Machine Learning Models Re-usability in Edge Computing Environments

Xenia Skotti (2299606s)

April 5, 2022

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 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 similar by adopting statistical learning methods. Then for each pair, the node model is chosen that has the highest inlier data space overlap. We further propose a decision making algorithm to unify information from the pair level to the network level. Our extensive experimental analysis in the context of both regression and classification shows the feasibility our solution in model reusability in Edge Computing environments.

1. INTRODUCTION

The combination of the rapid expansion of the Internet of Things (IoT) and the success of cloud computing services have contributed to the emergence of a new computing paradigm, edge computing [15]. The paradigm essentially calls for processing data at the edge of the network as a direct consequence of data being increasingly produced at the edge of the network making these nodes both data producers and consumers. Hence, it is more efficient to process the data at the edge of the network as well. Some of the benefits of edge computing compared to traditional cloud-based computing paradigm include: reduced response time for requests [5, 20], reduced energy consumption [2], reduced latency. Consequently, edge computing opportunities have been recognised in a variety of contexts such as smart homes and smart cities.

As the adoption of edge computing continues to grow and edge nodes record increasingly more data, the need to analyse information through machine learning (ML) methods [1, 16] becomes more and more prevalent. Nevertheless, the main challenges of ML include increased amounts of computation, energy consumption and storage [17]. One of the ways to reduce the burden on the network overall, is to reduce the amount of models needed to be trained in the first place by reusing existing models, a form of compute reuse.

Lee et. al [8] 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 [22] 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 [21] and building ad-hoc analytic models [7].

Model reusability is compelling and, therefore, both theoretical [22] and empirical [7, 19] 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 phases 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) [4] statistic does.

In addition to measuring the similarity between two datasets, we need to determine the direction of reusability. In other frameworks [7, 19], 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) [14] 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, the definitions of metrics in the context of model reusability and an extensive performance evaluation of the framework in terms of regression and precision. 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 [8] to quantify its gain. Experiments on three applications: matrix multiplication, face detection, and chess, showed that systems that adopt compute reuse, as opposed to those that do not, can finish the same task up to five times faster. In addition to the benefits of compute reuse they also highlight some challenges including task representation and privacy considerations. Model tasks need to have a clear specification detailing their purpose and speciality in order to identify the context in which they can be re-used. Simultaneously, when the model is shared, user privacy should be preserved. Motivated by similar concerns a theoretical paradigm named learnware was proposed by Zhou [22]. More specifically, a learnware is a machine learning 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 data scientists to identify 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) [19] was presented, 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. Roughly speaking, a kernel mean embedding is a point in the reproducing Hilbert space (RKHS) which "summarises" the probability distribution. 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. Therefore, there is no need to access the raw data since KME acts a proxy for them. In essence, the RKME's deployment phase, is similar to the MMD statistic [4], 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. The framework was tested in a series of experiments including a real-world project where it outperformed reuse baselines in terms of the root-mean-square error.

The author of the learnware paradigm [22] recognises transfer learning as a preliminary attempt to reusability. The aim of transfer learning is to transfer the knowledge of a pretrained model to a new model that is used for a different but related problem. In transfer learning there are three key research issues as identified in [13]: when, how and what to transfer. This corresponds to identifying a source domain that would benefit the target domain and then using an algorithm the transferable knowledge across domains is discovered. A two-stage framework dubbed as Learning to Transfer (L2T) was presented [18], 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: a pair of source and target domains, the transferred knowledge between them represented by latent factors and, the performance improvement ratio. Using these transfer learning experiences, L2T learns 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 further highlighting the similarity to RKME [19]. 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. A potential drawback of the RKME [19] framework, and by extension the learnware paradigm, is that the variance between pairs cannot be calculated since the raw data are not available during the testing phase. 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, a situation where the distribution of the data (usually stream data) changes. The assumption that previous data contain some useful information, indicates that the models corresponding to the data can be leveraged. Condor was proposed [21] as an approach to handling concept drift through model reuse. Condor consists of two modules, ModelUpdate and WeightUpdate which leverage previous knowledge to build new model, hence updating the model pool and adapting the weights of previous models to reflect current reusability performance respectively. The effectiveness of the approach was validated using both synthetic and real-world datasets.

Hasani et al. [7] proposed a two-phased approach, to build faster models for a popular class of analytic queries by leveraging model reuse. Similar to the other approaches [18, 19, 21], 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.

Moreover, they propose two methods for generating approximate models, one which is extremely fast but does not provide a fine-tuning option and another which does at the cost of efficiency. 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 [10], 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. In contrast to RKME [19] which is a compute reuse framework, ModelDiff is a model reuse detector.

Lee et al. [8] 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 [9]. 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.

In conclusion, reusing models results in significant reduction in compute usage resources. Both theoretical and empirical frameworks have been proposed to take advantage of the performance improvement afforded through model reusability. Nevertheless, model reuse has also been used to tackle concept drift and building ad-hoc analytic models. While model reuse is undoubtedly beneficial many have raised concerns including user privacy and intellectual property considerations. These are legitimate concerns of model sharing, however our model reuse framework is novel and therefore user privacy is not a concern at this stage of development. In the future we could amend the framework to not expose any data outside of the node. At this stage we have investigated whether the framework is feasible.

Contribution: 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 detector, based on the inlier data space overlap.
- A decision making algorithm, which given the results of the framework, it maximises 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

In this section, we elaborate on the two statistical learning methods MMD and OCSVM adopted for the model reusability framework.

3.1 Maximum Mean Discrepancy

The MMD is a statistic that quantifies the mean discrepancy of two data distributions in a kernel space in order to determine if two samples are drawn from different distributions [4]. Let p and q be two independent probability distributions, and $E_x\left[f\left(x\right)\right]$ (shorthand notation for $E_{x\sim p}\left[f\left(x\right)\right]$) denotes the mathematical expectation of $f\left(x\right)$ with x under the probability density p. The MMD statistic between p and q is defined as:

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

where the function class $\boldsymbol{\mathcal{F}}$ is a unit ball in the reproducing Hilbert space (RKHS) ($\boldsymbol{\mathcal{H}}$) 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:

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

where $k(\cdot,\cdot)$ 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 $k(x, y) = \exp\left(-\frac{1}{2\sigma^2} \parallel x - y \parallel^2\right)$, 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 an 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 [14] 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 (i.e., inliers) and -1 elsewhere (i.e., outliers). This function f is called a decision function being defined as:

$$f(x) = sign(g(x)) = sign(w^\top \phi(x) - \rho) \tag{3}$$

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.,

$$\min_{w,\xi\in\mathbb{R}^N,\rho\in\mathbb{R}} \frac{\| w \|^2}{2} - \rho + \frac{1}{\nu n}\sum_{i=1}^{n}\xi_i \tag{4}$$

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

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 $\nu \in (0, 1)$ is a regularization parameter. As shown in (4) 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. ν represents the upper bound limit of the fraction of outliers and a lower bound on the number of support vectors. In other words, ν specifies the number of training points which are guaranteed to be misclassified and the number of training examples being support vectors. As mentioned above $\nu \in (0, 1)$ and therefore a percentage, where a high value may lead to over-fitting and a low value to under-fitting. ν 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:

$$\min_{a} \frac{a^T Q a}{2} \tag{5}$$

$$\text{subject to: } 0 \leq a_i \leq \frac{1}{\nu n}, \sum_{i=1}^{n} a_i = 1$$

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

$$f(x) = sign(\sum_{i=1}^{n} a_i k(x, x_i)). \tag{6}$$

4. 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 between 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: Average Similarity MMD (ASMMD) between Nodes' Datasets.

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: $\boldsymbol{ASMMD}$

```
begin
  // Calculating the baseline ASMMD
  similar_mmds ⟵ []
  for x, y in get_pair_combos(similar_nodes) do
    sx ⟵ samples[x], sy ⟵ samples[y]
    mmd ⟵ 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 ⟵ samples[x]
    for y in similar_nodes do
      sy ⟵ samples[y]
      mmd ⟵ MMD(sx, sy, kernel, bandwidth)
      asmmd ⟵ mean(similar_mmds)
      if mmd < (asmmd + 1) * 0.05 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 ⟵ samples[x], sy ⟵ samples[y]
      mmd ⟵ MMD(sx, sy, kernel, bandwidth)
      asmmd ⟵ mean(similar_mmds)
      if mmd < (asmmd + 1) * 0.05 then
        similar_mmds.append(mmd)
      end
    end
  end
  asmmd = mean(similar_mmds)
end
```

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 5.1.2.

Algorithm 2: Similar Dataset Pairs Detector

```
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) * 0.05 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: Direction of Reusability Detector

```
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.

Algorithm 4: Reusability Maximizing Decision Making Algorithm

```
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
begin
  similar_pairs ⟵ pair_results.keys()
  mns ⟵ nodes.copy(), model_mns ⟵ {}
  for node in nodes do
    model_mns[node] ⟵ []
  end
  for node in nodes do
    node_similar_pairs ⟵
    get_node_similar_pairs(node, similar_pairs)
    for x, y in node_similar_pairs do
      model_node ⟵ pair_results[(x, y)]
      mn ⟵ difference(model_node, (x, y))
      if model_node in mns then
        mns.remove(model_node)
      end
      model_mns[mn].append(model_node)
      // ensures we do not encounter the pair
         again
      similar_pairs.pop((x, y))
    end
  end
  // Remove replacement options for an mn
     that can be replaced themselves
  for node in nodes do
    if model_mns[node].count() > 1 then
      for model_node in model_mns[node] do
        if model_mns[node] not empty then
          model_mns[node].remove(model_node)
          mns.append(model_node)
        end
      end
    end
  end
end
```

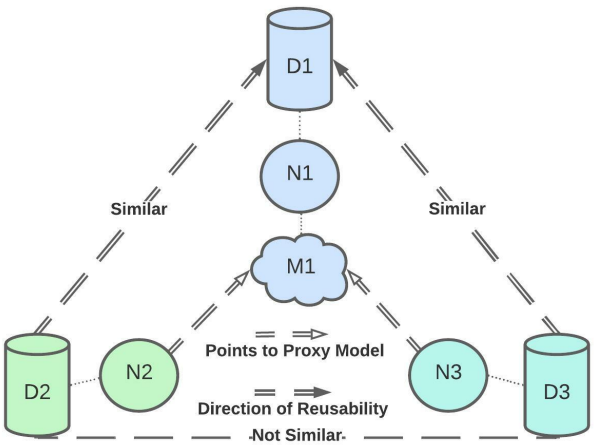


Figure 1: Example of an edge network adopting our framework. The letters N, D and M followed by a number stands for node, dataset and model, respectively.

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's 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.

5. EXPERIMENTAL EVALUATION

5.1 Experimental Setup

5.1.1 Datasets

We have evaluated our framework for both regression and classification.

Regression: We have used the ***GNFUV Unmanned Surface Vehicles Sensor Data Set*** [6] which includes data from three experiments. In each experiment there are four sets of mobile sensor readings data recorded by the Raspberry Pi's corresponding to four Unmanned Surface Vehicles (USVs). Each node (USV) dataset contains the humidity and temperature recorded when they were floating over the sea surface in a GPS pre-defined trajectory in a coastal area of Athens (Greece). The data description and visualisation can be found in Appendix A Table A.1 and Figure 2 respectively.

Classification: We have used the ***UCI Bank Marketing Dataset*** (BM) [12] which consists of four files of which we have used the bank-additional-full.csv one which contains 41188 records and 20 attributes. 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). To get a yes from a client often more than one contact was required hence class imbalance is present. More specifically there are 4640 yes instances and

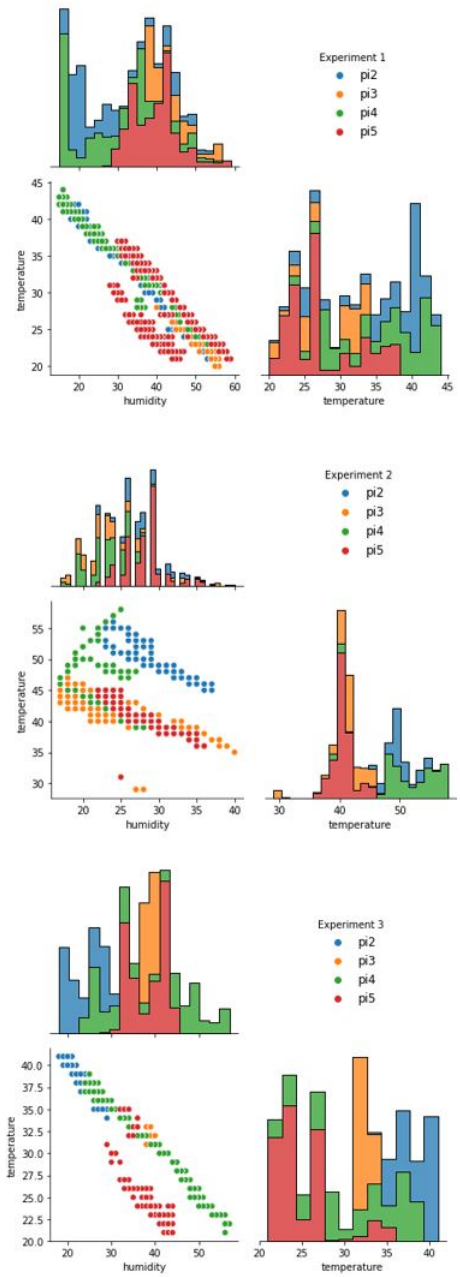


Figure 2: The relationship between humidity and temperature per experiment alongside their distribution plots for the original GNFUV data.

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

Table 1: ASMMD Algorithm Parameters per Dataset

36548 no instances.

As mentioned above the dataset has 20 attributes hence we have applied Principal Component Analysis (PCA) to reduce the number of dimensions. We have transformed data through PCA and kept 3 of the 20 dimensions and then subsequently used these data to execute the hypothesis testing. 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 (9280 total) 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 (4640 (yes) 13920 (no)). We left cluster 3 shown in Figure 3, which was large enough and has a good class balance, untouched and merged the other three clusters together to create a larger cluster. From these clusters we then created 5 nodes in total and the node data description can be found in Appendix A Table A.2.

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 (see Appendix A Figure A.1 for the data visualisation). While for the BM Dataset we used the node data created from the process above 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). In each case this was done to explore the effect of the data configuration change on the effectiveness of the framework.

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) of the network i.e. the node with the minimum number of entries. If the two thirds of the MSS is more than 500 instances the sample size is the two-thirds of the MSS, otherwise it is 500, unless the MSS is less than that in which case the sample size is equal to it. Even though for some nodes we are not really creating a sample but simply taking all or most the points of the node, the remainder of the nodes will be samples and therefore we are testing a different relationship between nodes with each sample and hence we do not compromise the validity of our experiment.

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

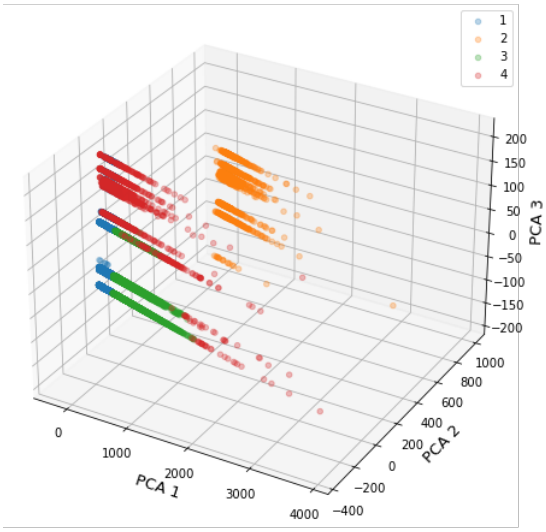


Figure 3: The result of clustering on the BM Dataset projected over the three PCs.

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 (see Figure 3 for a visual of each original cluster). Similarly, we tested both possible similar nodes sets for each data configuration (balanced and unbalanced) to determine which one was best.

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.

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"

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

5.1.3 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. [3]. The adaptation is accomplished by introducing an ϵ-insensitive region around the function, called the ϵ-tube as shown in Appendix B Figure B.1. The goal is to first minimise an ϵ-insensitive loss function and find the flattest tube that contains most of the training instances. Therefore, 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, defined as:

$$R^2 = 1 - \frac{\sum_{i=1}^{n}(y_i - \hat{y}_i)^2}{\sum_{i=1}^{n}(y_i - \bar{y})^2}, \tag{7}$$

where y_i, $\hat{y}_i$ and $\bar{y}$ are the actual, predicted and average values, respectively.

Our classification dataset, has two classes 'yes' and 'no' and we have used LR [11] specifically because it is usually a good baseline for binary classification. LR [11] is a statistical model that in its basic form uses a logistic function to model the conditional probability, of two classes in our case. Hence, the scikit-learn implementation of LR reports performance in terms of the mean accuracy on the test dataset. As mentioned in Section 5.1.1, the BM Dataset exhibits severe class imbalance which is why 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. Table 2 summarises all of the information discussed in the section for both classifiers.

5.1.4 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.

Before, we continue to define precision and speedup it is important to mention first how we assess the proxy model's performance and how we measure a model's training time. We assess the proxy model on the non-native node data by subtracting the **discrepancy** from the model's performance score on the native dataset. We define discrepancy as the difference between the model's performance score on the native dataset and the model's performance score on the non-native dataset. Therefore, essentially we simply use the performance of the model on the non-native dataset. The model's training time is the sum of the time required to train and optimise the model. The reason that we include the optimisation time is that some form of optimisation would take place in a real world scenario.

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 its 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. It is worth mentioning, that the ratio retains the same meaning across regression and classification since it measures how close the proxy model is to the true performance regardless of whether the performance is reported in terms of R^2 or accuracy. 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. Assuming the true performance of the model is 0.8 and we choose to use the lowest threshold (0.8) then the performance on the non-native model should not be lower than 0.64.

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 its performance is equal or less than 0.05 from the optimal, we consider that the framework has made the right decision.

This is the first step of the ***combined precision***, and then like the MMD precision, we consider the framework made the right decision if the ratio is above a threshold it is correct. 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

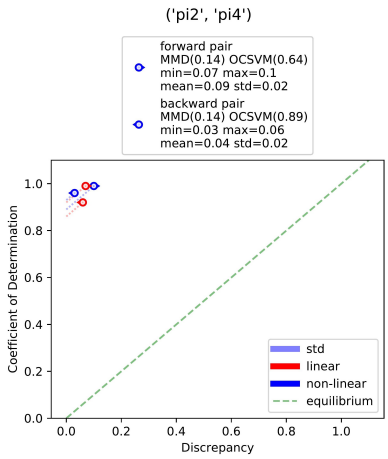


Figure 4: GNFUV Experiment 1 Original Data Results. Both nodes in the pair have good models and both of them could be used as a replacement model for the other, since the $R^2 - discrepancy$ is well above the equilibrium line. The rbf pi4 model is the best for this pair.

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 will not train a model for. As discussed in Section 4, as part of our framework we proposed a 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 (5.1.4), 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.

We discuss the precision results across the three levels, followed by speedup, by looking at a combination of tables and figures. The figures have been created by merging (averaging) the results of the 100 samples to create a single figure for each pair per data configuration (and experiment). 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 pair by pair for each experiment before drawing general conclusions using both the metric results and figures. Finally, we will draw some general conclusions on the applicability of the framework in regression, the effect of the kernel and whether or not we standardised data. Similarly, for the BM dataset precision we will discuss observations per by pair before drawing conclusions for the

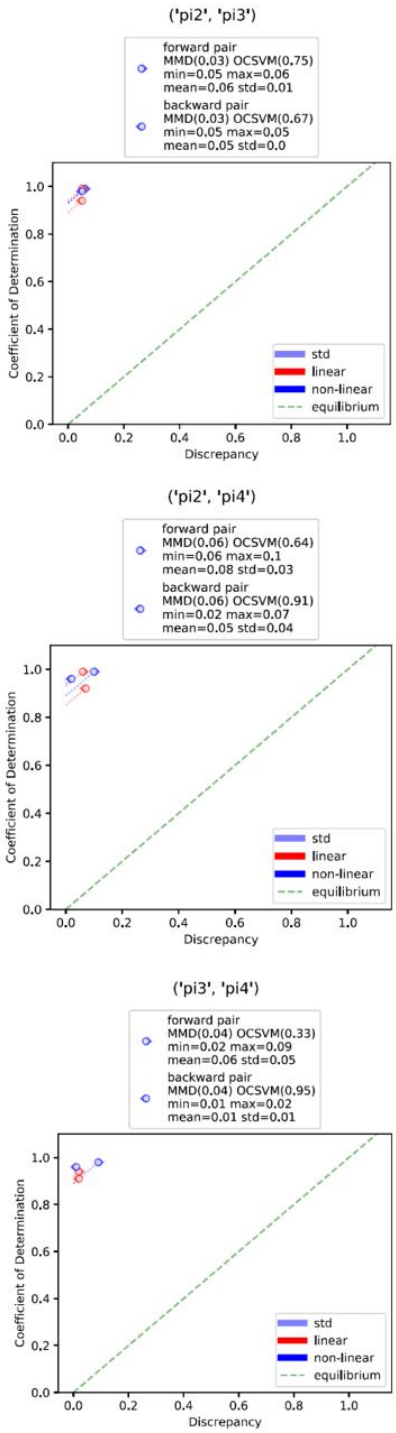


Figure 5: GNFUV Experiment 1 Standardised Data Results. All three pairs are good reusability pairs on both sides and the nodes of each pair have good performance on their native datasets. Hence, each node's model could be used as a replacement model for the other. Overall, the pairs have high OCSVM scores with the exception of the low pi3 score when paired with pi4. Rbf models perform better on their native dataset but have higher discrepancy.

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

Table 3: GNFUV Data combined precision Results. Overall, the framework has good combined precision, but it performs better on the original data. Precision for the original data can get as high as 0.77 at threshold 0.8 and 15% less for standardised ones.

dataset, the applicability of the framework in classification and the effect of using balanced and unbalanced data.

5.2.1 Regression Precision

Original Data: Starting off with the GNFUV original data, for Experiments 1, 2 & 3 the framework identifies the pair (pi2, pi4) for all of them and an additional pair, (pi3, pi5) for Experiment 2. As shown in Figure 4 (and confirmed by Table C.1) for Experiment 1, both nodes have good models and also high OCSVM scores. Both node models could be used as replacement models for the other, since the $R^2 - discrepancy$ is well above the equilibrium line. Overall, the pi4 rbf model, however is the best for this pair. For Experiment 2, neither of these pairs is a good pair for reusability (see Figure C.2). Even though for pair (pi3, pi5) we have instances models above the equilibrium line, the results are still not good. There is a high difference between the OCSVM scores of the nodes in both pairs. For both Experiment 1 & 2, depending on the direction of reusability i.e. which node's model we decide to reuse, different kernels are appropriate. These differences are small for Experiment 1 & 2 this choice is not important because ultimately the pairs are not good options. Similarly to Experiment 1, the pair in Experiment 3 (see Figure C.3) both nodes have good models and could be used as replacement models for the other provided the kernel is linear for reasons discussed for Experiment 1 and the linear pi4 model is the best for this pair. Across all three experiments rbf models perform better on the native dataset but nonetheless have higher discrepancy.

The above observations are reflected in the precision measures. 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 5.1.4. 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 is expected considering what we discussed above. Therefore, the framework has a combined precision when the threshold is set to 0.8, at 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 as shown in Table C.2, 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 (Tables C.3 & C.5) and OCSVM precision (Tables 4 & C.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.

Standardised Data: Moving on to the GNFUV standardised data, the framework identifies three, four and one pair for Experiments 1, 2 & 3 respectively. All three pairs for Experiment 1 are good reusability pairs on both sides and the nodes of each pair have good performance on their native datasets (see Figure 5). Similarly to the original Experiment 1 data, each node's model could be used as a replacement model for the other. Overall, the pairs have high OCSVM scores with the exception of the low pi3 score when paired with pi4. None of the pairs identified for Experiment 2 are good options for reusability despite the fact that most of them are above the equilibrium line. There is a high difference between OCSVM scores of each node in the pair with the exception of the pair (pi3, pi5). The same observations noted for the original data of Experiment 1 can be made for the Experiment 3 the only difference would be the best performing node. For both Experiment 1 & 2, depending on the direction of reusability different kernels are appropriate, but for the same reasons discussed for the non-standardised data this is not important for neither of the experiments. For all three experiments, depending on the direction of reusability, different kernels are appropriate but for the same reasons discussed for the non-standardised data this is not impor-

Data Configuration	OCSVM precision		
	Experiment	Strict	
		True	False
original	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

Table 4: GNFUV Data OCSVM precision Results. The framework correctly identifies the direction of reusability the OCSVM precision being as high as 0.97 for the original data and 65% for the standardised data.

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 5: BM Data combined precision Results. The combined precision for the BM Dataset is extremely high whether we distinguish the data configurations or whether we do not. This indicates the framework has good performance across both of its components and that it is suitable for the classification setting.

tant for neither of the experiments. The same is true for rbf models on standardised data as with the original data in terms of the performance on the native dataset and the discrepancy.

As with the original data, the above observations are reflected in the precision measures. 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 (Table C.4). Overall, 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 (Table C.3). Upon further analysis per kernel, the MMD precision increased 15% per kernel with the linear kernel providing much better results (Table C.5).

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 increases 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, nevertheless they have higher discrepancy. Hence, on average linear models provide better results.

5.2.2 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. The MMD precision is perfect across configurations which is expected as shown in Figure 6.

5.2.3 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 ex-

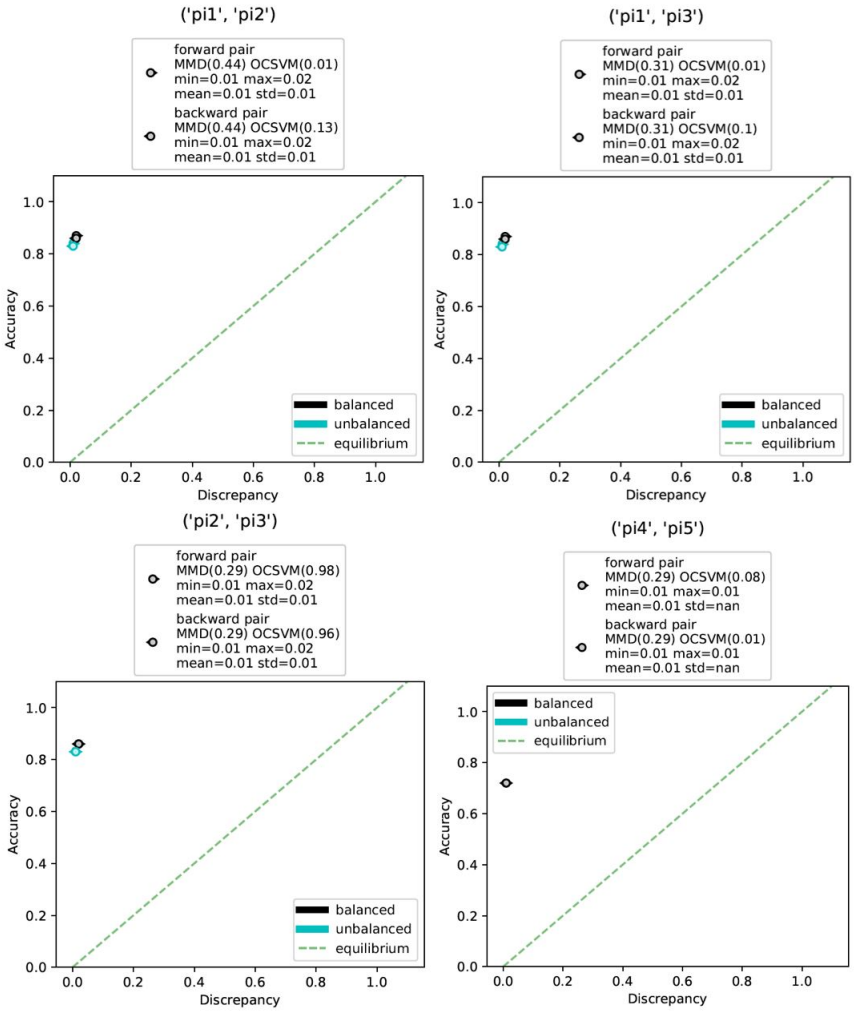


Figure 6: BM Data Results. In all the pairs detected for the BM dataset, the nodes have good models and both nodes' models in the pair could be used as replacement models for the other. The OCSVM scores are low for all pairs with the exception of the pair (pi2, pi3). The performance of unbalanced and balanced configuration is almost identical across pairs, however it seems that balanced datasets perform better on their native dataset.

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

Table 6: BM Dataset OCSVM precision Results. The strict OCSVM precision is low, however the non-strict one is almost 1 showing the effectiveness of the component on the classification dataset.

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

Table 7: Framework Speedup Results. The average regression speedup is 26% and 29% to 41% for classification depending on the data configuration.

pected 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. If we apply this to the results presented in Figure 6 this holds. 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 and maximising the speedup.

6. 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 a naive 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 source code is available in the Github repository named online_model_reuse_framework_edge_computing.

7. LIMITATIONS & FUTURE WORK

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 form 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 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.

Acknowledgments. This work is done with the supervision of Christos Anagnostopoulos whose guidance was crucial for the development of the framework.

8. REFERENCES

[1] Chen, J., and Ran, X. Deep learning with edge computing: A review. *Proceedings of the IEEE 107*, 8 (2019), 1655–1674.

[2] Chun, B.-G., Ihm, S., Maniatis, P., Naik, M., and Patti, A. Clonecloud: Elastic execution between mobile device and cloud. In *Proceedings of the Sixth Conference on Computer Systems* (New York, NY, USA, 2011), EuroSys '11, Association for Computing Machinery, p. 301–314.

[3] 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.

[4] 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.

[5] Ha, K., Chen, Z., Hu, W., Richter, W., Pillai, P., and Satyanarayanan, M. Towards wearable cognitive assistance. MobiSys '14, Association for Computing Machinery, p. 68–81.

[6] Harth, N., and Anagnostopoulos, C. Edge-centric efficient regression analytics. In *2018 IEEE International Conference on Edge Computing (EDGE)* (2018), pp. 93–100.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] Morgan, S. P., and Teachman, J. D. Logistic regression: Description, examples, and comparisons. *Journal of Marriage and Family 50*, 4 (1988), 929–936.

[12] 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.

[13] Pan, S. J., and Yang, Q. A survey on transfer learning. *IEEE Transactions on Knowledge and Data Engineering 22*, 10 (2010), 1345–1359.

[14] 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.

[15] Shi, W., Cao, J., Zhang, Q., Li, Y., and Xu, L. Edge computing: Vision and challenges. *IEEE Internet of Things Journal 3*, 5 (2016), 637–646.

[16] Sun, W., Liu, J., and Yue, Y. Ai-enhanced offloading in edge computing: When machine learning meets industrial iot. *IEEE Network 33*, 5 (2019),

68–74.

[17] Sze, V., Chen, Y.-H., Emer, J., Suleiman, A., and Zhang, Z. Hardware for machine learning: Challenges and opportunities. In *2017 IEEE Custom Integrated Circuits Conference (CICC)* (2017), pp. 1–8.

[18] 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.

[19] Wu, X., Xu, W., Liu, S., and Zhou, Z. Model reuse with reduced kernel mean embedding specification. *CoRR abs/2001.07135* (2020).

[20] Yi, S., Hao, Z., Qin, Z., and Li, Q. Fog computing: Platform and applications. In *2015 Third IEEE Workshop on Hot Topics in Web Systems and Technologies (HotWeb)* (2015), pp. 73–78.

[21] Zhao, P., Cai, L., and Zhou, Z. Handling concept drift via model reuse. *CoRR abs/1809.02804* (2018).

[22] Zhou, Z.-H. Learnware: on the future of machine learning. *Frontiers of Computer Science 10* (06 2016).

APPENDIX

A. DATASETS

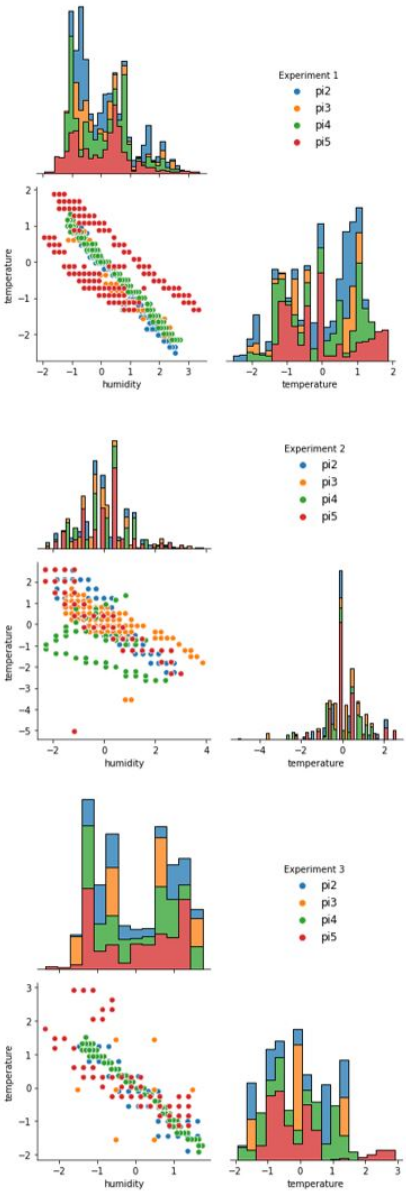


Figure A.1: The relationship between humidity and temperature per experiment alongside their distribution plots for the standardised GNFUV data.

Experiment 1					
Node	**No. of Entries**	**Humidity**		**Temperature**	
		Avg	Std	Avg	Std
pi2	1532	35.85	6.57	27.6	11.08
pi3	899	28.53	4.39	43.04	6.17
pi4	1766	35.47	6.55	28.2	11.71
pi5	2078	27.51	5.1	39.78	6.9
Experiment 2					
pi2	580	48.48	6.6	28.42	3.86
pi3	807	41.73	4.14	23.41	5.66
pi4	1021	50.56	6.4	22.76	2.76
pi5	1407	40.13	2.73	28.05	3.36
Experiment 3					
pi2	342	38.39	2.39	23.44	4.09
pi3	264	31.95	0.77	38.63	1.26
pi4	488	31.24	5.31	38.28	10.56
pi5	555	25.23	3.91	38.13	4.65

Table A.1: GNFUV Dataset Description.

Originating Cluster	Node	yes	no	Total
Newly Merged Cluster	pi1	696	3510	4206
	pi2	619	3587	4206
	pi3	658	3548	4206
Cluster 3	pi4	801	1224	2025
	pi5	765	1258	2023

Table A.2: BM Dataset Node Data Description.

B. CLASSIFIERS

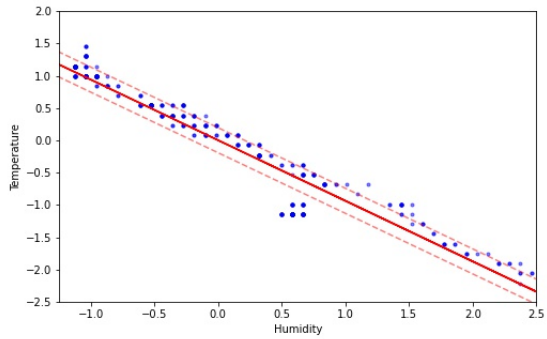


Figure B.1: Visual Representation of an SVR.

C. PERFORMANCE EVALUATION

Dataset	Classifer	Experiment	Data Configuration	Node Performance				
					Coefficient of Determination			
				kernel	**pi2**	**pi3**	**pi4**	**pi5**
GNFUV	SVR	1	original	linear	0.99	-	0.92	-
				rbf	0.99	-	0.96	-
			standardised	linear	0.99	0.94	0.91	-
				rbf	0.99	0.98	0.96	-
		2	original	linear	0.24	0.47	0.08	0.66
				rbf	0.35	0.57	0.51	0.77
			standardised	linear	0.46	0.54	0.05	0.51
				rbf	0.53	0.66	0.52	0.61
		3	original	linear	0.94	-	0.99	-
				rbf	0.94	-	0.99	-
			standardised	linear	0.96	-	0.98	-
				rbf	0.96	-	0.99	-
				Accuracy				
				pi1	**pi2**	**pi3**	**pi4**	**pi5**
BM	LR		balanced	0.87	0.86	0.86	0.72	0.72
			unbalanced	0.84	0.83	0.83	-	-

Table C.1: The average performance of node models on their native dataset.

Data Configuration	Kernel	combined precision			
		Experiment	**Threshold**	**Strict**	
				True	**False**
original	**rbf**	1	0.6	0.99	1.00
			0.8	0.99	1.00
		2	0.6	0.01	0.01
			0.8	0.00	0.00
		3	0.6	0.36	0.36
			0.8	0.34	0.34
		Weighted Average	0.6	0.40	0.40
			0.8	0.39	0.39
	linear	**Experiment**			
		1	0.6	1.00	1.00
			0.8	1.00	1.00
		2	0.6	0.90	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	0.78	0.95
			0.8	0.59	0.77
standardised	**rbf**	**Experiment**			
		1	0.6	0.58	1.00
			0.8	0.58	1.00
		2	0.6	0.19	0.25
			0.8	0.12	0.16
		3	0.6	1.00	1.00
			0.8	1.00	1.00
		Weighted Average	0.6	0.50	0.69
			0.8	0.48	0.66
	linear	**Experiment**			
		1	0.6	0.32	0.80
			0.8	0.32	0.80
		2	0.6	0.26	0.27
			0.8	0.25	0.26
		3	0.6	1.00	1.00
			0.8	1.00	1.00
		Weighted Average	0.6	0.43	0.62
			0.8	0.43	0.62

Table C.2: GNFUV Data 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.

Data Configuration	MMD precision		
	Experiment	**Threshold**	**Precision**
original	1	0.8	1.00
		0.85	1.00
		0.9	1.00
	2	0.8	0.47
		0.85	0.28
		0.9	0.12
	3	0.8	1.00
		0.85	1.00
		0.9	1.00
	Weighted Average	0.8	0.78
		0.85	0.70
		0.9	0.63
standardised	**Experiment**		
	1	0.8	1.00
		0.85	1.00
		0.9	1.00
	2	0.8	0.56
		0.85	0.51
		0.9	0.42
	3	0.8	1.00
		0.85	1.00
		0.9	1.00
	Weighted Average	0.8	0.82
		0.85	0.80
		0.9	0.76

Table C.3: GNFUV Data MMD precision Results. 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.

Data Configuration	Kernel	OCSVM precision		
		Experiment	**Strict**	
			True	**False**
original	**rbf**	1	0.99	1.00
		2	0.23	0.26
		3	1.00	1.00
		Weighted Average	0.68	0.69
	linear	**Experiment**		
		1	1.00	1.00
		2	0.96	0.96
		3	0.38	0.98
		Weighted Average	0.80	0.98
standardised	**rbf**	**Experiment**		
		1	0.58	1.00
		2	0.28	0.37
		3	1.00	1.00
		Weighted Average	0.54	0.74
	linear	**Experiment**		
		1	0.32	0.80
		2	0.28	0.30
		3	1.00	1.00
		Weighted Average	0.44	0.64

Table C.4: GNFUV Data OCSVM precision Results per kernel. 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 comments 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.

Data Configuration	Kernel	MMD precision		
		Experiment	**Threshold**	**Precision**
original	**rbf**	1	0.8	1.00
			0.85	1.00
			0.9	1.00
		2	0.8	0.00
			0.85	0.00
			0.9	0.00
		3	0.8	0.34
			0.85	0.27
			0.9	0.04
		Weighted Average	0.8	0.49
			0.85	0.37
			0.9	0.30
	linear	**Experiment**		
		1	0.8	1.00
			0.85	1.00
			0.9	1.00
		2	0.8	0.48
			0.85	0.28
			0.9	0.12
		3	0.8	1.00
			0.85	1.00
			0.9	1.00
		Weighted Average	0.8	0.78
			0.85	0.70
			0.9	0.63
standardised	**rbf**	**Experiment**		
		1	0.8	1.00
			0.85	1.00
			0.9	1.00
		2	0.8	0.15
			0.85	0.10
			0.9	0.06
		3	0.8	1.00
			0.85	1.00
			0.9	1.00
		Weighted Average	0.8	0.65
			0.85	0.63
			0.9	0.62
	linear	**Experiment**		
		1	0.8	1.00
			0.85	1.00
			0.9	1.00
		2	0.8	0.63
			0.85	0.56
			0.9	0.46
		3	0.8	1.00
			0.85	1.00
			0.9	1.00
		Weighted Average	0.8	0.85
			0.85	0.82
			0.9	0.78

Table C.5: GNFUV Data MMD precision Results per kernel. The kernel choice is not important for Experiment 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.

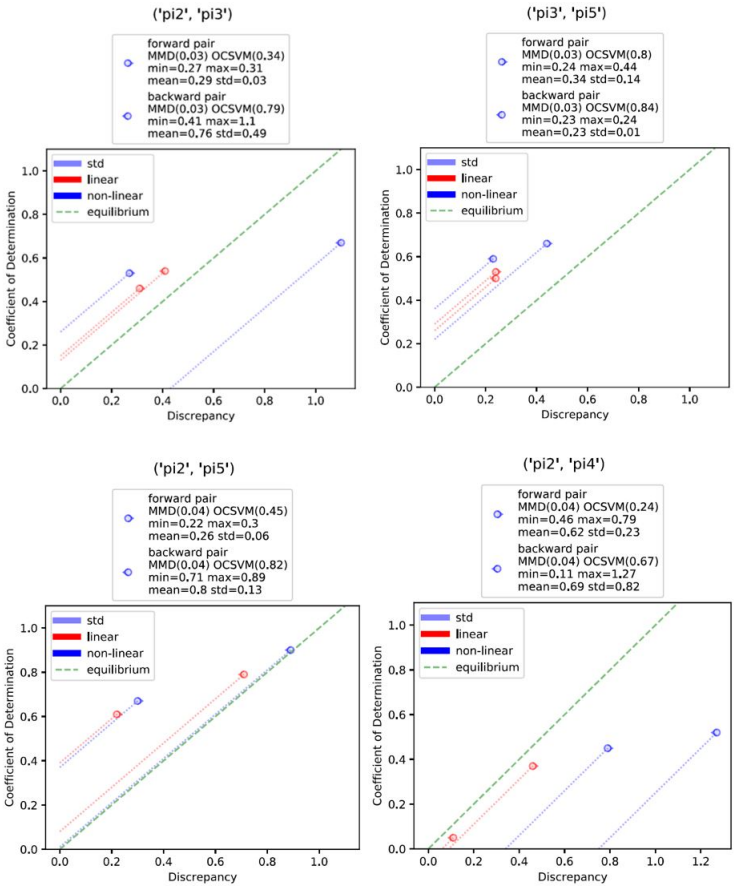
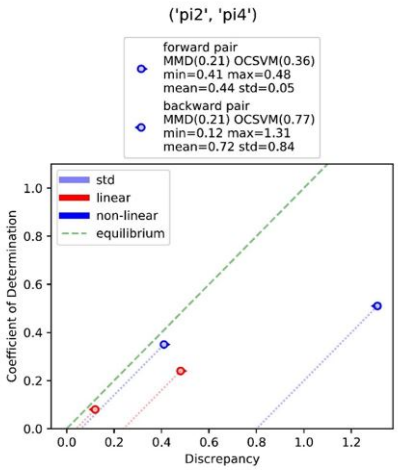


Figure C.1: GNFUV Experiment 2 Standardised Data Results. None of these pairs are good options for reusability despite the fact that most of them are above the equilibrium line. The rbf models perform better on the native dataset but nonetheless have higher discrepancy. There is a high difference between OCSVM scores of each node in the pair with the exception of the pair (pi3, pi5).

Data Configuration	MMD precision	
	Threshold	Precision
combined	0.8	1.00
	0.85	1.00
	0.9	1.00
balanced	0.8	1.00
	0.85	1.00
	0.9	1.00
unbalanced	0.8	1.00
	0.85	1.00
	0.9	1.00

Table C.6: BM Dataset MMD precision Results. The MMD precision is perfect across data configurations, this is due to the fact that the if the pairs are detected then they would be good pairs since they originate from the same cluster.



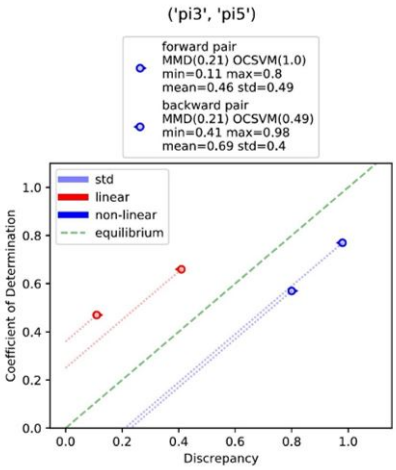


Figure C.2: GNFUV Experiment 2 Original Data Results. Neither of these pairs is a good pair for reusability. Even though for pair (pi3, pi5) we have instances models above the equilibrium line, the results are still not good. The rbf models perform better on the native dataset but nonetheless have higher discrepancy. There is a high difference between the OCSVM scores of the nodes in both pairs.

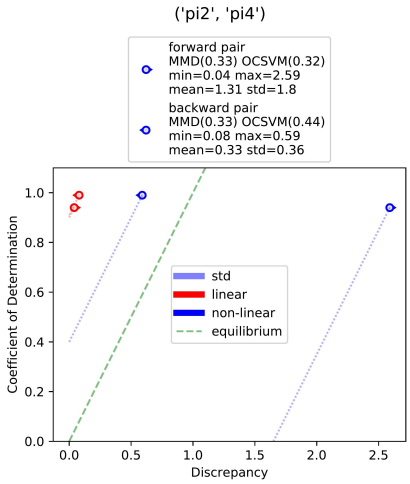


Figure C.3: GNFUV Experiment 3 Original Data Results. Both nodes have good models and could be used as replacement models for the other provided the kernel is linear, since the $R^2 - discrepancy$ is well above the equilibrium line. The linear pi4 model is the best for this pair and rbf models have higher discrepancy.

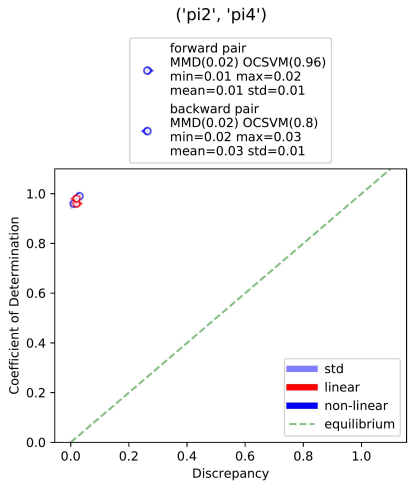


Figure C.4: NFUV Experiment 3 Standardised Data Results. Both nodes in the pair have good models and both of them could be used as a replacement model for the other, since the $R^2 - discrepancy$ is well above the equilibrium line.