice for recommending the best performing algorithm for the user.

Algorithm Recommendations

Recommending the best algorithm falls into 3 different possibilities based on the closes of the input dataset, which can be categorised as follows:

- Previously known dataset.
- Previously unknown dataset, which is similar to a known dataset.
- Previously unknown dataset, which is dissimilar to a known dataset.

In the case of a previously known dataset for which the best algorithm is also known, the distance between the points is 0 units, thus making the predictions 100% accurate. Whereas, for previously unknown dataset which is highly similar to known dataset the prediction of algorithm is accurate but the hyper-parameters require re-optimisation to suite the new dataset. But, in the case of an unknown dataset which is highly dissimilar to a known dataset, the model is able to predict the type machine learning problem the dataset belongs to, but not the best algorithm for it, instead it recommends a set of algorithms which it thinks are the best suited in this case instead of recommending a single algorithm.

3.5.2 Pre-Processing and Model Building

The data-frame which is the sent to the KNN Algorithm consists of the following features: meta-features and target type as input and the output is the dataset hash. The meta-features consists of a maximum of 29 features, if the feature doesn't exist of the dataset, the missing value is imputed with -1. The other pre-processing done are as follows the meta-feature values are scaled between 0 and 1, the feature "Target Type" is one hot encoded and the output variable "Dataset Hash" is label encoded. Since we are only trying to find the closest dataset the repeated datasets are removed based on the dataset hash, thus retaining the variations of the same dataset. The pre-processed dataset is sent to model for training and later used for prediction of the dataset. Based on the predicted dataset, FMLearn suggests the best performing algorithm to the user.

The model training and re-training is an automated process in FMLearn. The model is retained when a trigger from the database is sent to the application. This trigger is set off, after every 10 new records that are added to the database (this number is configurable though the `config file`). This enables FMLearn to keep improving it's model and it's predictions as and when new data is published to it, thus improving the recommendations.

3.6 FMLearn Workflow

The Figure 3.5 represents the Sequence Diagram of the application, explaining the object interactions in a time sequences. The sequence diagram of FMLearn represents 2 major use cases. Each use case represents the flow of object interactions in the case of where the user wants to:

- publish data to FMLearn
- get a recommendation from FMLearn for a given dataset.

Though the working of each individual modules/objects have been discussed in previous sections of this report, this section focuses explicitly on giving a in-depth understanding of how all the individual modules work together to form FMLearn.

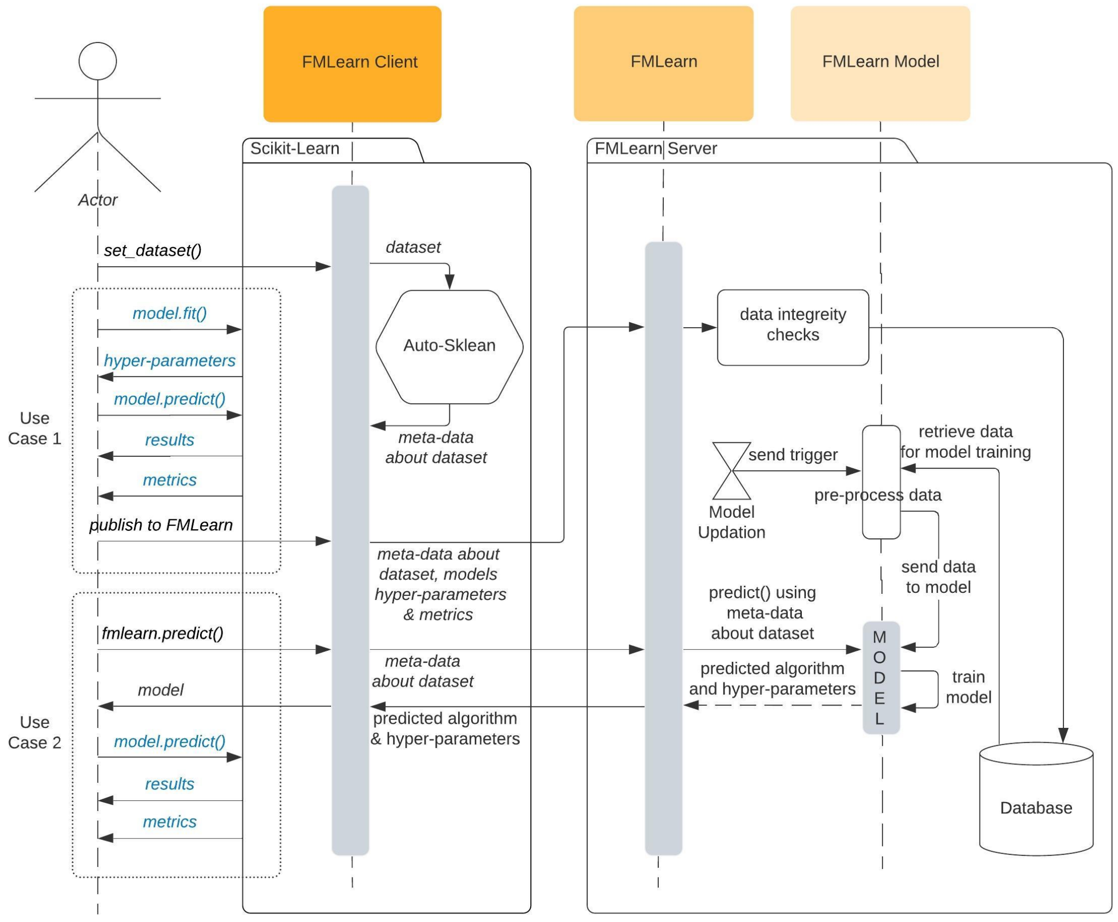


Figure 3.5: Sequence Diagram

Use Case 1: Publish Metric

The use case 1 describes the scenario where the user wishes to publish their algorithm-data metrics to the FMLearn application. The first step of this process is to let FMLearn know about the underlying dataset by using the method `set_dataset()`. As explained in section 3.4.1, it makes an internal call to `Auto Sklearn` to find the meta-data / meta-features about the dataset. These meta-features are stored in the FMLearn client object created and are later used when publishing. Then the user performs his preprocessing tasks, finds the best algorithm along with it's hyper-parameters and uses the model so thus created to evaluate the results using the `model.fit()` and `model.predict()` methods for the created model objects the results. This model, along with it's results is sent to FMLearn's client via the `publish()` method, where model details like the algorithm used and hyper-parameters are extracted and a request object is created with them along with the meta-features obtained from auto-sklearn and the algorithm metrics. This request object is a JSON object which is used to make an API post call to the FMLearn server. Once the server receives the request object it's pre-processed and checked for the data integrity and then pushed into the database and a `200 OK` response is sent to the client.

Use Case 2: Predict Metric(s)

The second use case characterises, the scenario where the user wishes to get a recommendation from the FMLearn application for the best performing algorithm for his dataset. The first step here as well is to let FMLearn know about the underlying dataset by using the method `set_dataset()`, this obtains the meta-features of the dataset and stores them in the client object. The next step is to make a call to the client to get the recommendation using the `predict()` method. This creates a JSON request object which contains the meta-features which is sent across by the client to the FMLearn server via an API call as discussed in section 3.5. Upon being received by the server, the request object is pre-processed and checked for data integrity. The pre-processed data is sent to the model built using KNN algorithm (discussed in section 3.5.1) to make predictions about the closest dataset available, upon finding the closest dataset, the best performing algorithm for the predicted dataset along with it's hyper-parameters are retrieved from the database and an response JSON object is created and sent to the client. The client then recommends the received algorithm to the user, which the user then further re-optimises if required and then proceed with his task.

3.7 Security and Privacy Concerns

FMLearn is a client-server architecture-based (28) application which provides public API's for it's users, and it brings its own security and privacy concerns. The different aspects of security and privacy issues that are to be considered with respect to FMLearn are - but not limited to:

1. A client-server architecture.
2. Publicly available API server (29) (which could be broken down)

3.7.1 Security Concerns

FMLearn was built using Python Flask, and uses PostgreSQL, which acts as the public API server which exposes various API to users to use the application and is currently hosted on Heroku .

One of the most important security concerns with the current implementation of FMLearn is that the users have unlimited access to APIs (Flash Crowd Problem) (30), this could have severe consequences. A denial of service is possible, and extraction of all information of few of the major effects. This could be resolved using various rate-limiting strategies:

- Limiting per connection property (IP address)
- Limiting per user (account / access token / API key)
- Limiting per application property (user account / resource type)
- Limiting based on context (region / type of app)

Once these rate-limiting features have been introduced if someone tried to repeatedly access the API, they would get the following error.

```
HTTP/1.1 429 Too Many Requests
Retry-After:  3600
```

Rate limiting prevents malicious code from abusing legitimate / illegitimate access to the API.

One of the basic and yet most powerful technique which helps to prevent a lot of security issues is Input validation (31). Input validation should act as the first line of defense in case of a malicious attack. It is also useful to reject malicious data which helps prevent DoS attacks by rejecting unreasonably large inputs and against injection attacks by rejecting crafted payloads. Some of the possible Input validation techniques which could be easily enforced in my case are:

- Enforcing sensible length limits on inputs.
  (i.e., 2MB of user's hashed dataset is not allowed)
- Enforcing strict content types on provided data inputs.
  (i.e., an API expecting JSON data should not accept anything else.
- Enforcing strict data type checking on inputs.
  (i.e., Numbers should be numbers, and SQL code as input results in an error)

Even though input validation is a decent first line of defense, it will fail if used as the only line of defense because when the application evolves and brings in complex input types it will not prevent attacks as it will be complex to determine the validity of complex data at input time. Moreover, complex validation procedures usually suffer from bypass attacks, and making validations might break functionality. (32)

3.7.2 Data Protection

Another possible issues specific to FMLearn is the storage of hash for the complete dataset in the proposed prototype, the dataset is hashed using sha256 though this is very secure it has its own set of vulnerabilities (33) that need to be taken into consideration, for now we can assume that sha256 is safe to use. But if it is broken there is a possibility that all the user data can become compromised. Even if the hash of the dataset is not broken, but if the meta-data about the model (like the model parameters) and dataset is available to the hacker, it is very much possible that a model could be reconstructed and the data of the user can be compromised.

Currently FMLearn is also vulnerable to Eavesdropping Attack, this is because the data sent from scikit-learn to FMLearn is not encrypted and is in plain-text, so various sniffing tools like Wireshark, Nmap, etc., can be used to monitor the network traffic and sniff the data transmitted between client and the server. To prevent such attacks, I can use a technique similar to twitter's developer API access, where in the user is required to create an account with FMLearn and is required to use certain keys generated during account creation:

```
ACCESS_TOKEN = ''
ACCESS_TOKEN_SECRET = ''
API_KEY = ''
API_SECRET_KEY = ''
```

This introduces a 3-legged oauth authentication workflow similar to twitter's developer API (34) access and these tokens can them be used to securely transfer data back and forth between the client and server, without worrying about eavesdropping attack. But then by introducing this we bring in new security vulnerabilities into the application which needs to be addressed separately in a larger scale. But for now sticking to just encrypting the information related to dataset let it be dataset hash or meta-data about the data, can be done as the first step in the necessary direction for this dissertation to proceed using public key encryption techniques. Doing just this helps protect the privacy of the user's data from potential sniffers and maintains the integrity of the data over the network.

The scope of improvement in terms of security is huge in this project, it can be made secure enough for users to confidently use, but all this is just until things don't go wrong.

3.7.3 Social Concerns

In the long run, social questions need consideration such as preventing manipulation of the data, developers of algorithms may have an interest that their algorithms are "recommended" to other users. So, the developers might manipulate data so the underlying trained model is skewed to recommend their algorithm for any task. Doing this will result is both unfair advantage to the developers algorithm making it popular and also cause FMLearn to suggest wrong algorithms. Another social issue that must be addressed in the free-rider problem, where the users benefit from the system without sharing their data.

4 Evaluation

In this section, the performance of FMLearn application is examined. The performance of the application is analysed based on the amount of time saved by the application when compared to the traditional methods of algorithm selection and hyper-parameter configuration. Apart from analysing the amount of time saved, this section also provides information about the electricity and money saved while using the FMLearn application. This section also provides information about the model's accuracy while recommending the best algorithm(s) for a given task. Since Federated Meta-Learning and FMLearn are a Novel Concept and a Novel Application respectively there are no pre-defined standards used to evaluate the accuracy of algorithm recommendation. Evaluation metrics like Mean Average Precision, Mean Average Recall, Intra-list Similarity, etc., used for recommender systems may not be able to accurately evaluate FMLearn. Moreover, there can be more than one best performing algorithm for the given dataset. Therefore, such evaluations are based on the comparisons from previous records obtained by performing algorithm selection and hyper-parameter optimisation for the same dataset under similar conditions.

NOTE: all these evaluation and testing are performed on the hardware mentioned in the Appendix A1.2.

4.1 Methodology

To evaluate the performance of FMLearn application, FMLearn's client was setup on two computers. The first computer trained eight machine learning algorithms on 2 different types of dataset, code for which can be found on GitHub:

https://github.com/mukeshmk/toy-datasets

The datasets can be classified into a small and large dataset depending on the number of instances each dataset contains. Five small datasets having about 500 instances each and five large dataset having about 15,000 - 250,000 instances were used.

Small Datasets

The small dataset used were: Breast Cancer (35), Diabetes (36), Wine (37), Boston (38) and Iris (39). These datasets were available as part of `scikit-learn` library and can be imported as follows.

```python
# package in which the datasets are available at:
from sklearn import datasets

# importing the dataset into the program
# example: boston dataset
boston_ds = datasets.load_boston()

# similarly for other mentioned datasets using the following methods
# load_diabetes(), load_breast_cancer(), load_iris() and load_wine()
```

Large Datasets

The **UCI Machine Learning Repository** (40) was used to obtain the large datasets, namely Adult, MAGIC Gamma Telescope, Skin Segmentation (41), Statlog-Shuttle and Nursery (40) datasets, these datasets have about 15,000 - 250,000 instances and are available to the public as a CSV file. This data was loaded into the program as follows:

```python
# importing pandas - a data manipulation and analysis
import pandas as pd

# importing data
dataset_df = pd.read_csv(path_to_data + "/file_name.csv", sep=',')
```

First Machine

On the First Machine, the task of algorithm selection and configuration was performed using the traditional time consuming method, to publish the performance metrics of the best performing algorithm-data pairs on to FMLearn. This was done so that the model can be re-trained with the published data.

Algorithm Selection

The evaluation process required finding the best performing algorithm for a given dataset along with with hyper-parameters. To find the best performing algorithm the code segment available below was used, here various algorithms were selected for evaluation and then added to a list called `models`. The algorithms in this list were used with their default hyper-parameter configuration to build a model and make predictions. The accuracy score of these models were recorded and then two to three best performing algorithms were chosen for hyper-parameter optimisation in the next step.

```
# assuming the required packages have been imported and
# data has been split into testing and training data.

models = []
models.append(('RFC', ensemble.RandomForestClassifier()))
# similarly adding other algorithms for evaluation before
# selecting the best performing algorithm

# finding the best algorithm
names = []
scores = []
for name, model in models:
    model.fit(x_train, y_train)
    y_pred = model.predict(x_test)
    score = accuracy_score(y_test, y_pred)
    scores.append(score)
    names.append(name)

print(pd.DataFrame({'Name': names, 'Score': scores}))
```

Hyper-Parameter Optimisation

Grid-Search technique was used for hyper parameter optimization and cross-validation. For the best performing algorithm a parameter grid was constructed where the different hyper-parameter values were set and this grid was sent to the `GridSearchCV` method along with the cross-validation datasets and a few other configurations.

```
# cross validation specifications
strat_k_fold = StratifiedKFold(n_splits=5, random_state=10)
```

```
# an example of the parameter grid for LogisticRegression
c_values = list(np.arange(1, 10))
param_grid = [
{'C': c_values, 'penalty': ['l1', 'l2'], 'solver': ['newton-cg', 'lbfgs',
    'liblinear', 'sag', 'saga'], 'multi_class' : ['ovr']}
]

# GridSearchCV for LogisticRegression
grid = GridSearchCV(linear_model.LogisticRegression(max_iter=10000),
    param_grid, cv=strat_k_fold, scoring='accuracy', iid=False)

grid.fit(X, Y)

# prints the best hyper-parameter configuration
print(grid.best_params_)
# prints the best scores
print(grid.best_estimator_)
```

This process resulted in the optimal hyper-parameters for the algorithm-dataset pair and this information was then published to FMLearn via the client.

Second Machine

On a second machine, the same experiments were run, but before the training started, an API call to FMLearn was made, where a prediction/recommendation request was made for the best performing algorithm for the given dataset. In this scenario the client just used the returned recommendation for the best algorithm with its hyper parameters, no training was needed. The model was created with the recommended results and then was re-optimised which enabled the user to proceed with the task of making predictions.

4.2 Results

FMLearn automatically submits all performance metrics and algorithm names along with the meta-features and hashes of the datasets to the server via the API calls from the client. The total execution time for traditional approach in algorithm selection and configurations methods are between 13.67 minutes (Iris) and 94.24 minutes (Breast Cancer) for the small datasets (see Table: 4.1), and between 256.42 minutes (Nursery) and 869.74 minutes (Skin Segmentation) for the large dataset (see Table 4.2). Whereas the total execution time, even if the user chooses to re-optimise the hyper-parameters are between 0.31 minutes (Iris) and 18.84 minutes (Breast Cancer) for small datasets and between 15.51 minutes (Nursery) and 29.91 minutes (Skin Segmentation) for large datasets.

Execution Time (in minutes) for Small Datasets				
Datasets	Optimize All Algorithms	FMLearn	Re-Optimise best algorithm	Saving in %
Breast-Cancer	94.24	0.05	18.79	80
Boston	47.36	0.05	6.96	85.01
Diabetes	62.17	0.04	10.37	83.25
Wine	26.54	0.04	3.25	87.6
Iris	13.67	0.02	0.29	97.73
Average	48.796	0.04	7.932	86.718

Table 4.1: Execution time when using GridSearch vs FMLearn for small datasets

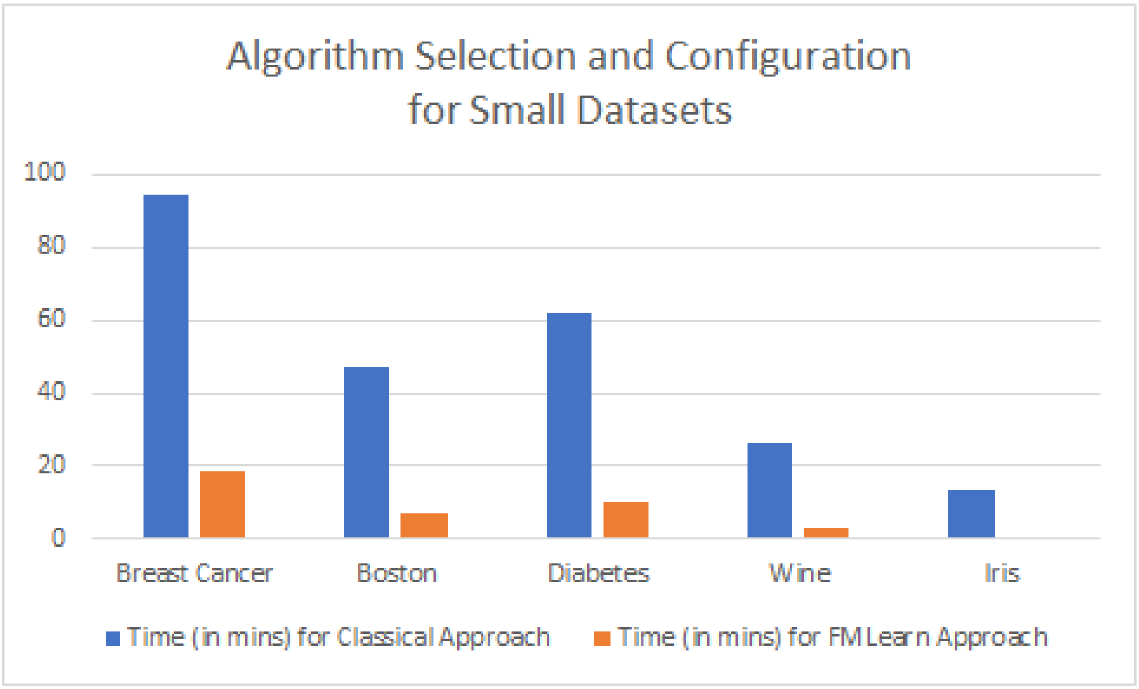


Figure 4.1: Execution Time for Small Datasets as seen in Table 4.1

Hence, for a small dataset, the user saves an average of 48.79 minutes and about 92.24 minutes (for Breast-Cancer Dataset) in a best case scenario. Whereas, for a large dataset the user saves an average of 533.21 minutes and about 869.74 minutes (for Skin Segmentation Dataset) in a best case scenario. This amounts to about 86.72% and 95.762% (for small and large datasets respectively) of time saved for the user, which otherwise is spent waiting for the machine learning program to performs algorithm selection and hyper-parameter optimisation to select the best algorithm-parameters pair for the given dataset. In a scenario where the user would want to re-optimize hyper parameters, re-training was required for only the best algorithm suggested by FMLearn. Under these circumstances, time saved on an average by the user was about 40.864 minutes for small datasets and 513.15 minutes for large datasets. These recommendations for best performing algorithms were consistent with the previously obtained results for the same data and thus reporting a 100% accuracy for recommendations for previously seen datasets.

Execution Time (in minutes) for Large Datasets				
Datasets	Optimize All Algorithms	FMLearn	Re-Optimise best algorithm	Saving in %
Adult	582.51	0.05	19.01	96.72
MAGIC Gamma Telescope	279.01	0.04	14.63	94.74
Nursery	256.42	0.04	15.47	93.95
Skin Segmentation	869.74	0.05	29.86	96.56
Statlog-Shuttle	678.37	0.04	21.35	96.84
Average	533.21	0.044	20.06	95.762

Table 4.2: Execution time when using GridSearch vs FMLearn for large datasets

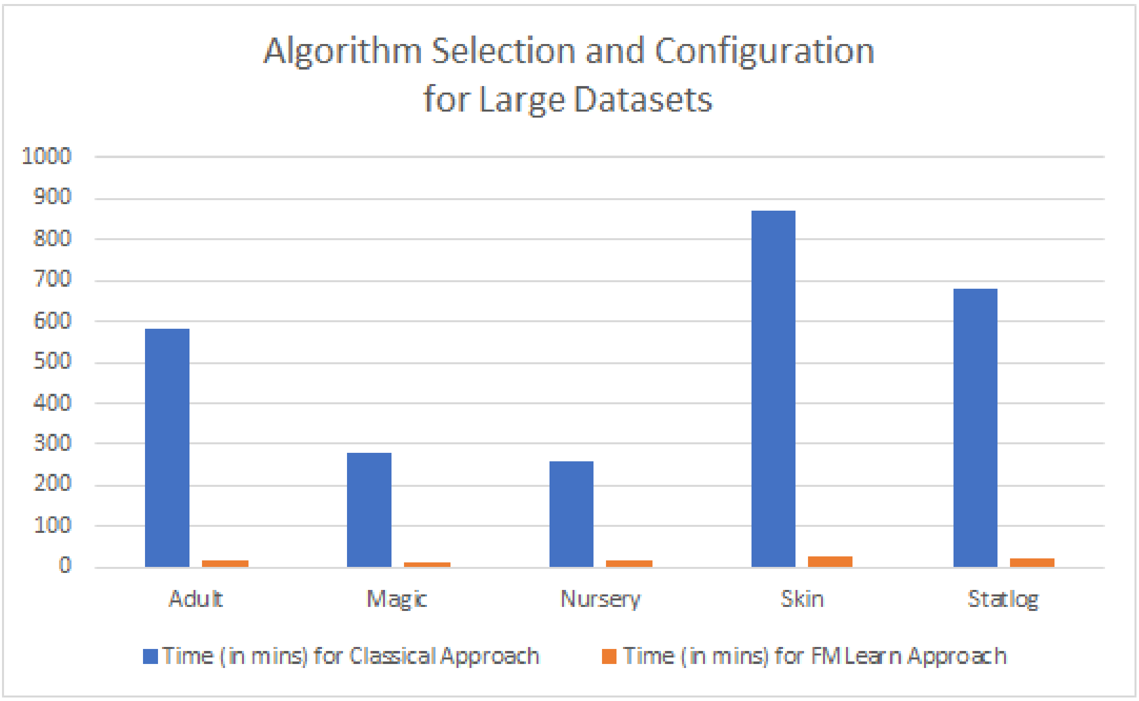
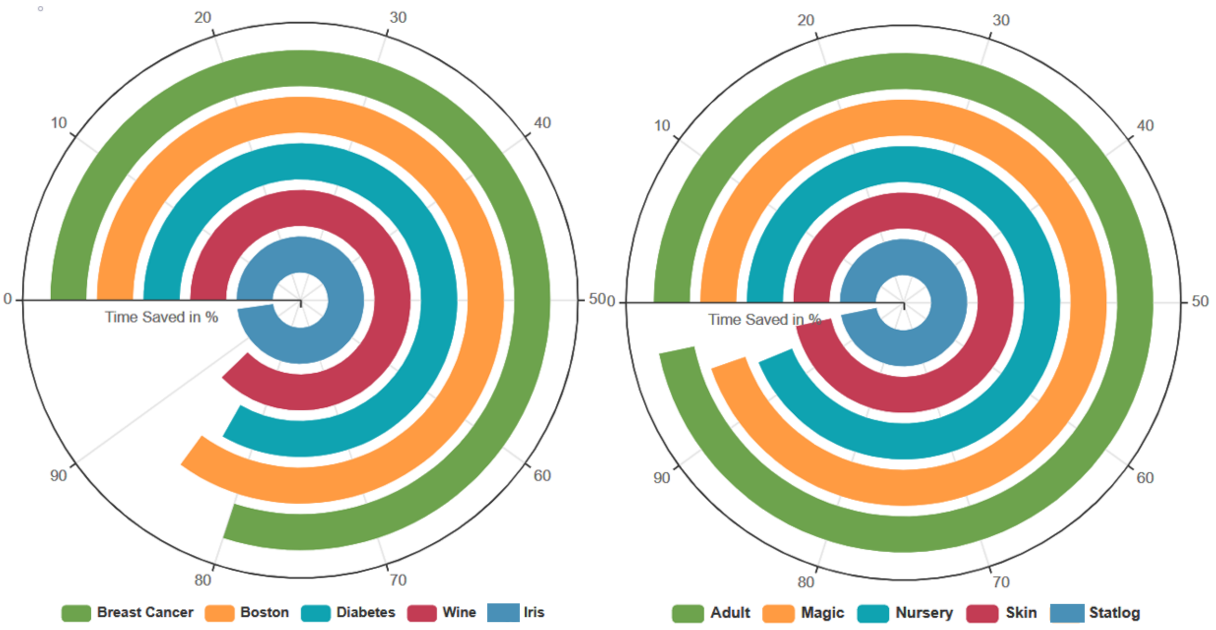


Figure 4.2: Execution Time for Large Datasets as seen in Table 4.2

Figures 4.1 and 4.2 visually represent and compare the overall time consumed by traditional approach vs FMLearn's approach. From these figures we can see the advantages of using FMLearn over the traditional approach for algorithm selection or other AutoML libraries. FMLearn provides accurate and quick responses by making recommendations from a model created using historic performance data. Since, it is an ever growing and ever learning application, the recommendations made by FMLearn gets better over time. Percentage of Time saved by using FMLearn for small and large datasets are represented in Figure 4.3. Figure 4.3a represents the percentage of time saved for small datasets as a radial chart, where each doughnut represents a dataset and the arc length of the doughnut represents the percentage of time saved by the user when using FMLearn. Similarly figure 4.3b represents the percentage of time saved for a large dataset.



(a) Small Dataset (b) Large Dataset

Figure 4.3: Percentage of Time Saved

Upon considering the statistics obtained from the above results and using them in calculating the power consumption, we can see that the average power consumed for finding the best performing algorithm using the traditional approach for one of the small datasets is about 65.061W, but when FMLearn was used the average power consumed is only about 10.629W which is about 86.718% saving in the power used. In the case of large datasets, the average power consumed to find the best performing algorithm is about 710.946W, but when FMLearn was used the average power consumed is only about 26.805W, which approximates to 95.762% reduction in power utilisation when compared to the traditional approach where FMLearn is not used.

Algorithm Recommendations

The best algorithm recommendations falls into 3 different categories based on the closeness of the input dataset as explained in Section 3.5.1, which can be categorised as follows:

- Previously known dataset.
- Previously unknown dataset, which is similar to a known dataset.
- Previously unknown dataset, which is dissimilar to a known dataset.

The results discussed above are for previously known datasets. In the case of unknown but similar datasets, a the same experiment was conducted but with one modification, instead of using a known dataset, an unknown yet, similar datasets was used. These similar datasets were obtained from a large dataset like the Statlog-Shuttle dataset. For example, this dataset was taken and broken down into small chunks of about 100k records, and these small subsets of data were used to get recommendations from FMLearn. Though the meta-features of the dataset varied, we can be confident that the changes in values will not be drastic as they were obtained from the same parent dataset. In this case, the prediction of algorithm is accurate, but the hyper-parameters requires re-optimisation to suite the new subset of the data. Whereas, in the case of a previously unseen dataset which is highly dissimilar to the datasets known to FMLearn, the model is able to predict the type of machine learning problem the dataset belongs to, i.e, if it's a classification, regression, clustering, etc, but not the best algorithm for it, it instead recommends a set of algorithms which it thinks are the best in this case instead of recommending a single algorithm. In this case, the best performing algorithm is recommended by FMLearn about 60% of the time. This set of algorithms are based on the closeness between the previously known datasets.

5 Conclusion

From the evaluation of the concept of Federated Meta-Learning via implemented version of the application FMLearn, there is a clear improvement in the workflow of Algorithm Selection and Hyper-Parameter Optimisation. The algorithm recommendation system was specifically designed to replicate real work scenarios, thus facilitating the algorithm selection and configuration process. Though the application FMLearn is a prototype and a proof of concept for Federated Meta-Learning, the results proved that the concept and the application saves about 86.718% of the time when compared to the traditional process of algorithm selection in the case of small datasets and about 95.762% of time in the case of large datasets. Apart from saving time, the results also proved the reduced use of energy, thanks to the reduced amount of time consumed, 86.718% and 95.762% reduction for small and large datasets respectively. This reduction in energy and time will also save money and computational resources for developers.

The biggest contribution of FMLearn in the algorithm selection and configuration process is, elimination of the repetitive and time consuming nature of the task. Other major contributions can be attributed to the accurate nature of recommendations made and the ability of the system to grow overtime. As discussed in Section 3.5.1 and Section 3.5.2, FMLearn makes better predictions when it has more data to work with. The model rebuilding/retraining techniques used in the application makes it easier for FMLearn to make better predictions and improve user experience.

6 Limitations and Future Work

FMLearn and Federated Meta-Learning opens up new avenues for research, the current implementation of FMLearn is a prototype, which just scratches the surface when compared to it's full potential.

Limitations

The current implementation of FMLearn is limited to tabular datasets and feature-based supervised machine learning algorithms. The limitation with respect to the use of tabular datasets is due to the fact that obtaining meta-features which describe different types of data like image, audio, video, etc., accurately are not available and is an area of intense research (42) (43). There are no widely used tools available and developing such a tool is out of scope of this research. The restriction to use a feature-based supervised machine learning algorithms is that they are relatively less complex and having less number of hyper-parameters when compared to complex structures and neural networks with thousands of hyper-parameters, implementing this was avoided due to the limitation in time.

Future Work

The immediate future work concerning this project should be to move the client out of scikit-learn and develop it as a stand alone library, this will enable wide spread use in the community which will result in the availability of vast variety of datasets. The increased availability will result in a better model thus improving the recommendations made my FMLearn. A more ambitious future work of this project could lead to research and implementation related to recommending complex neural networks or working with other forms of data source apart from tabular datasets.

Bibliography

[1] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Frechette, Holger Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. Aslib: A benchmark library for algorithm selection. *Artificial Intelligence*, 237, 06 2015. doi: 10.1016/j.artint.2016.04.003.

[2] Joaquin Vanschoren, Carlos Soares, Pavel Brazdil, and Lars Kotthoff. *Meta-Learning and Algorithm Selection*. 08 2014.

[3] Roberto Calandra. Workshop on meta-learning (metalearn 2017), 2020. URL `http://metalearning.ml/2017/`.

[4] Andrew Collins, Dominika Tkaczyk, and Joeran Beel. A novel approach to recommendation algorithm selection using meta-learning. In *AICS*, pages 210–219, 2018.

[5] Cristóbal Romero, Juan Luis Olmo Ortiz, and Sebastian Ventura. A meta-learning approach for recommending a subset of white-box classification algorithms for moodle datasets. 01 2013.

[6] M Vartak, A Thiagarajan, C Miranda, J Bratman, and H Larochelle. A meta-learning perspective on cold-start recommendations for items. advances in neural information processing systems. page 6907–6917, 2017.

[7] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren. *Automated Machine Learning - Methods, Systems, Challenges*. 01 2019.

[8] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Tobias Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In *Proceedings of the 28th International Conference on Neural Information Processing Systems - Volume 2*, NIPS'15, page 2755–2763, Cambridge, MA, USA, 2015. MIT Press.

[9] Ciro Castiello, Giovanna Castellano, and Anna Fanelli. Meta-data: Characterization of input features for meta-learning. pages 457–468, 07 2005. doi: 10.1007/11526018_45.

[10] Andrey Filchenkov. Datasets meta-feature description for recommending feature selection algorithm. 11 2015. doi: 10.1109/AINL-ISMW-FRUCT.2015.7382962.

[11] Lars Kotthoff, Chris Thornton, Holger H. Hoos, Frank Hutter, and Kevin Leyton-Brown. Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka. *Journal of Machine Learning Research*, 18(25):1–5, 2017. URL http://jmlr.org/papers/v18/16-261.html.

[12] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107, 07 2018. doi: 10.1007/s10994-018-5735-z.

[13] Masoud Mansoury and Robin Burke. Algorithm selection with librec-auto. 04 2019.

[14] Auto-surprise: An automated recommender-system (autorecsys) library with tree of parzens estimator (tpe) optimization, 2020.

[15] Joeran Beel. Federated meta-learning: Democratizing algorithm selection across disciplines and software libraries. 04 2019. doi: 10.13140/RG.2.2.25744.35844.

[16] Mukesh Arambakam and Joeran Beel. Federated meta-learning: Democratizing algorithm selection across disciplines and software libraries. *7th ICML Workshop on Automated Machine Learning (AutoML)*, 2020. URL https://www.automl.org/wp-content/uploads/2020/07/AutoML_2020_paper_39.pdf.

[17] Edesio Alcobaça, Felipe Siqueira, Adriano Rivolli, Luís P. F. Garcia, Jefferson T. Oliva, and André C. P. L. F. de Carvalho. Mfe: Towards reproducible meta-feature extraction. *Journal of Machine Learning Research*, 21(111):1–5, 2020. URL http://jmlr.org/papers/v21/19-348.html.

[18] Fei Chen, Zhenhua Dong, Zhenguo Li, and Xiuqiang He. Federated meta-learning for recommendation. 02 2018.

[19] Diego Peteiro-Barral and Bertha Guijarro-Berdiñas. A survey of methods for distributed machine learning. *Progress in Artificial Intelligence*, 2(1):1–11, 2012. doi: 10.1007/s13748-012-0035-5.

[20] Qiang Yang, Yang Liu, Tianjian Chen, and Yongxin Tong. Federated machine learning: Concept and applications. *ACM Trans. Intell. Syst. Technol.*, 10(2), January 2019. ISSN 2157-6904. doi: 10.1145/3298981. URL https://doi.org/10.1145/3298981.

[21] Joaquin Vanschoren, Jan N. van Rijn, Bernd Bischl, and Luis Torgo. Openml: Networked science in machine learning. *SIGKDD Explorations*, 15(2):49–60, 2013. doi: 10.1145/2641190.2641198. URL http://doi.acm.org/10.1145/2641190.2641198.

[22] John R. Rice. The algorithm selection problem**this work was partially supported by the national science foundation through grant gp-32940x. this chapter was presented as the george e. forsythe memorial lecture at the computer science conference, february 19, 1975, washington, d. c. volume 15 of *Advances in Computers*, pages 65 – 118. Elsevier, 1976. doi: https://doi.org/10.1016/S0065-2458(08)60520-3. URL http://www.sciencedirect.com/science/article/pii/S0065245808605203.

[23] Pascal Kerschke, Holger H. Hoos, Frank Neumann, and Heike Trautmann. Automated algorithm selection: Survey and perspectives, 2018.

[24] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. Satzilla: Portfolio-based algorithm selection for sat. *Journal of Artificial Intelligence Research*, 32:565–606, 2008. doi: 10.1613/jair.2490.

[25] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[26] Miguel Grinberg. *Flask Web Development: Developing Web Applications with Python*. O'Reilly Media, Inc., 1st edition, 2014. ISBN 1449372627.

[27] Hadi S. Jomaa, Lars Schmidt-Thieme, and Josif Grabocka. Dataset2vec: Learning dataset meta-features, 2019.

[28] 2020. URL https://en.wikipedia.org/wiki/Client-server_model.

[29] Venkataramulu Sunkari and Chakunta Rao. Input validation vulnerabilities (sqlia) and defenses in web applications security. *International Journal of Emerging Technology and Advanced Engineering (IJETAE)*, 4:207–217, 07 2014.

[30] Linlin Xie, Paul Smith, David Hutchison, Mark Banfield, Helmut Leopold, Abdul Jabbar, and James Sterbenz. From detection to remediation: A self-organized system for addressing flash crowd problems. pages 5809–5814, 05 2008. doi: 10.1109/ICC.2008.1087.

[31] Venkataramulu Sunkari and Chakunta Rao. Input validation vulnerabilities (sqlia) and defenses in web applications security. *International Journal of Emerging Technology and Advanced Engineering (IJETAE)*, 4:207–217, 07 2014.

[32] De Ryck PHILIPPE. Common api security pitfalls, 2018. URL https://owasp.org/www-chapter-belgium/assets/2018/2018-10-23/OWASP_20181023_CommonAPISecurityPitfalls.pdf.

[33] Henri Gilbert and Helena Handschuh. Security analysis of sha-256 and sisters. In Mitsuru Matsui and Robert J. Zuccherato, editors, *Selected Areas in Cryptography*, pages 175–193, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[34] Twitter developer authentication, 2020. URL https://developer.twitter.com/en/docs/authentication/oauth-1-0a.

[35] Brendan McMahan and Daniel Ramage. Federated learning: Collaborative machine learning without centralized training data, 2017. URL https://ai.googleblog.com/2017/04/federated-learning-collaborative.html.

[36] Robert Tibshirani, Iain Johnstone, Trevor Hastie, and Bradley Efron. Least angle regression. *The Annals of Statistics*, 32(2):407–499, 2004. doi: 10.1214/009053604000000067.

[37] Uci machine learning repository: Wine dataset, 1991. URL https://archive.ics.uci.edu/ml/datasets/wine.

[38] David Harrison and Daniel Rubinfeld. Hedonic housing prices and the demand for clean air. *Journal of Environmental Economics and Management*, 5:81–102, 03 1978. doi: 10.1016/0095-0696(78)90006-2.

[39] Ronald Aylmer Fisher. *Contributions to mathematical statistics*. Wiley, 1950.

[40] Dheeru Dua and Casey Graff. UCI machine learning repository, 2017. URL http://archive.ics.uci.edu/ml.

[41] Rajen Bhatt, Abhinav Dhall, Gaurav Sharma, and Santanu Chaudhury. Efficient skin region segmentation using low complexity fuzzy decision tree model. pages 1 – 4, 01 2010. doi: 10.1109/INDCON.2009.5409447.

[42] Jeffrey S. Ellen, Casey A. Graff, and Mark D. Ohman. Improving plankton image classification using context metadata. *Limnology and Oceanography: Methods*, 17(8):439–461, 2019. doi: 10.1002/lom3.10324. URL https://aslopubs.onlinelibrary.wiley.com/doi/abs/10.1002/lom3.10324.

[43] M. Calderisi, G. Galatolo, I. Ceppa, T. Motta, and F. Vergentini. Improve image classification tasks using simple convolutional architectures with processed metadata injection. In *2019 IEEE Second International Conference on Artificial Intelligence and Knowledge Engineering (AIKE)*, pages 223–230, 2019.

A1 Appendix

A1.1 Code Availability

All the code which is used for this paper is made available on GitHub:

- The modified scikit-learn is available at:
  https://github.com/mukeshmk/scikit-learn
- The code for FMLearn application is available at:
  https://github.com/mukeshmk/fm-learn
- The code used for the evaluation of the data is available at:
  https://github.com/mukeshmk/toy-datasets

The FMLearn application has also been deployed on heroku at https://fmlearn.herokuapp.com/.

NOTE: this is just a API server, the website doesn't provide any functionality per-se.

A1.2 Machine Details

The configuration and details of the machine used to measure these metrics are as follows:

- Laptop: Lenovo Legion Y540
- Processor: Intel i5 9300H
- RAM: 8GB
- GPU: Nvidia GTX 1650
- Hard Drive: 1TB HDD + 125GB SSD
- OS: Windows 10
- Power Consumption: avg 80W/hour

A1.3 Meta-Features: Auto-Sklearn

Meta-features used to describe the dataset are:

```
{
    ClassEntropy, SymbolsSum, SymbolsSTD, SymbolsMean, SymbolsMax, SymbolsMin,
    ClassProbabilitySTD, ClassProbabilityMean, ClassProbabilityMax,
    ClassProbabilityMin, InverseDatasetRatio, DatasetRatio,
    RatioNominalToNumerical, RatioNumericalToNominal,
    NumberOfCategoricalFeatures, NumberOfNumericFeatures,
    NumberOfMissingValues, NumberOfFeaturesWithMissingValues,
    NumberOfInstancesWithMissingValues, NumberOfFeatures, NumberOfClasses,
    NumberOfInstances, LogInverseDatasetRatio, LogDatasetRatio,
    PercentageOfMissingValues, PercentageOfFeaturesWithMissingValues,
    PercentageOfInstancesWithMissingValues, LogNumberOfFeatures,
    LogNumberOfInstances & TargetType.
}
```

On the next page you will went an example.

An example of meta-features obtained from the *calculate_all_metafeatures_with_labels()* method from autosklearn library for the Iris Dataset are:

```
{
  ClassEntropy: 0.9550393021523922
   SymbolsSum: 0.0
   SymbolsSTD: 0
   SymbolsMean: 0
   SymbolsMax: 0
   SymbolsMin: 0
   ClassProbabilitySTD: 0.12417582417582418
   ClassProbabilityMean: 0.5
   ClassProbabilityMax: 0.6241758241758242
   ClassProbabilityMin: 0.3758241758241758
   InverseDatasetRatio: 15.166666666666666
   DatasetRatio: 0.06593406593406594
   RatioNominalToNumerical: 0.0
   RatioNumericalToNominal: 0.0
   NumberOfCategoricalFeatures: 0
   NumberOfNumericFeatures: 30
   NumberOfMissingValues: 0.0
   NumberOfFeaturesWithMissingValues: 0.0
   NumberOfInstancesWithMissingValues: 0.0
   NumberOfFeatures: 30.0
   NumberOfClasses: 2.0
   NumberOfInstances: 455.0
   LogInverseDatasetRatio: 2.719100037288795
   LogDatasetRatio: -2.7191000372887952
   PercentageOfMissingValues: 0.0
   PercentageOfFeaturesWithMissingValues: 0.0
   PercentageOfInstancesWithMissingValues: 0.0
   LogNumberOfFeatures: 3.4011973816621555
   LogNumberOfInstances: 6.12029741895095
   TargetType: binary

}
```