On the Reusability of Machine Learning Models in Edge Computing: A Statistical Learning Approach

Xenia Skotti[1], Kostas Kolomvatsos[2], and Christos Anagnostopoulos[1]

[1] School of Computing Science, University of Glasgow, UK
[2] Dept. Computer Science & Telecommunications, University of Thessaly, GR,
xeniaskotti@gmail.com, kostasks@uth.gr,
christos.anagnostopoulos@glasgow.ac.uk

Abstract. The adoption of Edge Computing continues to grow with edge nodes recording increasingly more data, which inevitably requires that they should be processed through Machine Learning (ML) models to speed up the production of knowledge. However, training these models requires an increased amount of resources, which are limited, thus, the reuse of ML models becomes of paramount importance. Given that we do not have a pool of models to choose from, *is it possible to determine which nodes in the network require distinct models and which of them could be reused?* In this paper, we propose a solution to this question, an online model reuse framework which is evaluated for its precision and speedup. The framework considers all possible combinations of pairs in the network to determine which are good reusability pairs, by adopting statistical learning methods. Then for each pair, the node model is chosen that has the highest data space overlap. Our comprehensive experimental analysis in the context of both regression and classification shows the feasibility our solution in model reusability in Edge Computing environments.

Keywords: Edge Computing, Model Reusability, Machine Learning.

1 Introduction

Lee et. al [6] define compute reuse as 'the partial or full utilization of already executed computational task results by multiple users to complete a new task while avoiding computation redundancy'. Systems that adopt compute reuse benefit from significant performance gains motivating model reuse in Machine Learning (ML). Model reuse [14] attempts to construct a model from other pre-existing and pre-trained models for other tasks, in order to avoid building a model from scratch. Exploitation of pre-existing models can set a good basis for the training of a new model which translates into a reduced time cost, data amount and expertise required to train a new model. Moreover, model reuse has been used to tackle concept drift [13] and building ad-hoc analytic models [5].

Model reusability is compelling and, therefore, both theoretical [14] and empirical [5,12] frameworks have been proposed to take advantage of it. Many of the

proposed approaches involve a two-phased framework of a preprocessing and runtime phases, i.e., the model and its data are shared in a pool from which, in the runtime phase, the relevant ML models are identified. Consider the case of edge computing, where given a number of nodes and their corresponding datasets, we want to decide for which nodes to train a distinct model and for which to reuse one. In this context, the reuse comes from the fact that we do not train a model for all nodes but instead reuse one of the existing ones. A framework for model reuse in edge computing requires its online presence, thus, the aforementioned steps are merged. To the best of our knowledge no such framework has been proposed so far in the respective literature.

One of the fundamental requirements of any model reuse framework is to be able to choose the model that best fits the (test) data of the target domain. One of the ways this can be achieved is by finding the model whose source domain (training data) is drawn from the same distribution as the target domain. Therefore, the difference between domains needs to be quantified and minimised to find the best model. This is essentially what the Maximum Mean Discrepancy (MMD) [3] statistic does.

In addition to measuring the similarity between two datasets, we need to determine the direction of reusability. In other frameworks [5,12], the reused model originated from a pool, hence there was no such requirement because there was only one direction of reusability, the pool. In this setting though there are two directions per pair, and we need to define a method to do so. The method needs to measure the data space overlap between two datasets to determine potentially which would be better suited to be used to train a replacement model for the other.

The data space overlap can also be defined as the overlap of the inlier data space. A predictor for inlier space overlap is the probability of correctly predicting the non-native inliers of a model. In other words, what is the overlap between the inlier points of two datasets, the native and non-native one with regards to the inlier detection model. The reason behind using inliers to determine the overlap is that any dataset is expected to have a few outliers and hence some filtering needs to be applied anyway. Simultaneously, this can also be leveraged to determine the direction of reusability. We used the One-class Support Vector Machines (OCSVM) [10] to determine which points are inliers. Therefore, given two nodes and their corresponding OCSVM models, we can use each OCSVM model to predict the other node's inliers and then find the probability of detecting them, hence their overlap.

The paper is organized as follows: Section II highlights the relevant research with regards to model reuse and elaborates on our contribution. Section III provides preliminaries of the theory behind MMD and OCSVM. In Section IV, we introduce the reusability framework and provide the corresponding algorithms. Experimental evaluation is summarized in Section V highlighting the real datasets and classifiers used, the parameter configuration and the definitions of metrics in the context of model reusability. Section VI concludes the

paper with discussion on the important findings along with limitations and directions for future work.

2 Related Work

Compute reuse has been investigated in the context of edge computing by [6] to quantify its gain. Experiments on edge-based applications showed that systems that adopt compute reuse can finish the same task up to five times faster. Motivated by similar concerns a theoretical paradigm named 'learnware' was proposed by Zhou [14]. More specifically, a learnware is a ML model that is pretrained and achieves good performance paired with a detailed specification. The vision behind the paradigm was that learnware models can be shared in a pool without their raw data, allowing the identification of pretrained models that satisfy their requirements without concerns over privacy violations. Therefore, the author identified three characteristics: reusable, evolvable and comprehensible as fundamental for a model to be considered a learnware.

Based on this paradigm, the Reduced Kernel Mean Embedding (RKME) [12] was presented, i.e., a two phased framework consisting of the upload and deployment phase. During the upload phase, each model is paired with its Kernel Mean Embedding (KME) of the dataset and added to the pool of models. Then, in the deployment phase either a single or a combination of models is chosen based on the RKHS distance between the testing (target) mean embedding and reduced (source) embedding of pool models. In essence, the RKME's deployment phase, is similar to the MMD statistic [3], since by quantifying the distance of the mean embedding of two populations (source and target), it ensures that the target distribution is the same as the source.

In [14], the authors recognise transfer learning as a preliminary attempt to reusability. A two-stage framework dubbed as Learning to Transfer (L2T) was presented [11], which exploits previous transfer learning experiences to optimize what and how to transfer between domains. In the first stage each transfer learning experience is encoded into three parts and, then, are utilised to learn a reflection function, which approximates the performance improvement ratio and thus encrypts transfer learning skills of deciding what and how to transfer. The improvement ratio in this framework is the difference between domains calculated by MMD. In addition to the MMD between domains, the variance is also calculated since a small MMD paired with an extremely high variance still indicates little overlap. During the second stage, whenever a new pair of domains arrives, L2T optimizes the knowledge to be transferred by maximising the value of the learned reflection function.

Model reuse has also been used to handle concept drift. The assumption that previous data contain some useful information, indicates that the models corresponding to the data can be leveraged. Condor was proposed [13] as an approach to handle concept drift through model reuse. Condor consists of two modules, ModelUpdate and WeightUpdate which leverage previous knowledge

to build a new model, hence updating the model pool and adapting the weights of previous models to reflect current reusability performance respectively.

Hasani et al. [5] proposed a two-phased approach, to build faster models for a popular class of analytic queries. Similar to the other approaches [11] - [13], there is a preprocessing and a runtime phase. During the first phase the models, their statistics and some meta-data are stored, while in the second phase relevant models are identified from which an approximate model is constructed. Their approach can achieve speed-ups of several orders on magnitude on very large datasets, however, it is only geared towards exploratory analysis purposes and the approach is potentially less robust under concept drift.

Concerns over intellectual property (IP) infringement and vulnerability propagation of deep learning models (DNN) motivated the proposal of ModelDiff [8], a testing-based approach to DNN model similarity comparison. They compare the decision logic of models on the test inputs represented by a decision distance vector (DDV), a newly defined data structure in which each value is the distance between the outputs of the model produced by two inputs. These inputs are pairs of normal and corresponding adversarial samples and thus when used to calculate the DDV, the decision boundary is captured.

Lee et al. [6] also discuss alternative approaches and corresponding challenges of compute reuse including in networks. They identify that reuse can be achieved either in a distributed or centralized manner. The distributed approach involves forwarding tasks to the compute reuse node that is responsible for the operation. This adds additional complexity to the forwarding operations of routers resulting in a potential downgrade in performance. Reuse of results in a network setting, undoubtedly improves performance, however speeding up the estimation of parameters can also be beneficial in that regard. Nodes in a network can collaborate to estimate parameters as discussed in [7]. More specifically, their method takes advantage of the joint sparsity of vectors used for computations enhancing estimation performance. Joint sparsity simply means that the indexes of nonzero entries for all nodes are the same, but their values differ. The authors also adopt an intertask cooperation strategy to consider intertask similarities. Their method assumes that both the vectors of interest and their associated noise follow a zero-mean Gaussian distribution which is a strong assumption for the data to hold.

The contributions of this paper that, in parallel, depict its differences with other relevant efforts in the domain are as follows:

- An online model reuse framework for edge computing consisting of two steps, a pair similarity detector (based on MMD) followed by a direction of model reusability (based on the inlier data space overlap).
- A decision making algorithm which given the results of the framework it can maximise the number of nodes which do not require distinct models along with a list of replacement models.
- Extensive experimental evaluation of the framework with both classification and regression models over real datasets.

3 Background

3.1 Maximum Mean Discrepancy

MMD is a statistic that can quantify the mean discrepancy of two data distributions in a kernel space in order to determine if two samples are drawn from different distributions [3]. Let p and q be two independent probability distributions, and $E_x\left[f\left(x\right)\right]$ (shorthand notation for $E_{x\ p}\left[f\left(x\right)\right]$) denotes the mathematical expectation of $f\left(x\right)$ with x under the probability density p. The statistic definition between p and q is:

$$\begin{aligned} MMD\left(\boldsymbol{\mathcal{F}}, p, q\right) &= \sup_{f\ \in\ \boldsymbol{\mathcal{F}}} \left(E_x\left[f\left(x\right)\right] - E_y\left[f\left(y\right)\right]\right) \\ &= \sup_{f\ \in\ \boldsymbol{\mathcal{F}}} \left\langle f, \mu_p - \mu_q \right\rangle_{\boldsymbol{\mathcal{H}}} \end{aligned} \tag{1}$$

where the function class $\boldsymbol{\mathcal{F}}$ is a unit ball in the reproducing Hilbert space (RKHS) and μ_p, μ_q is the mean embedding of p and q respectively i.e., the mean of the feature mapping in the kernel space. The function class $\boldsymbol{\mathcal{F}}$ is universal meaning that $MMD\left(\boldsymbol{\mathcal{F}}, p, q\right) = 0$ if and only if $p = q$. Therefore, MMD is the largest difference in expectations over functions in $\boldsymbol{\mathcal{F}}$ and can only be zero if the two samples were drawn from the same distribution.

In practise, we use the square MMD in order to be able to use kernel functions. Let $X = \{x_1, ..., x_m\}$ and $Y = \{y_1, ..., y_n\}$ denote the independent and identically distributed (i.i.d.) samples from distribution p and q respectively. An unbiased estimation of $MMD^2\left(\parallel \mu_p - \mu_q \parallel^2_{\boldsymbol{\mathcal{H}}}\right)$ can be obtained using a U-statistic:

$$\begin{aligned} MMD^2\left(\mathbf{F}, p, q\right) = &\frac{1}{m(m-1)} \sum_{i=1}^{m} \sum_{j \neq i}^{m} k\left(x_i, x_j\right) + \\ &\frac{1}{n(n-1)} \sum_{i=1}^{n} \sum_{j \neq i}^{n} k\left(y_i, y_j\right) - \\ &\frac{2}{mn} \sum_{i=1}^{m} \sum_{j=1}^{n} k\left(x_i, y_j\right) \end{aligned} \tag{2}$$

where $k(.)$ denotes the kernel function. In our model, we adopt the linear and Gaussian RBF kernels as defined as: $k(x, y) = x^T y$ and $k(x, y) = \exp(-\frac{1}{2\sigma^2} \parallel x - y \parallel^2)$, where $\sigma \in \mathbb{R}$ is a kernel parameter and $\parallel x - y \parallel$ is a dissimilarity measure (e.g., Euclidean distance).

3.2 One-class Support Vector Machines

OCSVM is a one-class classification technique, which aims to classify instances into one of two classes, the inlier and outlier classes. It was first presented by Schölkopf et. al [10] and utilizes a training dataset with normal data to learn the boundaries of the normal data points. Therefore, data points which lie outside

the normal data region are going to be classified as outliers. OCSVMs utilize an implicit transformation function $\phi(.)$ defined by the kernel to project data to a higher dimensional space. The algorithm learns the decision boundary (a hyperplane) which achieves the maximum separation of the majority of data points. Only a small fraction of data are allowed to lie on the other side of the decision boundary and those data are considered outliers.

The OCSVM returns a function f that takes the value +1 for the normal region and -1 elsewhere. This function f is called a decision function being defined as: $f(x) = sign(g(x)) = sign(w^T\phi(x) - \rho)$ where w is the vector perpendicular to the decision boundary ($g(x) = 0$) and ρ is the bias. Given that the distance of any arbitrary data point to the decision boundary can be calculated by $d(x) = \frac{|g(x)|}{\|w\|}$ and that the origin's value when plugged to $g(x)$ is ρ, the distance of the origin to the decision boundary is $\frac{\rho}{\|w\|}$. The OCSVM essentially attempts to maximise the distance by solving the minimisation problem of $\frac{\|w\|}{2} - \rho$, i.e.,

$$\min_{w,\xi\in\mathbb{R}^N,\rho\in\mathbb{R}} \frac{\| w \|^2}{2} - \rho + \frac{1}{vn}\sum_{i=1}^{n}\xi_i \tag{3}$$

$$\text{subject to } (w^T \cdot \Phi(x_i)) \geq \rho - \xi_i, \xi \geq 0$$

where ξ_i is the slack variable for a point i which allows it to lie on the other side of the decision boundary, n is the size of the training dataset and $v \in (0, 1)$ is a regularization parameter. As shown in (3) the objective is not only to minimise the distance of the origin to the decision boundary but also minimise the slack variables ξ_i for all points. v represents the upper bound limit of the fraction of outliers and a lower bound on the number of support vectors. In other words, v specifies the number of training points which are guaranteed to be misclassified and the number of training examples being support vectors. As mentioned above $v \in (0, 1)$ and therefore a percentage, where a high value may lead to over-fitting and a low value to under-fitting. v controls the trade off between ξ and ρ.

For reducing the number of variables to a single vector and utilise the kernel trick, the primary objective is transformed into a dual objective:

$$\min_{a} \frac{a^T Q a}{2} \tag{4}$$

$$\text{subject to: } 0 \leq a_i \leq \tfrac{1}{vn}, \textstyle\sum_{i=1}^{n} a_i = 1$$

where Q is the kernel matrix and a the Lagrange multipliers. Now, the decision function becomes:

$$f(x) = sign(\sum_{i=1}^{n} a_i k(x, x_i)) \tag{5}$$

Algorithm 1: Calculates the average similarity MMD (ASMMD) between the given nodes.

Data: ***kernel***, ***bandwidth***: the kernel type and scalar value to be used for the MMD calculation, ***samples***: dictionary associating each node with a sample, ***similar_nodes***: nodes identified as similar to each other, ***other_nodes***: the rest of the nodes.
Result: ***ASMMD***

begin
  `// Calculating the baseline ASMMD`
  $similar\_mmds \longleftarrow []$
  for x, y **in** $get\_pair\_combos(similar\_nodes)$ **do**
    $sx \longleftarrow samples[x]$, $sy \longleftarrow samples[y]$
    $mmd \longleftarrow MMD(sx, sy, kernel, bandwidth)$
    $similar\_mmds.append(mmd)$
  end
  `// Compare which of the the other_nodes are similar to the similar_nodes using the current ASMMD in each iteration`
  for x **in** $other\_nodes$ **do**
    $sx \longleftarrow samples[x]$
    for y **in** $similar\_nodes$ **do**
      $sy \longleftarrow samples[y]$
      $mmd \longleftarrow MMD(sx, sy, kernel, bandwidth)$
      $asmmd \longleftarrow$ ***mean***$(similar\_mmds)$
      if $mmd < asmmd * 1.05$ **then**
        $similar\_mmds.append(mmd)$
      end
    end
  end
  `// Which the other_nodes are similar to each other`
  if ***len***$(other\_nodes > 1)$ **then**
    for x, y **in** $get\_pair\_combos(other\_nodes)$ **do**
      $sx \longleftarrow samples[x]$, $sy \longleftarrow samples[y]$
      $mmd \longleftarrow MMD(sx, sy, kernel, bandwidth)$
      $asmmd \longleftarrow$ ***mean***$(similar\_mmds)$
      if $mmd < asmmd * 1.05$ **then**
        $similar\_mmds.append(mmd)$
      end
    end
  end
  $asmmd =$ ***mean***$(similar\_mmds)$
end

4 ML Models Reusability Framework

Our online model reuse framework needs to be able to determine two things given a pair of nodes. First and foremost, the pairs of nodes which have similar datasets and then the direction of reusability.

The first objective is achieved using MMD, which measures the difference domains and hence theoretically when the MMD value is zero this means the two datasets are drawn from the same distribution. However, as discussed in section 3.1 in practise we utilise an estimation of MMD squared. As a consequence, the value is not actually zero and we need to define a threshold below which a pair would be considered similar. we have dubbed the threshold to be the average similarity MMD (ASMMD), a value calculated using Algorithm 1. Algorithm 1 requires that we categorise nodes into two sets, one where all nodes are similar to each other and the rest of them. Categorising nodes in these categories differs when using a regression and classification dataset. We discuss this further in section 5.1.

Algorithm 2: Finds the similar pairs of the dataset using MMD

Data: ***samples***: dictionary associating each node with a sample, ***asmmd***: average similarity (ASMMD) calculated using Algorithm 1, ***kernel***, ***bandwidth***: the kernel type and scalar value to be used for the MMD calculation.
Result: $similar\_pairs, pair\_mmds$

```
begin
    similar_pairs ⟵ []
    pair_mmds ⟵ []
    nodes ⟵ samples.keys()
    for x, y in get_pair_combos(nodes) do
        sx ⟵ samples[x], sy ⟵ samples[y]
        mmd ⟵ MMD(sx, sy, kernel, bandwidth)
        if mmd < asmmd * 1.05 then
            similar_pairs.append((x, y))
            pair_mmds.append(mmd)
        end
    end
end
```

Once we have identified these two sets, we calculate a baseline ASMMD by calculating the MMD of all pair combinations of the similar nodes. Then, we use ASMMD (allowing for a 5% variation) to judge whether the rest of the nodes are similar to each other or to the similar nodes. If they are we calculate the new ASMMD and we use this to judge the next pair. Using the result of this process we can then judge which pairs are similar for a given experiment as demonstrated in Algorithm 2. It is worth highlighting that for the MMD implementation to

work and by extension all of the algorithms that utilize it (Algorithms 1 & 2), the samples of each node need to be of equal size.

Algorithm 3: Calculates the OCSVM score of each node per pair

Data: ***samples***: dictionary associating each node with a sample, ***models***: dictionary associating each node with its OCSVM node model, ***similar_pairs***: the MMD identified similar pairs
Result: *pair_prob*

```
begin
    pair_prob ⟵ []
    for x, y in similar_pairs do
        sx ⟵ samples[x], sy ⟵ samples[y]
        pred_y_inliers ⟵ get_inliers(models[x], sy),
          pred_x_inliers ⟵ get_inliers(models[y], sx)
        x_y_overlap ⟵ size(pred_y_inliers)/size(sy)
          y_x_overlap ⟵ size(pred_x_inliers)/size(sx)
        pair_prob.append((x_y_overlap, y_x_overlap))
    end
end
```

Once we identify the similar pairs in the network we can then calculate the OCSVM scores of each node in each pair and hence determine the direction of reusability per pair. The OCSVM score is essentially the probability of detecting the inliers of the node by using the other node's model. Therefore, given two nodes x and y, and their corresponding OCSVM models, we use each OCSVM model to predict the other node's inliers and then we calculate the number of points that were identified as inliers and divide by the number points in the dataset, hence the probability. The reason we divide by the number of points in the dataset is because we expect to do some form of filtering prior and remove the outliers if they exist, hence all the points in the dataset are inliers. We calculate the OCSVM score for both directions and whichever is higher is the node for which we should train the model for. Algorithm 3 calculates of the OCSVM scores of each node per pair.

The framework presented by this point operates on the node level, however in order unify the information to the network level we prose a naive decision making algorithm (Algorithm 4). The algorithm provides the user with information about which nodes do not require distinct models and the respective potential replacement models. The algorithm is naive and thus simple whose aim is to find the maximum number of nodes for which we do not train a model for. Nevertheless, the algorithm would not take into account any performance optimising considerations.

A visual representation of the framework being applied to a network is shown in Figure 1.

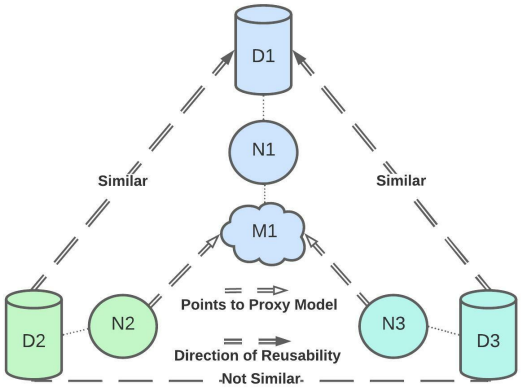
Algorithm 4: Finds nodes that can use a reused model, along with a list of replacements, based on the results of the framework

Data: ***pair_results***: dictionary associating each pair with the node whose model to be reused i.e. the direction of reusability, ***nodes***: the list of nodes from the MMD identified pairs

Result: ***mns***: modelless nodes i.e. nodes that do not require that a model is trained for them, ***model_mns***: associates each modelless node (mn) with a list of potential replacement node models

```
1  begin
2    similar_pairs ⟵ pair_results.keys()
3    mns ⟵ nodes.copy(), model_mns ⟵ {}
4    for node in nodes do
5    | model_mns[node] ⟵ []
6    end
7    for node in nodes do
8    | node_similar_pairs ⟵ get_node_similar_pairs(node, similar_pairs)
9    | for x, y in node_similar_pairs do
10   | | model_node ⟵ pair_results[(x, y)]
11   | | mn ⟵ difference(model_node, (x, y))
12   | | if model _node in mns then
13   | | | mns.remove(model_node)
14   | | end
15   | | model_mns[mn].append(model_node)
     | | // ensures we do not encounter the pair again
16   | | similar_pairs.pop((x, y))
17   | end
18   end
     // Remove replacement options for an mn that can be replaced
        themselves
19   for node in nodes do
20   | if model_mns[node].count() > 1 then
21   | | for model_node in model_mns[node] do
22   | | | if model_mns[node] not empty then
23   | | | | model_mns[node].remove(model_node)
24   | | | | mns.append(model_node)
25   | | | end
26   | | end
27   | end
28   end
29 end
```

Fig. 1. Example of a network where the framework is applied. The letters N, D and M followed by a number stands for node, dataset and model respectively.



5 Experimental Evaluation

5.1 Experimental Setup

Datasets We have evaluated our framework for both regression and classification models. For regression, we have used the ***GNFUV Unmanned Surface Vehicles Sensor Data Set*** [4] which includes data from three experiments. In each experiment there are four sets of mobile sensor readings data (humidity and temperature) recorded by the Raspberry Pi's corresponding to four Unmanned Surface Vehicles (USVs) (see Figure 3).

For classification, we have used the ***UCI Bank Marketing Dataset*** (BM) [9]. The data was collected by a banking institution through phone calls as part of a direct marketing campaign. The dataset is a binary classification dataset of classes 'yes' or 'no', to subscribe to the product (bank term deposit). More specifically there are 4640 'yes' instances and 36548 'no' instances.

We have applied Principal Component Analysis (PCA) to reduce the number of dimensions of the dataset from 20 to 3 and then subsequently used these data to execute the hypothesis testing.

Fig. 2. The Result of clustering on the BM Dataset

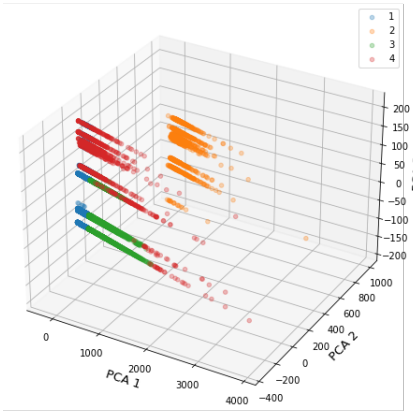
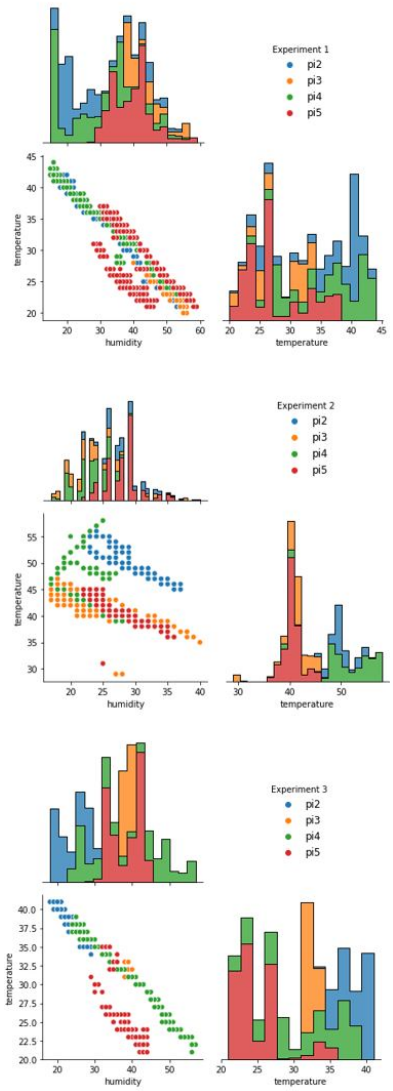


Fig. 3. The relationship between humidity and temperature per experiment alongside their distribution plots for the original GNFUV data



In comparison to the GNFUV dataset the BM dataset has no inherent network-node like structure and hence it was constructed. We trained a K-means classifier with an equal number of yes and no instances to split data into four clusters. This was done to avoid class imbalance from influencing the clustering algorithm. However, we wanted to have more available samples to split into more nodes so instead of clustering equal amounts of instances per class, we used three times the number of yes instances for no instances. We merged three of these clusters into one (clusters 1,2 & 4 in Figure 2) and then created 5 nodes from the two clusters.

It is worth mentioning we have used two data configurations per dataset. For the GNFUV dataset, the two configurations were the original data and a standardised version of them. For the BM Dataset, we used the node data created from the aforementioned process as well as a balanced version of them, by under sampling the majority class (no) to have an equal number of instances as the minority class (yes).

Lastly, we have drawn 100 unique samples per network, in each of which the node data have an equal number of examples in order to comply with the MMD implementation constraint discussed in section 4. The sample size of each node dataset is determined by the Minimum Sample Size (MSS), i.e., defined by the node with the minimum number of entries. The source code is available for re-producability at https://github.com/XeniaSkotti/online_model_reuse_framework_edge_computing.

ASMMD Algorithm Parameters As discussed in Section 4, the ASMMD Algorithm takes four arguments, the sample of each node, the kernel, bandwidth, the similar and other nodes. In this section we discuss how we set and what the kernel, band-

width, similar and other nodes are per dataset (and experiment in the case of the GNFUV dataset).

The approach to identifying the similar and other nodes for each dataset differed due to the nature of each dataset. Since the GNFUV is a regression dataset of only two dimensions, we plotted the points of each experiment and visually identified the pairs which we deemed as similar per experiment. Then we used Algorithms 1 & 2 to confirm our inferences, otherwise we adjusted the similar and other nodes sets. For the BM dataset the similar nodes are either the nodes of the newly merged cluster or cluster 3. Similarly, we tested both possible similar nodes sets for each data configuration (balanced and unbalanced) to determine which one was best.

Table 1. ASMMD Algorithm Parameters per Dataset

Dataset	**Experiment**	**Data Configuration**	**ASMMD Algorithm Parameters**			
			similar_nodes	**other_nodes**	**kernel**	**bandwidth**
GNFUV	1	standardised	pi2, pi3, pi4	pi5	rbf	0.5
		original	pi2, pi4	pi3, pi5	rbf	10
	2	standardised	pi2, pi3, pi5	pi4	rbf	1
		original	pi3, pi5	pi2, pi4	rbf	100
	3	standardised	pi2, pi4	pi3, pi5	rbf	1
		original	pi2, pi4	pi3, pi5	rbf	5
BM		balanced	pi1, pi2, pi3	pi4, pi5	linear	0.001
		unbalanced	pi4, pi5	pi1, pi2, pi3	linear	0.001

Once we had an initial idea of the similar and other nodes sets, we could then use them to determine the kernel and bandwidth. The two kernels we considered were the Radial Basis Function (rbf) and Linear kernels. We aimed to choose the parameters which would most effectively separate the similar from dissimilar pairs. The full parameter configuration of each dataset (experiment) and data configuration is found in Table 1.

ML Models For each problem type we chose distinct classifiers, namely Support Vector Regression (SVR) and Logistic Regression (LR) for regression and classification respectively.

Starting off with regression, we have trained SVRs to capture the relationship between the humidity and temperature attributes of the dataset. SVRs are a version of SVM for regression proposed by Vapnik et al. [2]. SVRs have a few variables that should be optimised for each node model. First, we experiment with both the linear and rbf kernels in order to evaluate how different kernels interact with our framework. Moreover, we optimise the regularization parameter and the epsilon in the epsilon-SVR model using grid search given a node's dataset to ensure we find the best ϵ-insensitive region for the data. It is worth noting that the SVR implementation in scikit-learn reports the performance of the classifier in terms of the coefficient of determination (R^2).

Table 2. Classifier parameter values that are fixed and optimised per dataset

Dataset	Classifier	Classifier Parameters		
		Fixed	**Gird Search Optimised**	
GNFUV	SVR	**kernel**	**C**	**epsilon**
		linear	0.01, 0.1, 1, 10	0.1, 0.5, 1, 2, 5
		non-linear	0.01, 0.1, 1, 10	0.1, 0.5, 1, 2, 5
BM	LR	**class_weight**	**C**	**solver**
		balanced	0.01, 0.1, 1, 10	"lbfgs","liblinear", "saga", "sag"
		None	0.01, 0.1, 1, 10	"lbfgs","liblinear", "saga", "sag"

Our classification dataset, has two classes yes and no and we have used LR [1] specifically because it is usually a good baseline for binary classification. Hence, the scikit-learn implementation of LR reports performance in terms of the mean accuracy on the test dataset. As mentioned in section 5.1, for the BM Dataset we experiment with two data configurations, one which data are balanced and another in which they are not. For the case in which the data are not balanced, we configured the class weight parameter of LR to be balanced to deal with the imbalance. The other parameter which we control for both data configurations is the regularization parameter. Lastly, the scikit learn implementation offers a variety of solver options hence we optimise it as well.

Model Reusability Metrics Investigating the effectiveness of the framework, requires that we examine two aspects, the **speedup** we benefit from when we avoid training models for some nodes in the network, and the **precision** of the framework in terms of the recommendations it makes. We have defined both speedup and precision in the context of model reusability.

Starting off with precision, precision needs to be assessed across three different levels. The precision of MMD at identifying good pairs for reusability, the precision of OCSVM at identifying the correct node to reuse it's model and lastly the combined precision of the framework. In order for the ***MMD precision*** to be a meaningful measure to use, it is expressed in terms of the ratio between the performance of using a proxy model and the true model. We then consider this ratio with regards with a threshold and if it is above that threshold it is correct. The thresholds we considered were 0.8, 0.85 and 0.9 and are extremely high.

The ***OCSVM precision*** is either calculated strictly or with a 0.05 error margin, that is if the node pointed by the direction of reusability does not yield the optimal performance, but it's performance is equal or less than 0.05 from the optimal, we consider that the framework has made the right decision. Then like the MMD precision, we consider the framework made the right decision if the ratio is above a threshold.

For the ***combined precision*** we utilized lower values for the threshold, namely values 0.6 and 0.8 since when the components are combined this will likely result in higher errors. Nevertheless, 0.8 is still not only a high threshold but also it is common threshold across the MMD and combined precision allowing

us to track their difference. The reason that we assess precision across three levels is to be able to gauge how effective each component of the framework is in isolation but also combined. Consequently, we can provide a more holistic evaluation of the framework.

In terms of the speedup, we need to be able to quantify how much time did we save by not training some models with respect to what time we would need if we trained all of them. This requires that we first identify the nodes for which we won't train a model for. As discussed in section 4, as part of our framework we proposed a reusability maximising decision making algorithm (Algorithm 4). The algorithm can provide us with the nodes which we do not need to train a model for, the model-less nodes along with a list of potential replacement models. We utilise the list of of model-less nodes to calculate the speedup. It is worth noting that the speedup potential varies across datasets and samples hence we simply report it as a number.

5.2 Performance Evaluation

As discussed in the previous section (5.1), we assess the framework across two metrics, precision and speedup. In this section we evaluate these metric results one by one for each dataset and provide a discussion around the effectiveness of the framework.

In the following sections we discuss the precision results across the three levels, followed by speedup. We will analyse each dataset's precision individually and then discuss the speedup across both datasets simultaneously. More specifically, in the case of the GNFUV dataset precision, we will provide observations for each experiment before drawing general conclusions using the metric results. Finally, we will draw some general conclusions on the applicability of the framework in regression, the effect of the kernel and standardisation of data. Similarly, for the BM dataset precision we will draw conclusions for the dataset, the applicability of the framework in classification and the effect of using balanced and unbalanced data.

Regression Precision *Original Data:* Starting off with the GNFUV original data, the combined precision is almost 1 for Experiment 1 and Experiment 3 if we allow a 0.05 margin of tolerance in terms of the OCSVM predictions (non-strict - as discussed in Section 5.1). The combined precision falls to 0.69 when we are strict about the predictions because of Experiment 3. The combined precision for Experiment 2 is low but that's expected considering what we discussed above. Therefore, the framework, when the threshold is set to 0.8, has a combined precision of 0.59 with no tolerance and increases to 0.77 when there is. These results are illustrated in Table 3. If we analyse combined precision per kernel, the linear kernel is better suited for original data across all three experiments. Similar trends to those discussed either when we do or do not distinguish per kernel, can be found in the MMD precision and OCSVM precision (Table 4), with MMD precision at 0.78 when the threshold is 0.8 and OCSVM precision at

0.79 when we are strict and 0.97 when we are not. It is wroth noting that the OCSVM precision for Experiment 2 when the kernel is linear is as high (almost 1) as for the other two experiments which illustrates the importance of the kernel choice. Upon further analysis linear results yield the best results on average for the original GNFUV data and hence the framework's high precision overall.

Table 3. GNFUV Data combined precision Results.

Data Configuration	**combined precision**			
	Experiment	**Threshold**	**Strict**	
			True	**False**
original	1	0.6	1.00	1.00
		0.8	1.00	1.00
	2	0.6	0.89	0.90
		0.8	0.46	0.47
	3	0.6	0.38	0.98
		0.8	0.38	0.98
	Weighted Average	0.6	0.77	0.95
		0.8	0.59	0.77
standardised	**Experiment**			
	1	0.6	0.39	0.80
		0.8	0.39	0.80
	2	0.6	0.27	0.29
		0.8	0.25	0.25
	3	0.6	1.00	1.00
		0.8	1.00	1.00
	Weighted Average	0.6	0.46	0.63
		0.8	0.45	0.61

Standardised Data: The comments made previously for the combined precision of Experiment 3 when we are strict cease to be true and are instead true for Experiment 1 and the combined precision is low. Nevertheless, similarly to the original data the combined precision is extremely high for Experiment 1 & 3 we are not strict with OCSVM. The Experiment 2 combined precision is almost half what it is for the original data at the 0.8 threshold. Consequently, the overall combined precision of the framework drops at the threshold level 0.8, to 0.45 and 0.61 when we are strict and non-strict respectively (Table 3). Contrary to the original data where the combined precision per kernel showed that the linear kernel is better suited, for the standardised data the opposite is true, while this difference is not significant. This is also true for the OCSVM precision when analysed per kernel. Overall, the OCSVM precision for Experiment 2 drops (Table 4), hence the OCSVM weighted average precision across experiments drops by 30%. On the other hand, MMD precision increases slightly by %4 due to an increase in the precision of Experiment 2. Upon further analysis per kernel, the MMD precision increased 15% per kernel with the linear kernel providing much better results.

Table 4. GNFUV Data OCSVM precision Results.

Data Configuration	**OCSVM precision**		
original	**Experiment**	**Strict**	
		True	**False**
	1	1.00	1.00
	2	0.94	0.95
	3	0.38	0.98
	Weighted Average	0.79	0.97
standardised	**Experiment**		
	1	0.39	0.80
	2	0.29	0.33
	3	1.00	1.00
	Weighted Average	0.47	0.65

GNFUV Precision Performance Overall: Overall, the MMD precision of the framework is high, however it is low for Experiment 2 across both original and standardised data. The difference between the MMD precision of original and standardised is not high when the threshold is set at 0.8 (only at 4%). However as the threshold increases this difference as well, to 10% and 13% for thresholds 0.85 and 0.9 respectively. This is because the performance on Experiment 2 is better on standardised data and as the threshold increases it does not deteriorate in the same way as for the original data. Considering how high of a threshold 0.9 is, having a 0.63 and 0.76 MMD precision is really good performance. The kernel choice is not important for Experiment 1, when it comes to the MMD precision since the performance is perfect regardless of the kernel and data configuration. This is also true for Experiment 3 for the standardised data, while for the original data the linear kernel is better suited for this experiment. For both data configurations the linear kernel performs better for Experiment 2. Lastly, in terms of the OVSVM Precision when the original data are used, the kernel choice is unimportant for Experiment 1, while for Experiments 2 & 3 the linear kernel is better, even though for Experiment 3 the difference is not significant. When the standardised data are used, the statements made for Experiments 1 & 3 are now reversed, with the slight difference that it is the rbf kernel that is better for Experiment 1 instead of the linear one. Similarly, to Experiment 1 the rbf kernel is slightly better for Experiment 2 but the difference is only 0.07.

The framework performs better on the original data across all three levels of precision with 0.77 (non-strict) combined precision at threshold 0.8 and a drop of 15% for standardised ones. When analysing the combined precision results per kernel, the linear kernel is better suited for the original data, while the opposite is true for standardised data even though this difference is not large. The rbf kernel models have higher performance on their native datasets compared to linear ones, but nevertheless have higher discrepancy. Hence, on average linear models provide better results.

Table 5. BM Data combined precision Results.

Data Configuration	combined precision		
	Threshold	Strict	
		True	False
combined	0.6	0.55	0.98
	0.8	0.55	0.98
balanced	0.6	0.58	0.99
	0.8	0.58	0.99
unbalanced	0.6	0.56	1.00
	0.8	0.56	1.00

Table 6. BM Dataset OCSVM precision Results.

Data Configuration	OCSVM precision	
	Strict	
	True	False
combined	0.55	0.98
balanced	0.58	0.99
unbalanced	0.56	1.00

Classification Precision In terms of the classification performance of the dataset, the BM Dataset results are very good. All the nodes in the BM Dataset, have good performance on their native dataset, with balanced models having slightly better performance. All pairs identified are good pairs for reusability on both sides and the performance across configurations is almost identical. This is confirmed by the combined precision depicted in Table 5 which is extremely high regardless of whether we distinguish between the configurations or not if we are not strict. If we are strict this performance drops at 0.55 on average and this is a direct reflection of the OCSVM precision (Table 6). However, considering how good the performance is overall, the real combined precision of the framework is the one given by the non-strict measure.

Table 7. Framework Speedup Results.

Dataset	Speedup		
	Experiment	Data Configuration	
		standardised	original
GNFUV	1	0.23	0.26
	2	0.3	0.28
	3	0.24	0.23
	Weighted Average	0.26	0.26
BM		unbalanced	balanced
		0.29	0.41

Speedup Overall, the speedup of the framework for the particular datasets used for regression and classification are 26%, and 29% to 41% respectively (Table 7). These results are expected if you consider that for the GNFUV dataset regardless of the data configuration on average there is one good pair for reusability hence one node's model is not trained. The two data clusters created from the BM dataset mean that ideally we would only train two models. Nevertheless the results are lower than this average case due to the fact we use samples of the dataset hence the true reusability differs from sample to sample. Hence, for both the classification and regression case we can argue that the framework if effective in identifying the true number similar pairs.

6 Discussion & Conclusions

In this paper, we presented a novel online model reuse framework in edge computing. The framework considers all possible pairs of nodes in the network and infers which are good reusability pairs as well as which of the two nodes' model can be used as a replacement model for the other per pair. We utilise MMD as our dataset similarity measure and we present a newly defined algorithm which calculates a threshold that distinguishes similar from non-similar pairs. The node model that is chosen to be reused in each pair is the one with the highest inlier data space overlap. Experiments in the context of both regression and classification have shown the framework achieves good precision. Lastly, we present an algorithm that, given the results of the framework, can maximise the number of nodes which use reused models along with a list of potential replacement models.

The framework presented is novel and therefore the results presented in this paper while encouraging they are still preliminary. We experimented with only one model per data domain and a limited range of data configurations. Consequently, the evaluation of the framework needs to be extended to check the compatibility with more domain models and data configurations. Even though this framework in its current does not preserve user privacy it could be amended to meet this requirement. In this paper, we hypothesise that the inlier space overlap is an indicator for the direction of reusability. However, we only consider one outlier detection model and there many more that could be used. Furthermore, the naive decision making algorithm proposed as part of the framework is maximising the speedup, which does not guarantee that the solution is optimal performance wise. Defining an algorithm which can produce either the performance optimal or partially optimal solution is a different and challenging task altogether.

Acknowledgement: This research has received funding from the European Union's Horizon 2020 research and innovation programme under Grant Agreement no. 101037247.

References

1. Cramer, J. The origins of logistic regression. *Tinbergen Institute, Tinbergen Institute Discussion Papers* (01 2002).
2. Drucker, H., Burges, C. J. C., Kaufman, L., Smola, A., and Vapnik, V. Support vector regression machines. In *Proceedings of the 9th International Conference on Neural Information Processing Systems* (Cambridge, MA, USA, 1996), NIPS'96, MIT Press, p. 155–161.
3. Gretton, A., Borgwardt, K. M., Rasch, M. J., Schölkopf, B., and Smola, A. A kernel two-sample test. *J. Mach. Learn. Res. 13* (Mar. 2012), 723–773.
4. Harth, N., and Anagnostopoulos, C. Edge-centric efficient regression analytics. In *2018 IEEE International Conference on Edge Computing (EDGE)* (2018), pp. 93–100.
5. Hasani, S., Thirumuruganathan, S., Asudeh, A., Koudas, N., and Das, G. Efficient construction of approximate ad-hoc ml models through materialization and reuse. *Proc. VLDB Endow. 11*, 11 (July 2018), 1468–1481.
6. Lee, J., Mtibaa, A., and Mastorakis, S. A case for compute reuse in future edge systems: An empirical study. In *2019 IEEE Globecom Workshops (GC Wkshps)* (2019), pp. 1–6.
7. Li, C., Huang, S., Liu, Y., and Zhang, Z. Distributed jointly sparse multitask learning over networks. *IEEE Transactions on Cybernetics 48*, 1 (2018), 151–164.
8. Li, Y., Zhang, Z., Liu, B., Yang, Z., and Liu, Y. Modeldiff: Testing-based dnn similarity comparison for model reuse detection. In *Proceedings of the 30th ACM SIGSOFT International Symposium on Software Testing and Analysis* (New York, NY, USA, 2021), ISSTA 2021, Association for Computing Machinery, p. 139–151.
9. Moro, S., Cortez, P., and Rita, P. A data-driven approach to predict the success of bank telemarketing. *Decision Support Systems 62* (2014), 22–31.
10. Schölkopf, B., Williamson, R., Smola, A., Shawe-Taylor, J., and Platt, J. Support vector method for novelty detection. In *Proceedings of the 12th International Conference on Neural Information Processing Systems* (Cambridge, MA, USA, 1999), NIPS'99, MIT Press, p. 582–588.
11. WEI, Y., Zhang, Y., Huang, J., and Yang, Q. Transfer learning via learning to transfer. In *Proceedings of the 35th International Conference on Machine Learning* (10–15 Jul 2018), J. Dy and A. Krause, Eds., vol. 80 of *Proceedings of Machine Learning Research*, PMLR, pp. 5085–5094.
12. Wu, X., Xu, W., Liu, S., and Zhou, Z. Model reuse with reduced kernel mean embedding specification. *CoRR abs/2001.07135* (2020).
13. Zhao, P., Cai, L., and Zhou, Z. Handling concept drift via model reuse. *CoRR abs/1809.02804* (2018).
14. Zhou, Z.-H. Learnware: on the future of machine learning. *Frontiers of Computer Science 10* (06 2016).