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

School of Computer Science and Statistics

Siamese Algorithm Selection: A Novel Approach to Automated Algorithm Selection

Bryan Tyrrell

August 28, 2020

A Dissertation submitted in partial fulfilment
of the requirements for the degree of
MSc in Computer Science

Declaration

I, the undersigned, hereby declare that this project is entirely my own work and that it has not been submitted as an exercise for a degree at this or any other university.

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

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

Signed: _____ Date: _____

Bryan Tyrrell August 28, 2020

Permission To Lend Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Signed: _____ Date: _____

Bryan Tyrrell August 28, 2020

Abstract

The process of selecting an algorithm for use on a dataset is typically done by taking a pool of algorithms, optimizing the hyperparameters, and selecting a final algorithm based on its performance over an entire dataset. While the selected algorithm, when compared to its peers, has performed strongest over the entire dataset, it typically has not performed strongest on every instance within the dataset. This dissertation found on the Kaggle Loan Dataset, a 58% reduction in mean absolute error (MAE) was possible using an ensemble of 8 algorithms, by selecting the best performing algorithm on a per-instance basis. This gives insight into the potential performance increase available for an optimal per-instance approach.

Per-instance algorithm selection often outperforms single algorithm approaches in many domains and has proven a useful avenue to improve performance. While per-instance regression models trained on instances and algorithm characteristics have proved useful, to date, they have been found to perform inconsistently across datasets leaving room for new per instance approaches.

In this dissertation, Siamese Algorithm Selection (SAS) is proposed as a new method of per-instance algorithm selection, utilizing a Siamese Neural Network (SNN) to learn Algorithm Performance Personas (APP), which are neighbourhoods of instances that map to similar performances. Trained on instance-performance pairs, Siamese Algorithm Selection uses the representations learned by the SNN in conjunction with k-nearest neighbours clustering algorithm for per instance algorithm selection.

It was found that the proposed method works and successfully outperformed the best single algorithm, reducing MAE by 15%. The selection accuracy, the number of times the best performing algorithm was selected on a per instance basis, showed that Siamese Algorithm Selection had a 8.5% higher selection accuracy compared to the best single algorithm.

Acknowledgements

Firstly I would like to thank Edward Bergman, who is currently undertaking a PHD in Trinity college, for his insights, academic writing advice and help in implementing a Random Forest baseline. Secondly, I would like to thank my supervisor, Prof. Joeran Beel, for his countless hours of help, guidance and insight. Without him, the idea of Siamese algorithm selection would not exist today. His creation of the performance metric was integral for the success of this dissertation. Finally, Id like to thank my parents, for supporting and encouraging me throughout my college career.

Contents

List of Figures

List of Tables

Nomenclature

$\mathcal{D}$	Dataset
$\mathcal{F}$	Feature Space
$\mathcal{E}$	Embedding space
$\mathcal{P}$	Performance Space
$\mathcal{A}$	Algorithm Suite
APP	Algorithm Performance Personas
CNN	Convolutional Neural Network
MAE	Mean Squared Error
MLP	Multi Layer Perceptron
MPRE	Maximum Possible Relative Error
RF	Random Forest
RIIP	Realitive Intra Instance Performance
RSME	Root Mean Squared Error
SAS	Siamese Algorithm Selection
SGD	Stochastic Gradient Descent
SNN	Siamese Neural Network

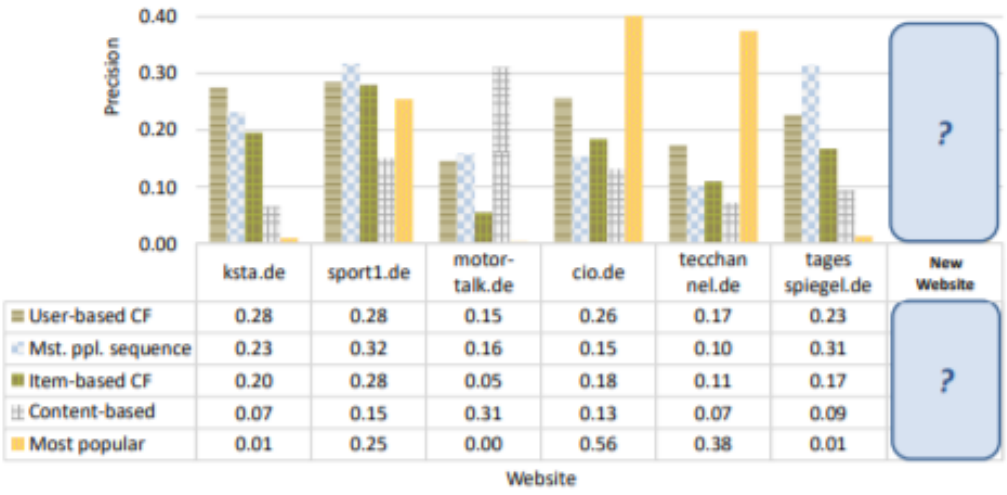
1 Introduction

1.1 Background

The algorithm selection problem was first introduced by Rice [1], as the process of selecting, from a predefined pool of algorithms, the best algorithm to run for a given task of an optimization problem. The main objectives of solving the algorithm selection problem are to reduce the time taken or increase the accuracy of the solution [2].

Algorithm selection can occur on three levels, global, mid and micro level. These 3 levels represent entire datasets, groups of instances within datasets and single instances respectively. The historical focus of algorithm selection has taken place on a global level [3] [4].

The global algorithm selection problem focuses on how multiple candidate algorithms can be applied to similar problems within the same domain, yet differing algorithms can perform "best" on each problem. "Best" can be measured using performance metrics such as RSME or measured in terms of time taken.



	ksta.de	sport1.de	motor-talk.de	cio.de	tecchannel.de	tages spiegel.de	New Website
User-based CF	0.28	0.28	0.15	0.26	0.17	0.23	
Mst. ppl. sequence	0.23	0.32	0.16	0.15	0.10	0.31	
Item-based CF	0.20	0.28	0.05	0.18	0.11	0.17	
Content-based	0.07	0.15	0.31	0.13	0.07	0.09	
Most popular	0.01	0.25	0.00	0.56	0.38	0.01	

Figure 1.1: Performance of an ensemble of 5 recommendation algorithms, measured using the precision metric, on 6 separate news websites [5].

Above in figure 1.1 an experiment involving algorithm selection on a global level is carried out by Beel et al. [5] and provides an example of the algorithm selection problem. In this work they apply an ensemble of 5 optimized recommendation algorithms to 6 individual news websites. The "Most popular" algorithm outperforms all other algorithms on the clo.de website by a margin of 0.3 measured using the precision metric. The "Most popular" algorithm is also applied to the ksta.de website, a similar task within the same domain, and achieves the worst performance with a precision measurement of 0.01 which was 0.27 below the best performing algorithm "user-based collaborative filtering". From this experiment, no algorithm performed best across all websites within a similar task domain. The idea "one size cannot fit all" in terms of algorithm selection can be applied as each algorithm displays strengths and weaknesses when measured on differing websites.

The mid and per instance level algorithm selection problem focuses on individual datasets. Within a datset, factors such as feature correlation, algorithm characteristics and hyperparameter tuning are a subset of the large pool of factors that can influence individual algorithm performance both positively and negatively.

A mid level algorithm selection problem classifies this problem as a unknown grouping of instances for which a select algorithm, due to one or multiple factors, will outperform its peers. A per instance view of this problem applies on a more granular level in which an unknown algorithm will perform best on individual instances of a dataset.

Figure 1.2: Performance of an ensemble of 8 algorithms on the MovieLens 100k and 1m dataset, showing the percentage share for which each algorithm performed strongest.

Collins et al. [6] carries out a per instance approach to algorithm selection. Figure 1.2 shows the selection share of each algorithm for an ensemble of 8 algorithms on the MovieLens 100k and 1M datasets. The selection share is defined as a percentage measure from a possible 100% of how many instances a given algorithm was the optimal performer compared to its peers within an ensemble. From figure 1.2 above, the best performing algorithm on the 1M dataset, "SGD++" achieved a selection share of 19.65% meaning that other algorithms within the ensemble where stronger performing and a more suitable choice for 80.45% of instances in the MovieLens dataset.

When compared to the global algorithm selection problem, the mid and per instance algorithm selection problem highlight that by applying a single algorithm to an entire dataset, a large proportion of instances, in this case 80.45%, would be more accurately predicted by another algorithm within the ensemble. Using an ensemble of algorithms on a per instance basis would lead to an increase in the accuracy of the overall solution, in the case of Collins et al. [6] experiment a reduction in root mean squared error (RSME) of 25.5%. This problem is also replicated in other work [7] and [8] showing the per instance algorithm selection problem extends across datasets and domains.

A solution to the algorithm selection problem on all levels is automated algorithm selection. The automated selection of algorithms for a given problem can be defined as: Given a dataset $\mathcal{D}$ of instances for a problem $\mathcal{P}$, a set of algorithms $A = \{A_1, ...A_n\}$ that are trained on $\mathcal{D}$ to solve problem $\mathcal{P}$ and a metric $\mathcal{M}$ that measures performance of each algorithm, construct a selector $\mathcal{S}$ that maps any number of instances $d \in \mathcal{D}$ to an algorithm $\mathcal{S}_i \in \mathcal{A}$ such that the overall performance of $\mathcal{S}$ on $\mathcal{D}$ is optimal according to metric $\mathcal{M}$.

The goal of automated algorithm selection is to create a selector $\mathcal{S}$, that can be trained to automatically select the optimal algorithm for either a single instance, grouping of instances or an entire dataset to maximise performance according to some performance metric $\mathcal{M}$.

Automated algorithm selection is used in the context of either optimization techniques or meta-learning, where meta learning is described as the process of "learning to learn" [9]. This dissertation focuses on the area of Meta-learning.

The Meta-learning community has viewed automated algorithm selection as a classification task. That is, for a single, group or dataset of instances, train a machine learning model to predict and classify the best performing algorithm from an ensemble of algorithms. The model itself is trained on the features and sometimes meta features of instances coupled with their performance to learn how each algorithm from an ensemble performs on a given instance. This learning is applied to predict and classify the performance rank of each algorithm within an ensemble for an unseen task.

In the context of meta learning, automated algorithm selection can also be implemented as a regression task. A machine learning model is trained to predict the expected label and hence performance of a algorithm or group of algorithms on an instance. This differs to a classification task in the sense that a meta learner is aiming to predict the expected label of an algorithm on an instance rather than classify its ranked performance within an algorithm ensemble.

This dissertation focuses on automated per instance algorithm selection in the field of meta learning. Treating the problem as a regression task, the goal is to create a meta learner that can successfully predict the performance of an ensemble of algorithms on a per instance basis. The meta learner will learn from the error between the predicted value of each algorithm and the given label coupled with the features of each instance to allow for a prediction of future algorithm performance for an unseen instance.

1.2 Research Problem

By selecting a globally best performing algorithm strong performance is guaranteed, but theoretically perfect performance can predominantly only be achieved by selecting the best performing algorithm on a per instance basis as opposed to a global level.

The per instance selection problem has been discussed above and shown in figure 1.2 where the globally best performing algorithm "SGD++" achieved a selection share of 19.65%. In Collins et al. [6] research it was found that a saving of 25.5% RSME could be achieved with a selection share of 100%.

To achieve this increase in performance, the meta learning community have focused on meta feature creation. These meta features coupled with an instances original features are used to directly learn and predict algorithm performance. This may not be optimal as explained below.

	Features				Target	Predicted Value		Error		Rank	
ID	X1	X	....	XN	Y	A1	A2	A1	A2	A1	A2
1	A	1	....	24	8	2	7	6	1	2	1
2	B	11	....	42	8	2.5	6.5	5.5	1.5	2	1
3	A	1	....	52	4	3	1	1	3	1	2
4	B	5	....	38	3	1	3	2	0	2	1

Figure 1.3: A theoretical example of a regression task with two algorithms A1 and A2. The goal of the task is to predict a rating ranging from 1 to 10.

From figure 1.3 above, two instances are shown, denoted by id 1 and 2. A model trained to predict the performance of these two instances based on their features would struggle as the features of id 1 and 2 are quite dissimilar yet their performances are similar, that is, the instances have the same algorithm perform best, A2, and have similar predicted target values. A meta learner would struggle to learn a direct correlation between these dissimilar features and similar performances.

The problem that needs to be addressed, is how to create a model that accurately predicts performance of these instances and new unseen instances taking into account the potential for seemingly dissimilar features and little intuitive correlation.

1.3 Research Question

This dissertation will answer the question: "Can Siamese Neural Networks be adopted into per instance algorithm selection to accurately learn and predict instance performance similarity".

1.4 Research Goals

The overall goal of this dissertation is to create a novel automated per instance algorithm selection approach, training a meta learner to accurately predict current and future performance of a suite of algorithms on a per instance basis.

To achieve this, the potential of the performance space will be leveraged as a tool for prediction of instance performance. Historically the performance space has been seen as a product of algorithm selection rather than a resource in performance prediction. This dissertation aims to reverse this and leverage the performance space to learn if two instances are similar in terms of performance. With this knowledge a meta learner can learn a transformation function, that can accurately transform an instances features into an embedding which represent its performance within the performance space.

The meta learner described above will learn a transformation based on performance similarity between instances. While feature similarity is usually straightforward to calculate due to large numbers of features representing distinct attributes such as age or gender, performance similarity is not as clear. Performance of algorithms typically take up small segments of the available performance space, with subtle changes in instance performance indistinguishable in large datasets. With this in mind, a goal of this dissertations is augment the performance space, making use of the entire space to model the performance relationship an instance has with its features and also its instance peers within a dataset.

1.5 Contributions

To summarize, the contributions of this dissertation are:

1. A novel per-instance selection method, utilizing a Siamese neural network architecture to increase performance over a single best algorithm.
2. The concept of an 'Algorithm Performance Persona' (APP), used to identify groupings of instances with similar performances characteristics within the performance space.
3. An implementation of a novel performance metric, used to normalize algorithm performances accounting for cases of only small differences in their relative performances.

This novel per-instance selection approach dubbed "Siamese Algorithm Selection" was tested on the Kaggle loan dataset and reduced the mean absolute error (MAE) by 15.8% compared to the globally best performing algorithm Multilayer perceptron (MLP) regressor. This approach shows promising results and through further testing outlined in future work has the potential to provide an automated solution to the per instance algorithm selection problem faced by many in the meta learning community.

2 Background

2.1 Siamese Neural Networks

The use of Siamese Neural Networks (SNN) was originally implemented to identify signature similarity [10] but has since found its way to object tracking [11], sentence similarity [12] and speaker recognition [13].

SNNs have been highly successful in the field of facial recognition [14] and more specifically in the task of few shot image classification [15]. This task involves the classification of images where very few training samples exist. Traditional models have struggled on this task as they are typically trained on large labelled datasets. SNNs excel in this area as they learn the similarity between two instances rather than the features of an instance itself which for images of faces are in the thousands, this allows SNNs to learn from very few training examples.

Figure 2.1: An example of a few shot classification task [16].

A typical deep learning model, e.g a convolutional neural network (CNN) would have a high probability of misclassification of the two bird classes, 1 and 2 in figure 2.1 above as the feature set of both classes are quite similar yet the number of training images is minimal. A CNN requires a large pool of training examples to learn subtle differences in similar classes. A SNN takes a different approach where it calculates the similarity between two instances. If an image similarity compared to an anchor image is under a specified margin α the image is classified as belonging to the same class. In the task above, an SNN would compare each test image to the given class image 1 an 2 to calculate a similarity score for both classes. Using image similarity comparison, it gives the SNN a higher probability of correctly classifying the unlabelled image for smaller number of training examples. SNNs can learn and compute similarity between images with a small number of training examples while CNNs require large amounts of training data to learn all the features of a class before an accurate comparison between similar classes can occur.

This point is shown by Wei et al. [17] where they train a SNN and CNN on the CUB Birds dataset shown in figure 2.1 above, containing 11,788 images. Their goal was to show how SNNs outperform CNNs on few shot image classification tasks. Their SNN approach had a selection accuracy 62.48% ±1.21% while the CNN approach using a K nearest neighbour (KNN) algorithm achieved a selection accuracy of 41.93% ±1.69%, with both approaches being tested on a 5 shot image classification task. Their SNN approach produced a 20.55% stronger selection accuracy on this few shot image classification task showing SNNs strength on few shot image classification tasks.

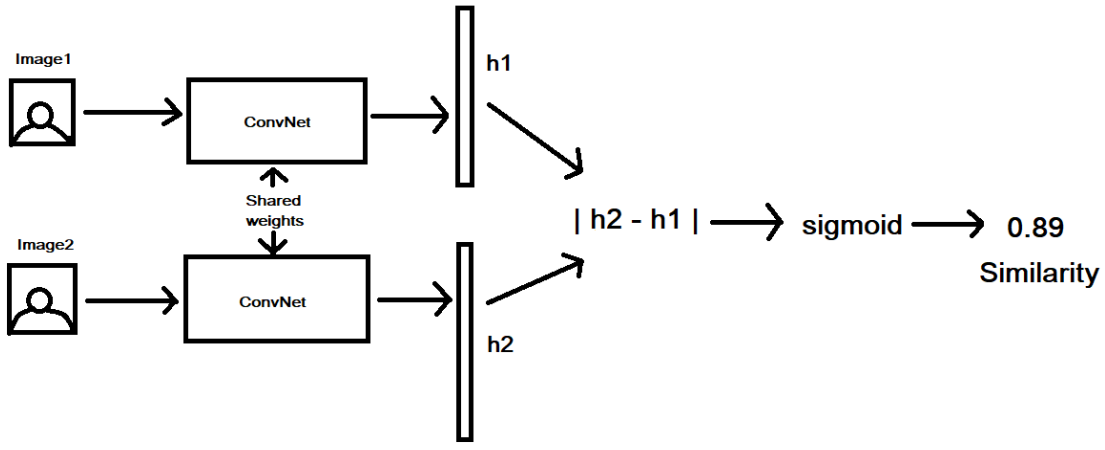


Figure 2.2: SNN architecture containing 2 identical neural networks computing the similarity between two images credit:[18].

The architecture of a SNN consists of 2 identical neural networks with shared weights. By tying the weights together, it guarantees symmetry of the network. This is an important aspect as the architecture should be invariant to the switching of input data between each side of the network.

$$(W, Y, \overrightarrow{X}_1, \overrightarrow{X}_2) = (1 - Y)\frac{1}{2}(D_w)^2 + (Y)\frac{1}{2}\{\max(0, m - D_w)\}^2 \quad (1)$$

Contrastive loss [19], a distance based loss function shown by equation 1 above is a popular loss function implemented in SNN architectures. Contrastive loss ensures positive instance pairs are embedded closer in the embedding space and vice versa with negative pairs. Contrastive loss works by not altering the distance between positive pairs while insuring distance between negative pairs must be greater than a margin m. Y is used to label pairs where 0 represents a positive pair and 1 represents a negative pair in equation 1 above. An interesting aspect of contrastive loss is once a negative pairs distance is above a margin m no extra effort is used on embedding the pair.

The final embedding space produced by the SNN represents the similarity between instances where smaller distances signify greater similarity. For classification two main approaches are common for SNNs, the first approach involves calculating a similarity to each distinct class within the embedding space by measuring the euclidean distance from the test embedding vector to an instance embedding from each distinct class. The second approach involves using a K nearest neighbour algorithm to select the K nearest embeddings with known class label to the test embedding to classify the test instance. From figure 2.3 below, each approach will be discussed.

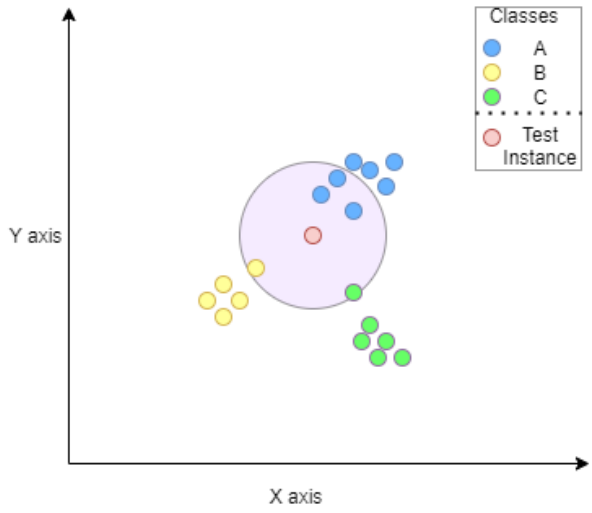


Figure 2.3: A theoretical embedding space showing a test instance in red with 3 separate classes.

For all classification methods the test instance features are transformed into an embedding and placed within the embedding space. Above in figure 2.3, a theoretical example of a SNN embedding space can be seen. The red circle indicates the location of a test instance embedding, with all other colours representing a distinct class.

The first classification approach uses euclidean distance to compute the similarity of each distinct class to the test embedding. A random training embedding from each class is selected and paired with the given test instance.

For each pair, the result of the distance calculation is passed to a sigmoid function. The role of the sigmoid function is to normalize the output to a scale between 0 and 1 which represents the pairs similarity. This process is carried out for all classes, selecting the class with highest similarity to the test instance.

The second classification approach utilizes a KNN algorithm. This approach calculates the K closest neighbours and selects the class with the highest number of neighbours. From figure 2.3 above, the purple circle highlights the 5 closest neighbours to the test instance. The test embedding is classified as class A, with the blue circles present 3 of 5 times representing a majority for class A.

3 Related Work

Automated algorithm selection has shown to be effective in many disciplines including databases [20, 21], SAT [22], machine learning [23, 24, 25, 26], data mining [27], information retrieval [28, 29, 30, 31], recommender systems [32, 33, 34], reference-string parsing [35], cloud resource allocation [36], electronic design [37], and material sciences [38]. This section will discuss the work that has lead to this dissertation, exploring the positive and negative approaches deployed in previous work.

3.1 Algorithm Performance Space

The performance space, $\mathcal{P}$, represents the the past performance of algorithms and is often seen as a product rather than a tool of algorithm selection. When used, prior performance of algorithms has been a useful characteristic as shown by Lobjois and Lemaitre [39] and Bensusan and Giraud-Carrier [40] who both model prior evaluations of algorithms to infer which algorithm is best for unseen instances.

Lobjois and Lemaitre [39] proposes a method called Selection by Performance Prediction (SPP) which allows for the selection of a Branch and Bound algorithm from among several promising algorithms when faced with a particular problem instance.

Their goal is to select the best candidate algorithm to solve the problem of the given instance in the shortest time possible. For clarity, in this paper each instance is a problem that needs to be solved by an optimization algorithm. They predict the best performing algorithm by estimating the performance of each algorithm on the problem instance. Estimation involves running the algorithm suite on a single node to gauge runtime performance before applying this estimate to the total number of nodes to calculate runtime. Using past performance for a single node of the problem instance, a reasonable estimate of future performance is made on the entire problem instance.

Lobjois and Lemaitre show that their solution outperforms all baselines set, achieving a saving in computation time of 56.49% on cumulative running times for 238 random instances compared to the next best baseline, BB2 [41].

Bensusan and Giraud-Carrier [40] uses the performance space in a similar way to this dissertations proposed use in which they estimate that certain datasets will map to certain locations in the performance space, they refer to these locations as tasks. Their goal is to allow for the classification of new datasets into these tasks to allow for an accurate prediction of multiple algorithms run times based on the past performance of other instances within the task.

Their results find that their approach performs better over the standard approach of selecting the default class, although fails to outperform their selected baseline. While the results are not optimal they show how a performance space can be of use in predicting future algorithm performance.

Xu et al. [42] put the performance space to great use in the 2008 propositional boolean satisfiability (SAT) competition, training on historical algorithm performances from previous competitions to create a winning per-instance selection model.

Focused in the area of SAT, this approach builds an empirical hardness model to select the best performing algorithm from a suite of potential options on a per instance basis. Their model outperformed their competitors in the categories of selection accuracy and average runtime achieving a selection accuracy of 5% higher than the next best solution coupled with an average runtime reduction of 13.98%.

The use of algorithm performances has shown to be a major factor in algorithm selection, leading us to leverage algorithm performance in our approach to create a per instance algorithm selection approach using the performance space that is flexible enough to be applied to many areas of machine learning.

3.2 Algorithm Characteristics

One approach to per-instance selection SAS draws from is to incorporate algorithm characteristics. The work of Pulatov and Kotthof [43] investigates this in the domain of SAT solvers, training a regression model on algorithm source code (referred to as algorithm features) and standard instance features to predict a solver for each instance.

Their solution was able to automatically extract algorithm features for use in the pipelines. Some of the main algorithm features they used in this pipeline where cyclomatic complexity (average and total), max indent complexity (average and total), lines of code (average and total), size in bytes (average and total) and number of files. They also made a point of ignoring source code that was not critical to algorithm feature selection such as certificate generation which is mainly used for security.

They chose the random forest regression algorithm to train on the selected algorithm and instance features and trained on several algorithm selection library datasets such as SAT11-INDU and OPENMLWEKA-2017 and also libraries not related to algorithm selection such as SAT-2018 [44].

This model focuses on improving performance in the context of time and achieved a 95% reduction in time spent solving when compared to the single best algorithm, however, later experiments [45] found mixed and inconsistent results for other datasets.

3.3 Performance Clusters

Work by Ekstrand and Riedl [46] proposed a mid level meta learner that aimed to apply the best algorithm to a subset of data within the dataset where "best" is measured using the RSME metric. They selected a grouping of 5 diverse algorithms and using a classifier, attempted to choose the best algorithm between item-item and user-user collaborative filtering for each user. Their solution (RMSE: 0.78) failed to outperform the globally best algorithm (RSME: 0.74).

3.4 Algorithm Selection Architectures

3.4.1 Siamese Neural Networks

In the work of Kim et al. [47] they train an SNN to learn which acquisition function to choose in warm-starting algorithm configurations. By learning the similarity of meta features between multiple datasets, configurations that where successful on previous similar datasets in terms of meta feature similarity can be applied to current datasets reducing optimization time.

By learning a similarity between datasets from prior algorithm performances they successfully identify good configurations, this dissertation extends this idea of learning algorithm performance from a dataset level to a per-instance level of algorithm selection.

3.4.2 Clustering Trees

Another approach to per instance algorithm selection has been in the field of clustering trees. Clustering trees are tree like structures in which nodes or "leafs" are used to naturally segment the data into clusters to allow for the partitioning of data based on a given rule or attribute.

Todorovski et al. [48] uses clustering trees to predict algorithm performance on a dataset level. The clustering tree are trained to predict the ranking of an algorithm from an esnemble on a given dataset rather than learning an error metric. This is done to stop the variance that can occur in using metrics such as absolute error.

They show concrete evidence to correlate lower error values when the trees where trained on rank as apposed to error metrics. This is an interesting finding and will be factored into this dissertation. Their approach does not outperform their baseline, regression trees.

3.4.3 Error Based Algorithm Selection

Another per instance approach was proposed by [49], this work was carried out on recommender systems, where a meta learner was trained on the prediction error of movie ratings with an ensemble of 9 algorithms on the MovieLens 100k and 1M datasets [50]. This approach to algorithm selection is more consistent with our proposed Siamese algorithm selection approach.

Their method involved training a linear model on the errors in predictions coupled with the features of the training data, they also include 10 meta features in this training data, an example of some of the meta features included are the rating mean and standard deviation.

The goal of the trained meta learner was to accurately predict the future error of each algorithm on a per instance basis to select the best performing algorithm. Their approach (RMSE, 100K: 0.973; 1M: 0.908) did not outperform the globally single best algorithm, SVD++ (RMSE, 100K: 0.942; 1M: 0.887) performing worse in terms of RSME by 3.24% and 2.34% on the 100k and 1M MovieLens datasets respectively. They state a cause for this could be the use of similar class of algorithms in their ensemble, this dissertation applies this theory and selects a diverse pool of algorithms to protect against this potential issue.

3.4.4 Ensemble learning

Ensemble learning, which is the process of combining several algorithms to make a prediction, has been used in the field of algorithm selection with strong success. Ensemble learning encompasses multiple sub fields including voting, stacking, bagging and boosting which are independent ensemble learning approaches that can be applied to problem tasks.

Gaikwad and Thool [51] uses an bagging ensemble approach to outperform single algorithms in terms of selection accuracy, on the NSLKDD [52] and DARPA datasets.

Their ensemble approach showed an increase of a 4.13% in selection accuracy compared to the best single algorithm on the test dataset. An interesting point of note is the ensemble approach employed measured a model build time of 44 seconds which was 7 seconds slower compared to the best single algorithm. Ensemble approaches must execute every algorithm before a prediction can be made meaning the approach can be resource intensive at build time which must be weighed against any potential performance improvement.

4 Siamese Algorithm Selection

Siamese Algorithm Selection consists of two main parts, a Siamese Neural Network (SNN) that embeds instances to a space $\mathcal{S}$ and a nearest-neighbours algorithm to assign to each unseen embedded instance $s_i \in \mathcal{S}$ an algorithm from a pool of algorithms $\mathbf{A} = \{\mathcal{A}^{(1)}, \ldots, \mathcal{A}^{(m)}\}$. The SNN is trained from Algorithm Performance Personas (APP), clusters of points for which algorithms performed similarly to create said clusters in the embedding space $\mathcal{S}$. For an unseen instance d_{new}, to select an algorithm to run on said instance, it is embedded with the SNN to get its embedding s_{new} and select the algorithm $\mathcal{A}^{(j)}$ which most often performed best for the embeddings k-nearest neighbours. This section will discuss the main topics of Siamese algorithm selection in depth, to give insight to the approach.

4.1 Performance Space

A key component of Siamese algorithm selection is how algorithm performance is measured and plotted within the performance space. Requirements of a selected performance metric for this approach are:

- To create a performance space that exposes intricate performance characteristics through utilization of the entire space.
- To plot instance performance taking into account performance similarity between instances within a dataset.
- To highlight performance differences between ensemble algorithms on a given instance.

These three points will insure a performance space is constructed that can be subsequently classed into differing APPs. Conventional performance metrics will be discussed in this section but may not be suitable for use in creating this performance space. This is due to performance metrics being used to measure exact performance of each instance without taking into account relative performance of other instances or algorithm performance comparison for instances within a dataset.

4.1.1 Performance Rank

Performance Rank is an intuitive option to separate algorithm performances within a performance space. By using a performance metric such as RSME and applying performance rank, a distinct grouping for each APP will be present. For example, a single APP can be defined by a set of algorithm performances (A1,A2,A3: 1st, 3rd,2nd) where 1st 3rd and 2nd represent algorithms performance rank on individual instances. This solution may be viable for larger ensembles of algorithms where the possible factorial combination of algorithms is quite large. This will allow multiple characteristics of algorithm performance to be captured within separate APPs. For smaller suites of algorithms, the number of possible combinations becomes increasingly small. For a suite of 3 algorithms the possible combinations of algorithm ranks totals to 6. Algorithm performance over large datasets is quite varied and can fluctuate due to many factors, with 6 possible APPs, subtle changes in algorithm performances will be difficult to represent within such a small pool of APP clusters. This limits the effectiveness of performance rank to solely measure algorithm performance.

Another issue with using rank to model algorithm performance is it constricts algorithm performance into discrete values. By doing this, subtle patterns in performance data which may have been exposed with a continuous measurement of performance may be lost. A theoretical example of this is shown in figure 4.1 below.

ID	Features				Target	Predicted Value		Error		Rank	
	X1	X2		XN	Y	a1	a2	a1	a2	a1	a2
1	0	220	...	5	8	7	4	1	4	1	2
2	0	215	...	2	8	7.9	7.6	0.1	0.4	1	2
3	1	217	...	3	8	7	6	1	6	1	2

Figure 4.1: A theoretical example demonstrating how rank is not solely a suitable performance measure.

In figure 4.1 above, the performance rank for three instances are shown. All instances would be classed as identical in terms of similarity using performance rank which is not the case. The performance of the algorithms on instance 1 is a factor of ten weaker when compared to instance 2 but this is not reflected in the performance rank. When instance 1 is compared to 3 there is also dissimilarity between the instances. While the rank is identical, algorithm a2 have performed 25% better on instance 1 as apposed to instance 3 yet both these instances are classed as identical in terms of performance rank.

By classing these instances as identical, all instances would be placed in the same APP when underlying factors have caused vast performance differences in algorithm performance.

4.1.2 Mean absolute Error

Another option to measure algorithm performance is mean absolute error (MAE). MAE is a measure of errors between paired observations, in the case of machine learning, the observations are the label and the predicted value of each algorithm respectively. Below MAE is displayed in equation 1, where y and $\hat{y}$ are label and predicted value and m is a count of the total number of instances for a dataset.

$$MAE(y, \hat{y}) = \frac{1}{m} \sum_{i=1}^{n} |y_i - \hat{y}_i| \tag{1}$$

In terms of a measurement of algorithm performance for SAS, MAE is not a suitable choice. MAEs main application is measuring the performance of an algorithm over an entire dataset. If two algorithms perform similarly in prediction of test data the resultant performance space will be densely populated in a small sector. MAE also does not consider the performance similarity between instances which is a key component in classifying APPs.

4.2 Novel Performance Metric

The overall goal of the performance metrics discussed above are to give an accurate measurement of performance of a single algorithm on a given task. While this is also a goal of Siamese algorithm selection, the exposure of intricate performance patterns and instance performance similarity are also goals of a suitable performance metric for the approach. With the metrics above partially fulfilling these goals, this dissertation proposes the implementation of a novel performance metric to satisfy these requirements for each instance.

This performance metric combines relative intra-instance performance (RIIP) and max-possible relative error (MPRE) to expose both underlying performance patterns and instance performance similarity which can be later classified into APPs. It should be noted that RIIP and MPRE mainly target error based performances such as MAE or RSME but these ideas can be extended to positive performances such as accuracy.

RIIP and MPRE are to account for relative algorithm performances and the possible scale of performances respectively, separating performances that would otherwise indicate several good candidate algorithms. In the sections below both these components which make up this metric will be discussed in depth.

4.2.1 Max-Possible Relative Error

MPRE takes into account the maximum possible error for an instance, dependant on both its label y_i and the range it is bounded in, $[B_{min}, B_{max}]$. MPRE of a performance is calculated as $p_i^{(j)}/\epsilon_i$ where the maximum possible error for that instance is given by $\epsilon_i = \max(B_{max} - y_i, y_i - B_{min})$. MPRE is used to factor that for individual instances the scale of performance may vary, this will allow for an accurate representation of relative instance performance and hence performance similarity between instances.

For the kaggle loan dataset the prediction label, interest rate, ranges from 0 to 30.99% for the entire dataset. If a label for an instance is 15.00% then the maximum possible error for a prediction is 15.99%, which occurs if an algorithm predicted 30.99%. However if the target interest rate is 28%, the maximum possible error is also 28% and a greater margin for error in performance. An example below demonstrates this concept.

ID	Features				Target	Predicted Value		Error		Rank	
	X1	X2		XN	Y	a1	a2	a1	a2	a1	a2
1	0	220	...	5	8	7	4	1	4	1	2
2	0	215	...	2	8	0.7	0.4	7.3	0.4	1	2
3	1	217	...	3	8	7	6	1	6	1	2
4	0	222	...	1	8	7.2	7.8	0.8	0.2	2	1
n	1	262	...	4	4	3.2	3.8	0.8	0.2	2	1

Figure 4.2: A theoretical example predicting a rating between 1 and 10.

In figure 4.2 above, the absolute errors for instance **4** and instance **n** are the same, measuring 0.8 and 0.2 respectively. The target for id 4 is 8 while the target for id n is half that at 4. The MPRE metric argues that the algorithms performance on id n is twice as good as id 4. For id n, the algorithms are only 10% and 2.5% off the actual target, while for id 4 the algorithms are 20% and 5%, where lower percentage shows stronger performance.

MPRE considers an algorithms performance as its error compared to its maximum possible error on each instance. By doing this, MPRE can calculate and provide a relative scale of performance which can be used to compare relative performance of algorithms on similar instances.

4.2.2 Relative Intra-Instance Performance

RIIP is calculated as $\min_k p_i^{(k)}/p_i^{(j)}$, scaling the errors to the range $[0, 1]$ with the minimum error at 1 and greater errors approaching 0 as error increases. This metric allows for comparison of algorithm ensemble performance on a given instance, comparing individual algorithm performance to highlight slight performance differences between several similar performing algorithms on a given instance.

ID	Features				Label	Predicted value		Error		RIIP	
	X1	X2	...	X74	Y	SGD	Catboost	SGD	Catboost	SGD	Catboost
4	0.230769	1		0	19.99	19.48957	20.31139	0.500434	0.321393	0.642228	1
7	0.230769	0		0	11.49	11.5027	11.70016	0.012697	0.210164	1	0.060417
14	0.746795	0		0	10.41	10.66892	10.45823	0.25892	0.048235	0.186292	1

Figure 4.3: Calculation of RIIP metric on the Kaggle Loan Dataset.

From figure 4.3 above the RIIP calculation for 2 algorithms, SGD and CatBoost, can be seen. Taking instance id 4 as an example, Catboost is classed as best performing algorithm with an absolute error of 0.32. The calculated RIIP value for Catboost is therefore 1 and SGD is scaled relative to this performance. If SGD was also a very strong performing algorithm this scaling would be less severe, and in this case SGD performs relatively well and as such it is scaled to represent this performance, achieving a MPRE measurement of 0.64. This allows for a greater distinction in terms of performance of algorithms for this instance.

This scaling is even more apparent for instance 7. SGD performs extremely well on this instance achieving an error of 0.012, while Catboost also performs quite well achieving an error of 0.21, the third lowest recorded error for these 3 instances. Relative to SGDs performance, Catboost performs quite poorly for this particular instance.

RIIP accounts for this, scaling Catboost to a RIIP value of 0.06 relative to SGD to signify the very strong performance of the SGD algorithm for this instance. Through RIIPs scaling, it will allow for a clear indication of SGDs strength on this particular instance. By using RIIP as a measurement, the strong performance of an algorithm can be captured and make it a clear choice for use on instances with similar characteristics.

4.2.3 Final Metric

The final metric is represented by joining MPRE and RIIP as $\tilde{p}_i = (\tilde{p}_i^{(1)}, \ldots, \tilde{p}_i^{(m)})$ we have that each performance is normalized according to

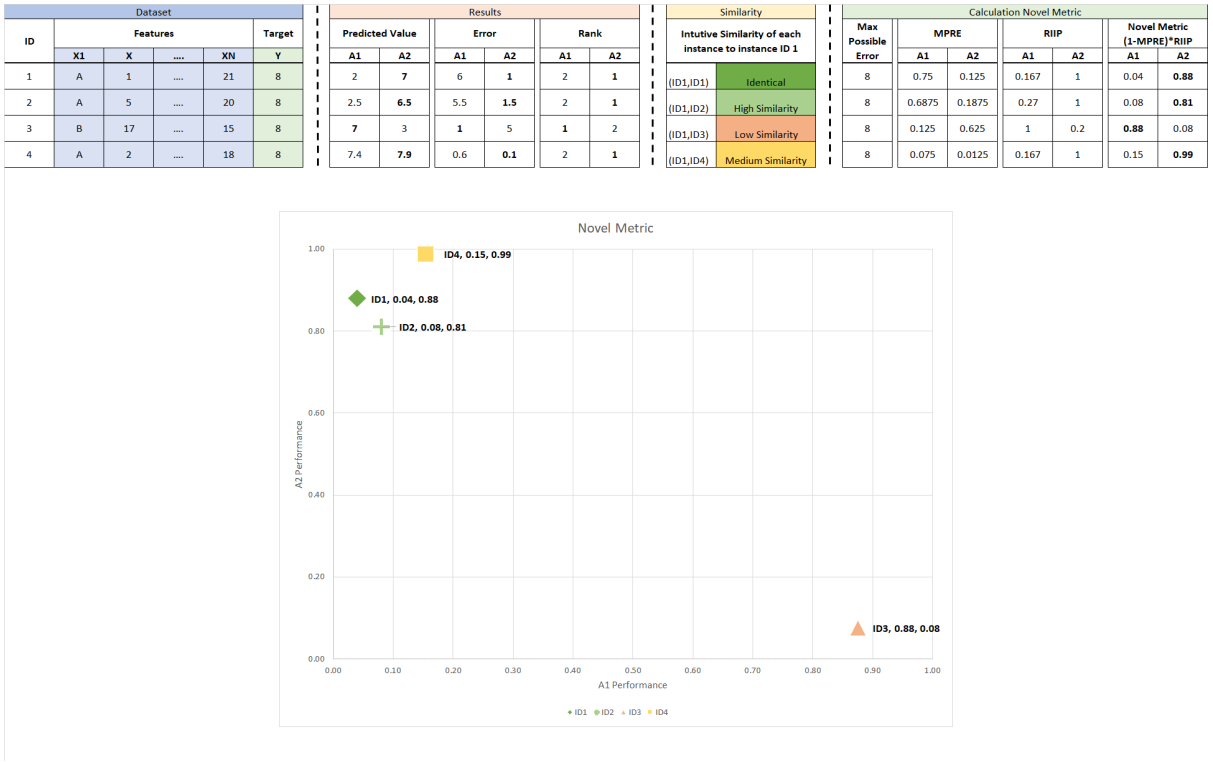
$$\tilde{p}_i^{(j)} = \left(1 - \frac{p_i^{(j)}}{\epsilon_i}\right) \cdot \frac{\min_k p_i^{(k)}}{p_i^{(j)}} \tag{2}$$

$p_i^{(j)}$ Instance performance

ϵ_i Maximum Possible Error

$\min_k p_i^{(k)}$ Best Performing algorithm from the Algorithm ensemble

Equation 2 above, presents the novel metric used to augment the performance space in terms of instance similarity. Combining MPRE and RIIP will allow for similarity of instances in the performance space to be measured. Below a theoretical demonstration of the metric in practice will be illustrated.



Dataset

ID	Features				Target
	X1	X	....	XN	Y
1	A	1	....	21	8
2	A	5	....	20	8
3	B	17	....	15	8
4	A	2	....	18	8

Results

Predicted Value		Error		Rank	
A1	A2	A1	A2	A1	A2
2	**7**	6	**1**	2	**1**
2.5	**6.5**	5.5	**1.5**	2	**1**
7	3	**1**	5	**1**	2
7.4	**7.9**	0.6	**0.1**	2	**1**

Similarity

Intutive Similarity of each instance to instance ID 1	
(ID1,ID1)	Identical
(ID1,ID2)	High Similarity
(ID1,ID3)	Low Similarity
(ID1,ID4)	Medium Similarity

Calculation Novel Metric

Max Possible Error	MPRE		RIIP		Novel Metric (1-MPRE)*RIIP	
	A1	A2	A1	A2	A1	A2
8	0.75	0.125	0.167	1	0.04	**0.88**
8	0.6875	0.1875	0.27	1	0.08	**0.81**
8	0.125	0.625	1	0.2	**0.88**	0.08
8	0.075	0.0125	0.167	1	0.15	**0.99**

Figure 4.4: Calculation of novel Performance metric on a theoretical dataset.

Figure 4.4 above, gives an example of how the metric performs on 4 theoretical instances. Taking instance with id 1 as an anchor point for similarity, each instance has a varying level of similarity highlighted by individual colours. When plotted, the metric has satisfied this intuition and plotted instances relative to their given similarity.

4.3 Algorithm Performance Personas

Siamese neural networks have mainly been used in the calculation of similarity between faces [53]. For training data, two input faces are fed into the network with a label of 1, indicating the same person or 0, indicating faces of two different people. The data can be easily labelled as it is inherently given if two faces are the same person or different people. For algorithm performance this is not the case as there is no "definite" label that can be applied to pairs of instance performances as being the same or not. This dissertation proposes using Algorithm Performance Personas (APP) to identify instances as the same or not.

The term 'Persona' originates from user experience design and market research, which define 'Personas' as "archetypal users whose goals and characteristics represent the needs of a larger group of users" [54]. As this term explains the needs quite concisely, an APP is defined in a similar manner,

"A subset of algorithm performances which represent the characteristics of a larger group of instances."

APPs provide a mechanism to allow for labelling of training data. By characterising a subset of instance performances as an APP, they can be classified as the "same" for Siamese neural network training. This allows for the SNN to learn an embedding which can translate future instances with unknown performance into these APPs, where APPs performance characteristics can be applied to the instance to predict its performance. These APPs are used to categorize a group of instances within a certain distance in the performance space as "same".

4.4 Learned APPs through Example

The SNN is trained to embed a pair of instances d_i, d_j to some vectors r_i, r_j whose distance encode their APP similarity to one another. As training relies on pair comparisons, if all n^2 possible pairs were considered, training time would become cost prohibitive. Instead, 4 kinds of pairs are selected for training, *easy-positive*, *hard-positive*, *easy-negative* and *hard-negative*. Using the distance between performance vectors $\|\tilde{p}_i - \tilde{p}_j\|$, the pairs are classified as *positive* if they are close and *negative* if they are far within the performance space created by the proposed performance metric. Depending on the distance $\|d_i - d_j\|$ in feature space, the pairs are similarly classified as being *easy* if they are already located close and *hard* in the case that they are distant. These notions of distant and close are considered hyperparemeters of the model and can also be used to reduce training time though a finer selection of pairs considered.

From a selected pair of instances d_i, d_j,the SNN is taught with a contrastive loss function to indicate whether the distance between embeddings, $\|r_i - r_j\|$, should be minimized in the case of *positive* examples or maximized in *negative* ones. Similarly, contrastive loss will scale up the learning gradient in the case of *hard* examples, scaling it down for *easy* ones.

4.5 Per-Instance Algorithm Prediction

The SNN has been trained to embed instances with similar APPs in close proximity. This involves translating an instances features into an embedding that represents the algorithm ensembles performance on that instance coupled with the instances performance relative to other instances in its surrounding area. In terms of classification, normally Siamese neural networks are used to calculate a similarity between a test instance and a labelled training instance, for example with face recognition to classify if two faces match or not. The SNN will then test an unseen test instance against all labelled face classes, calculating a similarity for each. This is an effective process in terms of faces where the number of test instances to be classified is usually in the region of singles to tens of images and classes to be tested against e.g. number of faces is in the same region in terms of amount.

For per instance algorithm selection this process is not optimal. For classification, the number of APPs could be in the region of thousands and the number of test instances to be classified also in the hundreds of thousands. To classify these instances by comparing class similarity would be extremely time consuming to the point any gains in terms of minimizing error would be offset by increased processing time to classify instances.

Instead a different approach of classification will be used for the SNN. The SNN has created an embedding replicating the performance space holding the APPs of similar performing instances. These APPs are small clusterings of similar performing instances. When an embedding vector for each test instance is created, it has a location within this embedding space. This dissertation will use a k-nearest neighbour classifier to group the most similar performing instances surrounding the test instance.

The k is a hyperparater that can be tuned depending on the needs of the dataset. An optimal k will need to be obtained and will be a unique to each dataset. Multiple clustering approaches will be explored as APPs do not have a set size and as such exploratory research will need to be conducted into the variance of APPs size and if a fixed k system is a suitable choice for classification. Using a clustering approach will provide increased performance in terms of time without hindering the effectiveness of the classification process of SNNs.

5 Methodology

5.1 Dataset

For this implementation, the Lending Club Loan dataset [55] was selected. This dataset was chosen due to its large number of columns and rows, 145 and 2.26 million respectively. Siamese neural networks (SNN) have been predominately used in facial recognition where a single image of a face can contain thousands of individual features such as pixel brightness, facial features and object contours among multiple others. A SNN will learn a similarity measure between two images based on these features. Regression tasks on numeric datasets tend to contain much smaller amounts of features ranging from tens to hundreds. The Lending Club Loan dataset was selected due to its large number of features 145, to mimic the large number of features present in images which historically has lead to successful SNN training.

The interest rate column was chosen as the prediction label, ranging from 0-30.99%. The dataset went through multiple prepossessing steps before being used in the pipeline. The initial dataset had multiple columns which contained missing data amounting to or above 5% of the total column data. Any rows which also contained missing values where removed from the dataset if an appropriate filling method was not suitable. Label encoding was used to encode categorical columns where needed. The final step of preprocessing was to normalize the data to improve the training efficiency of the Siamese neural network. The total number of rows containing missing data and subsequently removed amounted to 130 thousand and the total number of features removed where 71.

The final processed dataset, consisted of 2.13 million rows and 74 columns. 50,000 instances where randomly selected for algorithm suite training with the remaining data randomly split 90/10 to give a training and test split of 1.872 million and 208 thousand instances respectively.

5.2 Algorithm Suite Training

A suite of 8 regression algorithms were optimized and trained. The selection of algorithms was varied to minimize the absolute error of each predicted label using the strengths of each algorithm. Each algorithm was optimized using gridsearch cross validation where possible to find the optimal hyper parameters. The list of selected algorithms used in the ensemble along with their implementation libraries can be seen below.

- **Lasso Regression** - Scikit Learn [56]
- **Stoctic Gradient** Decent Regression - Scikit Learn
- **MLP Regressor** - Tensorflow [57]
- **CatBoost** - Yandex [58]
- **Random Forest Regression** - Scikit Learn
- **AdaBoost** - Scikit Learn
- **RANSAC** - Scikit Learn
- **Gradient Boosting** - Scikit Learn

A learning curve was created to measure a saturation point in terms of amount of data supplied to the models. A final value of 50,000 instances was chosen as training data for the algorithm suite. Above this value little to no accuracy increase was observed at the cost of increased training time. While the learning curve evens out quite quickly, 50,000 training instances where selected to allow for a fair sample from the total dataset to be trained on for each algorithm.

Once each algorithm was trained, it was asked to predict a label for the remaining 1.872 million instances and 208 thousand test instances. The predicted labels for the test data where put aside for final performance measurement of the SAS approach. The training data was taken forward in the pipeline for use in SNN training.

5.3 Pairing

The raw performance of the algorithm ensemble on the training data was converted to a final performance space using the novel performance metric introduced in equation 2. The metric takes the error between the algorithm ensembles predicted value and the target label for each instance and converts it to an augmented performance space. This metric allows for utilization of the entire performance space, exposure of relationships between similar performing instances and a scale to assess instance performance similarity for identification of APPs and training pairing.

Once the final performance of each instance has been calculated, pairing can occur. Introduced in section 4.4, instances needs to be paired for SNN training where a pairing indicates two instances are the same in terms of performance. Two approaches to pairing where implemented and will be discussed below.

5.3.1 Cluster Performance Pairing

To select two instances as a positive pair, the concept of APPs where introduced in section 4.4. An APP is a cluster in which instances enclosed are classed as "same" based on displaying similar performance characteristics. This approach aims to classify these clusters before pairing instances.

The augmented performance space is not directly suitable for clustering as clusters could overlap between the performance boundary where each algorithm is classified as performing strongest. Below in figure 5.1 an example of this is shown.

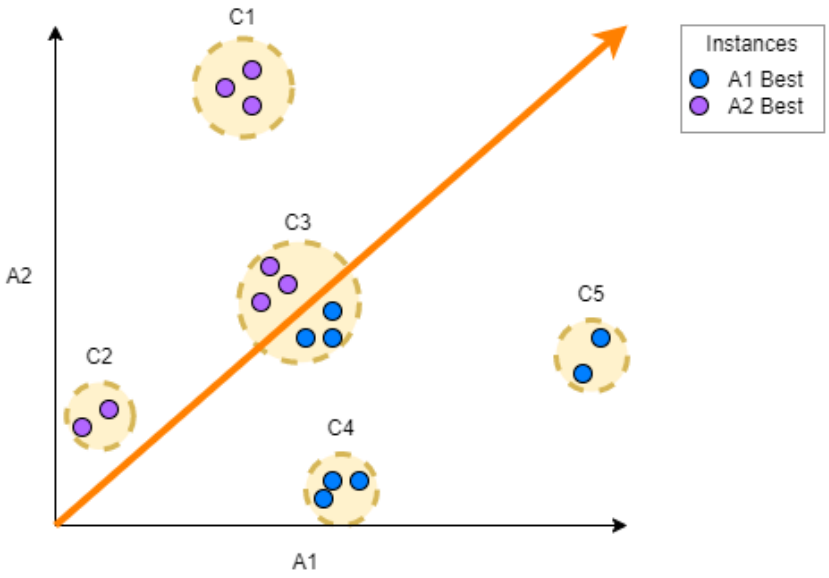


Figure 5.1: Clustering with two algorithms A1 and A2.

The orange line in figure 5.1 signifies the boundary to differentiate which algorithm performed strongest on a given instance. If clustering is applied to this space, clusters could form on these boundaries as shown by cluster C3 in figure 5.1. This will create a cluster with mixed performance in terms of strongest performing algorithm and will make labelling of new unseen instances for this cluster challenging.

This cluster pairing approach combats this problem by separating the performance space into their respective strongest performing algorithm before clustering is applied. This will mitigate the problem of clusters forming on boundaries between algorithm performance.

Once all instances have been separated into their respective strongest algorithm performances, clustering can take place. The k Means clustering algorithm is implemented from the scikit learn library. This algorithm aims to separate n instances into k clusters where k is set by the user. An ablation study was conducted with varying numbers of cluster sizes and 5 was selected as the optimal k value, this value will be dataset specific.

After clustering has taken place, pairing of positive and negative training pairs is carried out. First, instances inside each of the 5 clusters will be paired as positive training pairs. Negative pairs will be selected by picking intra cluster instances and pairing them. Once this has been conducted for each algorithm, intra algorithm pairing can occur. This involves pairing positive pairs as ones for which the same algorithm performed best regardless of cluster. Negative training pairs will be instances for which different algorithms performed best. This intra algorithm pairing will allow for greater distinction in the embedding space between strongest performing algorithm.

5.3.2 Distance Performance Pairing

To establish if two instances belong to an APP cluster a distance threshold was selected. This threshold will classify if a pair of instances belong to an APP. By using a distance threshold between two instances APPs will overlap, but this is acceptable behaviour of APPs as long as the algorithm performance boundary is respected which was discussed in the previous section. Performance classification of instances is not a binary task as instances from differing APPs will hold a similarity to instances from other APPs. By setting a distance threshold, the similarity of two instances must be above a certain amount to be classified as same for training.

This similarity threshold, set by a given distance, is calculated through analysis of both the feature and performance space is a hyperparameter of Siamese algorithm Selection. The training proportion of the dataset contains 1.872 million instances. A subsample amounting to 16,494 instances where randomly selected using a confidence level of 99% with a confidence interval of $\pm$ 1%. This allows for an accurate representation of the training data to be measured. The mean and median distance between all these instances where calculated to build a view of distance in both feature and performance space.

A selected distance threshold was set based on the results from the sample of data, taking the mean distance into account to set what is considered as similar in terms of distance for both spaces.

The term "hard", "easy", "positive" and "negative" are used to describe a classification in each space. In the performance space, two instances closer than the distance

threshold set are classified as positive and display a similarity above the set threshold. Whereas if the instance pair are further than the distance threshold they are classified as negative. This classification decides the label of 1 or 0 attached to pairs for Siamese Neural Network training. Below in figure 5.2 an example of these classification terms can be seen.

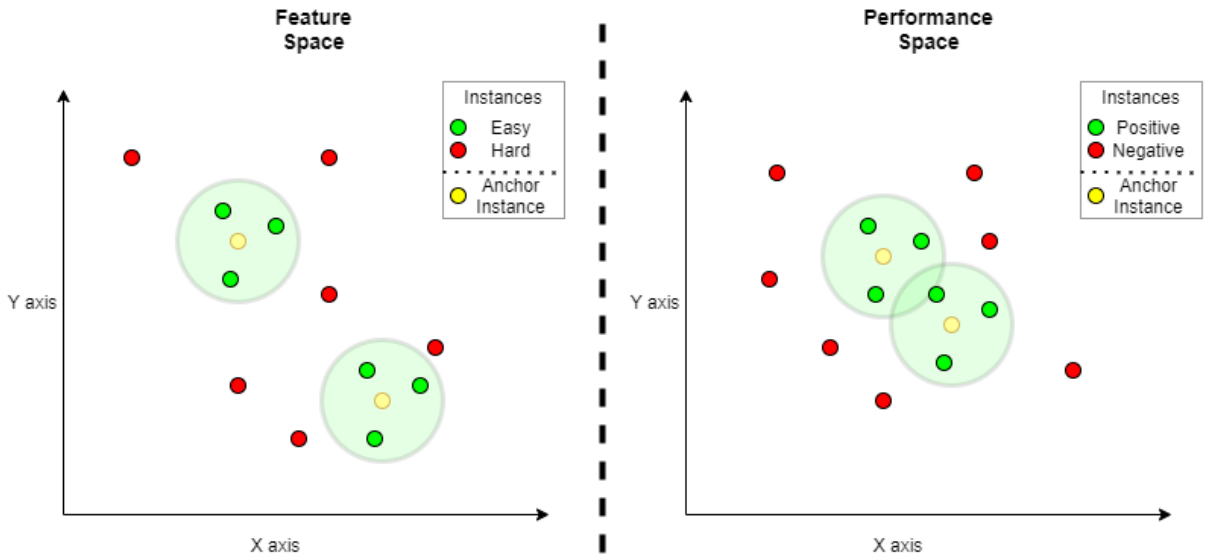


Figure 5.2: Examples of instances pairing in both feature and performance space.

In the feature space a second distance threshold is set. This threshold signifies "hard" and "easy" training pairs. This concept is introduced as it is cost prohibitive to pair and train all instances in a 1.874 million instance dataset.

For each pair of instances the SNN will learn a transformation which will embed instance features closer or further depending on its positive or negative label. If two instances features are close in the feature space this is classified as an easy training sample as the SNN will need to do minimal work to transform this pair.

This amount of hard and easy positive and negative training pairs is set manually and is a hyperparameter of the pipeline. The thresholds can be scaled up and down depending on the requirements of the task.

5.4 Siamese Neural Network Training

Once pairing of instances has occurred, the pairs are given to the Siamese neural network (SNN) for training. The goal of the SNN is to learn an transformation function that can transform an instance pairs features into an embedding that represents their performance similarity in terms of distance.

The multilayer perceptron Siamese Neural Network consists of 4 layers. The first consisting of 40 neurons, the second and third layer both consisting of 20 and the output layer consisting of 8. All layers used the Relu activation function with the exception of the output layer, this was done to allow for easier clustering in later stages as the Relu activation function can tend towards zero in a process known as necrosis [59]. The contrastive loss function, sometimes known as pairwise ranking loss, was used in this network. This distance based loss functions objective is to minimise distance d, between positive pairs and maximise the distance d between the negative pairs in the embedding space. The SNN was trained for 800 epochs before outputting a final embedding.

5.5 Algorithm Selection

The final learned embeddings from the SNN were used in the classification of test instances. All positive training pairs and test data were sent through the trained network to get a final embedding for each. Representing the training data in the embedding space, a k-nearest neighbour algorithm with varying k neighbours was run on each embedded test instance to find the k corresponding training instances. These k training instances with known algorithm performances were used to find the best performing algorithm to be chosen for each test instance. Two variants of the k nearest neighbour algorithm where run, the first a standard k nearest neighbour algorithm, and the second a distance weighted k nearest neighbour algorithm. Both algorithms where implemented from the scikit learn library.

Two further approaches where implemented on the K nearest neighbours to calculate the optimal performing algorithm for the cluster. The first approach implemented was a mean rank system to select the strongest algorithm from the selected neighbours. This involved calculating an average rank for each algorithm over k selected instances.

A second approach, a most often vote of algorithm performance rank, was also tested as an approach to decide on the best performing algorithm. For this approach in the event of a tie, lower ranks was taken into consideration to select a suitable algorithm.

5.6 Baselines

For comparison, the SAS approach was measured against a Random Forest (RF) regression baseline which was implemented based on work from Pulatov and Kotthoff [45]. While this work was later proven not to be a globally reproducible approach to per instance algorithm selection, it gives a baseline comparison of other state of the art approaches. The baseline was trained to predict an error vector on a per instance basis.

For training, the error vector was calculated from the predicted labels of all algorithms in the algorithm ensemble. The random forest regression algorithm was trained to learn the pattern between the error vector and instance for all training data. Once trained, the RF baseline predicts algorithm errors for test instances as a comparison to Siamese algorithm selection. The algorithm that the RF baseline predicts has the lowest error is chosen as the best performing algorithm for the given test instance.

Another baseline calculated during this process was the oracle, this is a theoretically perfect meta learner with 100% selection accuracy on a per instance basis. The oracle produces the lowest possible MAE for a theoretically perfect meta learner.

A random baseline is implemented as a control for the SAS pipeline. The random baseline selected an algorithm at random for each instance from the ensemble pool of 8 algorithms. While this baseline should easily be outperformed, it will give a control to measure SAS approach against.

The final baseline implemented is a mean weighted ensemble. Ensemble learning was discussed in related work and is a viable avenue to per instance algorithm selection. While the random forest baseline will act as the main baseline, the mean weighted ensemble will act as a indicator of overall algorithm ensemble performance. The mean weighted ensemble was created by averaging the predictions of the 8 algorithm predictions on a per instance basis to achieve a stable baseline which will be resistant to noise from outlier predictions of single algorithms.

6 Results and Discussion

This section will present the results of each step in the pipeline coupled with the final performance of Siamese algorithm selection against both a baseline and a global algorithm selection solution.

6.1 Algorithm Suite Performance

The final performance of each single trained algorithm in the ensemble is shown in table 6.1 below and was measured using R squared accuracy.

Algorithm	**R-Squared Accuracy**
Lasso	96.6 %
SGD	95.6 %
MLP Regressor	98.9 %
CatBoost	99.2 %
Random Forest	97.3 %
AdaBoost	93.5 %
RANSAC	96.5 %
Gradient Boosting	83.6 %

Table 6.1: The R-Squared accuracy of each algorithm on the the model training data consisting of 50,000 instances with a 80/20 train test split.

It is clear that the *MLP Regressor* and *Catboost* perform considerably stronger compared to other regression algorithms in the suite, achieving a R squared accuracy of at least 1.6% higher against all peer algorithms and 15.3% higher than the worst performer Gradient Boosting. All algorithms within the suite perform to a very high standard with the worst performer Gradient Boosting, achieving a high R squared accuracy of 83.6%.

In table 6.2 below, the performance rank of each algorithm measured against its peers is shown. The performance rank is a measure out of a possible 100%, for how many instances a given algorithm was the best performing and hence the most suitable selection choice.

Rank	Lasso %	SGD %	CatBoost %	MLP %	Random Forest %	Ada Boost %	RSNAC %	Gradient Boosting %
1st	5.85	6.81	31.96	32.15	8.69	4.74	5.84	3.97
2nd	7.03	7.11	29.42	29.40	10.38	4.97	7.03	4.65
3rd	11.06	10.08	17.46	15.79	18.20	7.27	11.94	8.20
4th	18.48	11.95	9.98	8.37	17.87	8.17	18.04	7.15
5th	23.24	13.30	6.49	4.92	18.04	6.88	22.36	4.78
6th	20.84	21.25	4.08	2.62	16.10	9.06	20.55	5.51
7th	11.00	20.79	1.90	1.04	8.86	30.90	11.88	13.63
8th	2.81	9.59	0.25	0.31	2.75	28.71	2.88	52.70

Table 6.2: Performance of each algorithm at each rank, with 1st being the optimal rank.

An interesting finding between table 6.1 and 6.2 shows while CatBoost is the best performing algorithm in training achieving an R squared accuracy of 99.2%, the MLP regressor algorithm holds the largest selection share of first rank on the SNN training data. This indicates slight overfitting by CatBoost on the model training data.

The best performing algorithm MLP regressor from table 6.2 achieves a selection share of 32.15% for 1st rank. Selection share is measured out of 100% meaning for 67.85% of instances, other algorithms in the suite are stronger performing and a more suitable choice. This provides evidence of the per instance algorithm selection problem introduced in section 1.1, that the globally best performing algorithm is not the most suitable choice for every instance within the dataset.

Algorithm	**MAE**
MLP Regressor	0.212
CatBoost	0.236
Random Forest	0.549
SGD	0.641
RANSAC	0.655
Lasso	0.742
AdaBoost	1.09
Gradient Boosting	1.44

Table 6.3: Each ensemble algorithm listed with the individual performance on the test dataset, measured using the MAE metric.

Table 6.3 above shows the final performance, measured using MAE, of each single algorithm on the unseen validation data.The MLP Regressor algorithm records the best performance from the ensemble achieving a final MAE of 0.212. This is an important measure and is classified as the global solution to the algorithm selection problem. A goal of Siamese algorithm selection is to achieve an MAE below this value.

6.2 Performance Metric Visualized

An important aspect of the pipeline is the performance metric used to normalize the performance space and allow for easier identification of APP clusters. An analysis of the performance metric on the Kaggle loan dataset will be carried out to visualize if the performance metric is carrying out the desired functionality. Below, figure 6.1 shows a subsection of the raw performance space plotted using the MAE metric.

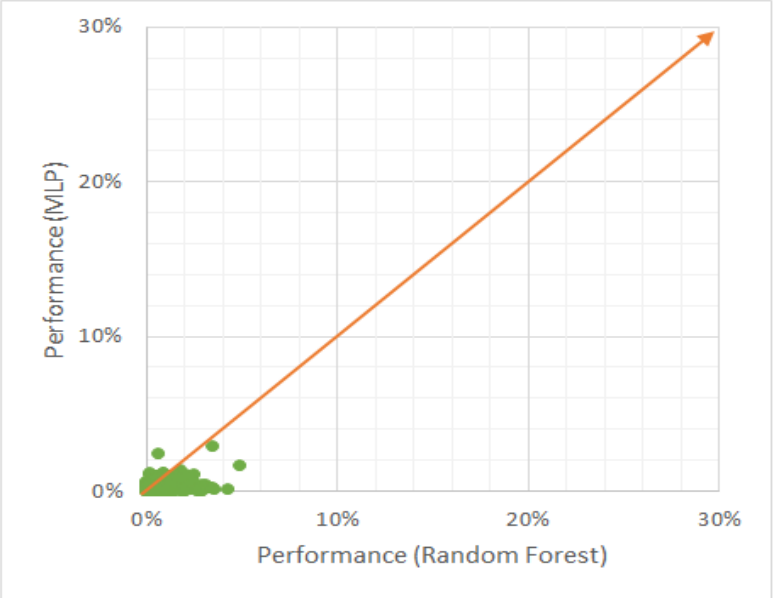


Figure 6.1: MAE measurement for 1000 instances on the Kaggle loan dataset for algorithms Random forest regressor (x axis) and MLP Regressor (y axis) where 0% equates to optimal performance and interest rate ranges from 0-30%.

Both algorithms Random forest Regressor and MLP Regressor perform strongly which correlates to occupying the bottom left segment of the graph. Using MAE to separate instances into APPs would be challenging as the distance between instances is minuscule.

The orange line on the graph signifies the selection boundary for which algorithm performed stronger. Instances on this boundary will be harder to predict as both algorithms performed similarly with a slight performance differences either side of the boundary. From figure 6.1 above, MAE does little to separate instances at this boundary which is needed for clearer APP clustering.

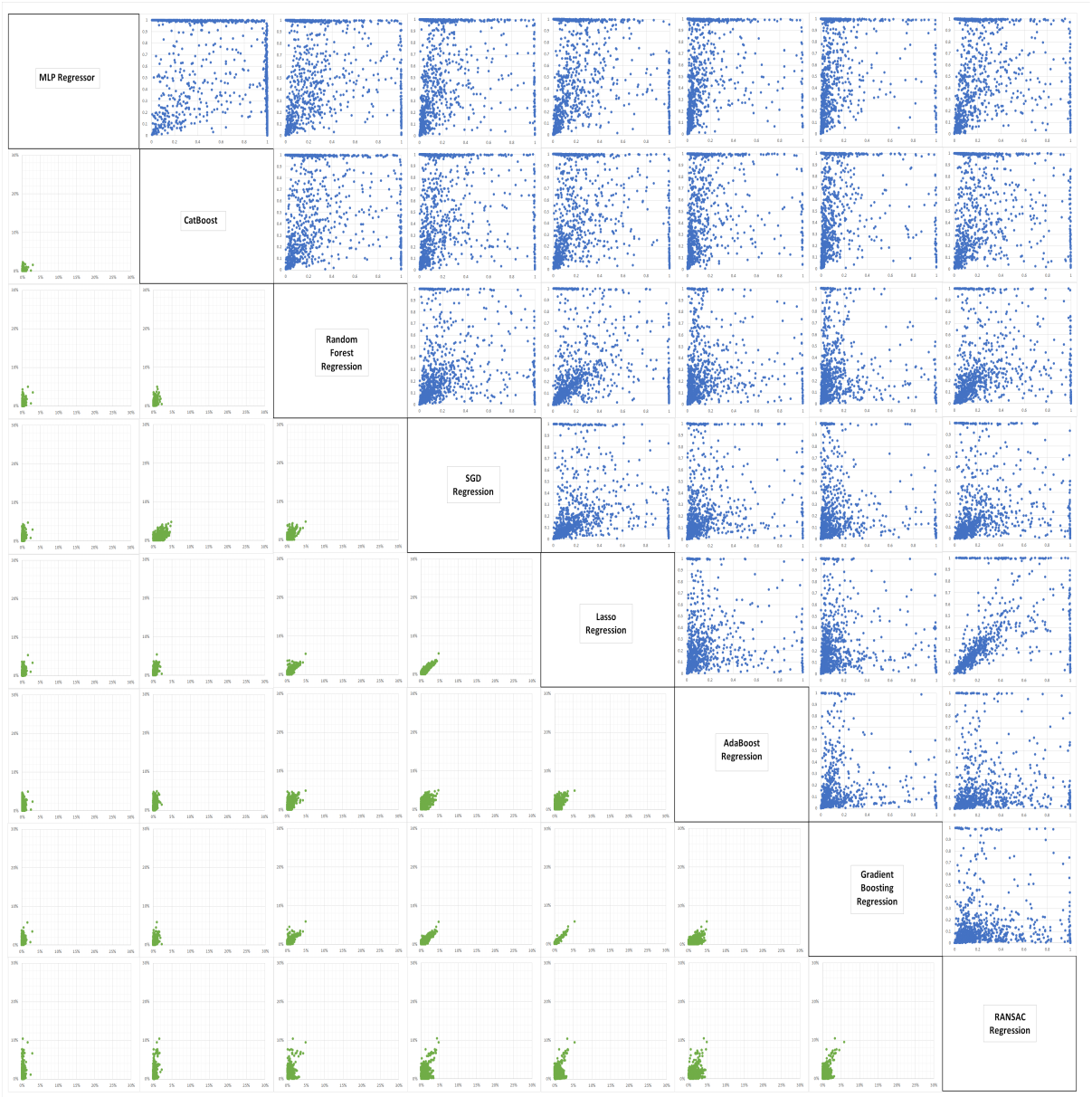


Figure 6.2: Algorithm comparison for 1000 sample points in the both performance spaces. Performance measured using MAE (green points) and the novel performance metric (blue points).

In figure 6.2 above the novel performance metric (blue points) is plotted and compared against the raw performance space (green points) using the MAE measurement of 1000 instances. The raw performance graphs for all algorithms show dense clusterings of points in the bottom left corner of the graph as all algorithms perform similarly on the given instances. The augmented performance space spreads instances across the entire space, moving easier to classify instances to the fringes of the graphs. For easier visualization three plots of each space will be extracted and discussed in figure 6.3 and 6.4 below.

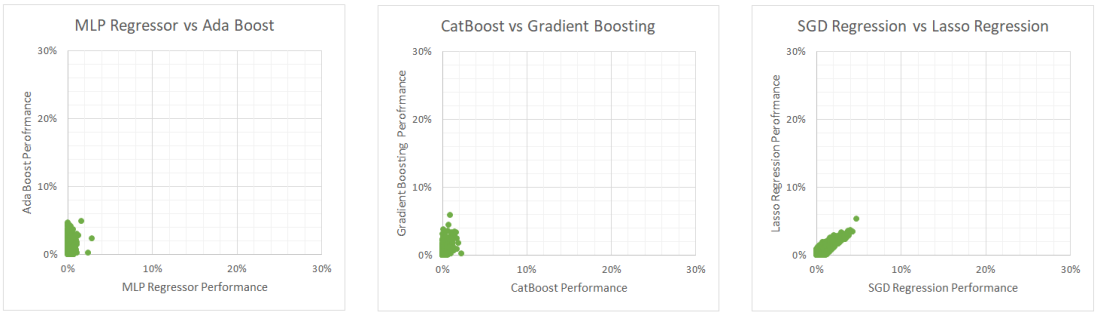


Figure 6.3: Algorithm comparison for 1000 sample points in the raw performance space, performance measured using MAE.

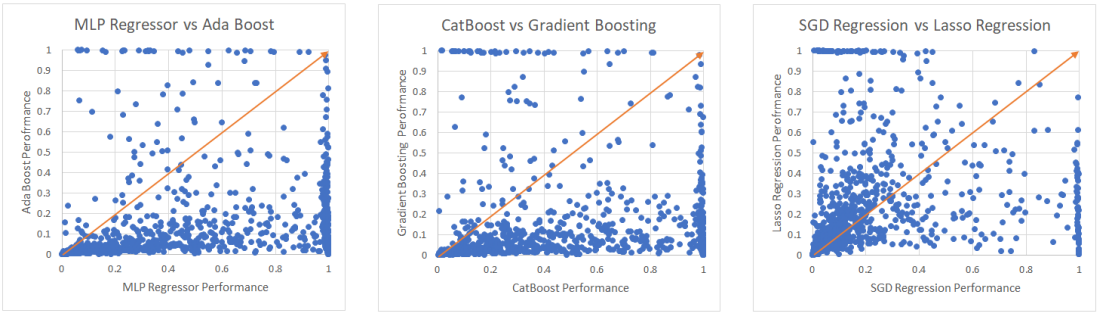


Figure 6.4: Algorithm comparison for 1000 sample points in the augmented performance space, performance measured using the novel performance metric.

From figure 6.3 above a comparison between the same algorithms in the raw performance space and the novel metric performance space can been seen. From the augmented space in figure 6.4 the entire space is utilized to represent instance performance which is not being carried out in the raw performance space. The augmented performance space also spreads instances away from the decision boundary between the two algorithms, this allows for easier identification of which algorithm performs best for a given instance and easier to identify APPs.

Another important aspect of the augmented performance space is the rank of the best performing algorithm does not change. A goal of Siamese algorithm selection is to correctly select the best performing algorithm, if a metric changes the ranking of algorithms this will lead to false training of algorithm performance of instances.

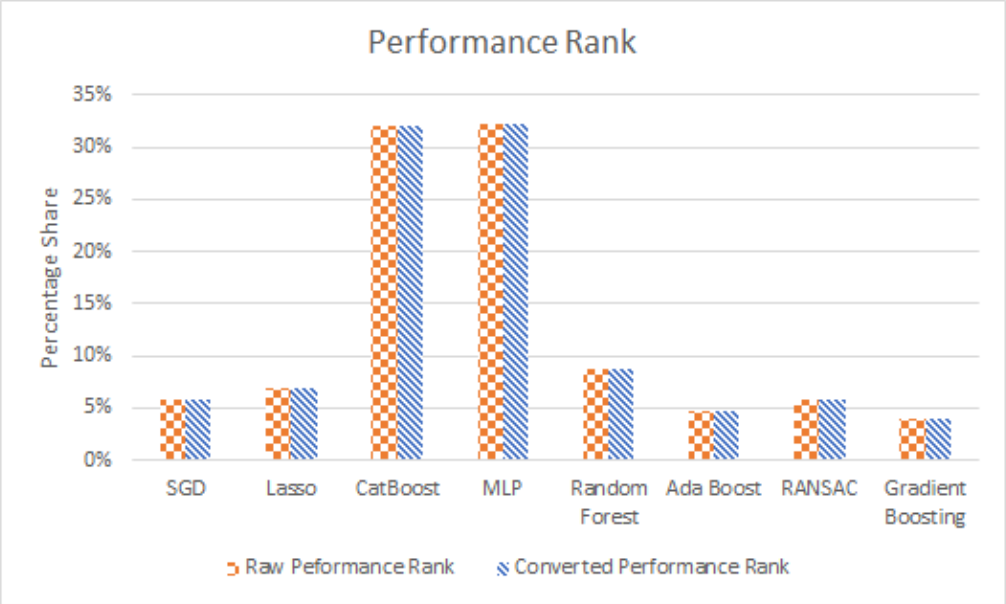


Figure 6.5: Percentage share for each single algorithm before and after the novel performance metric is applied.

Shown in figure 6.5 above the overall rank of best performing algorithm on instances does not change after the performance data is converted. This is backed up by a 0% change in instance rank when measured on a per instance level. This is a desired outcome of the performance metric.

6.3 Evaluating the Siamese Neural Network

In figure 6.6 and 6.7 below, the SNNs loss and learning curve are displayed respectively. The final recorded accuracy of the SNN was 0.7415 and loss value 0.177. In terms of learning the SNN, like many deep learning models show rapid initial progress before steadying at 100 epochs to a more gradual learning rate. This curve seems to maintain its gradient up until 600 epochs where a flattening of the curve occurs and shows diminishing returns on training.

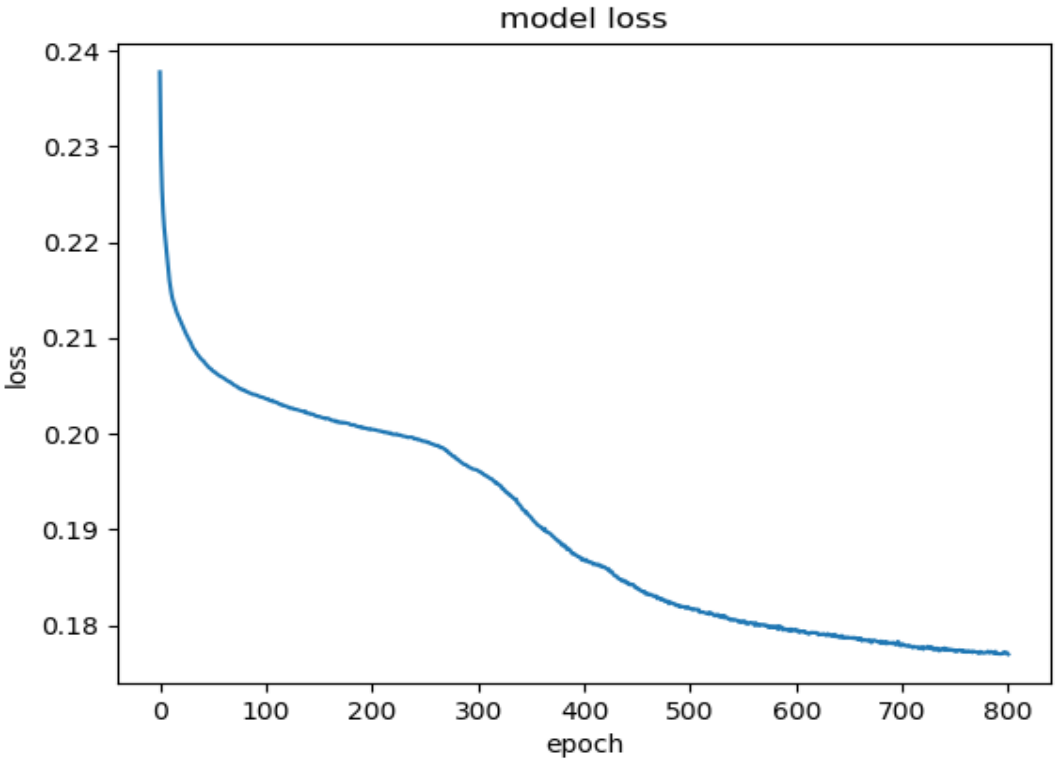


Figure 6.6: Siamese neural networks loss curve recorded over 800 epochs.

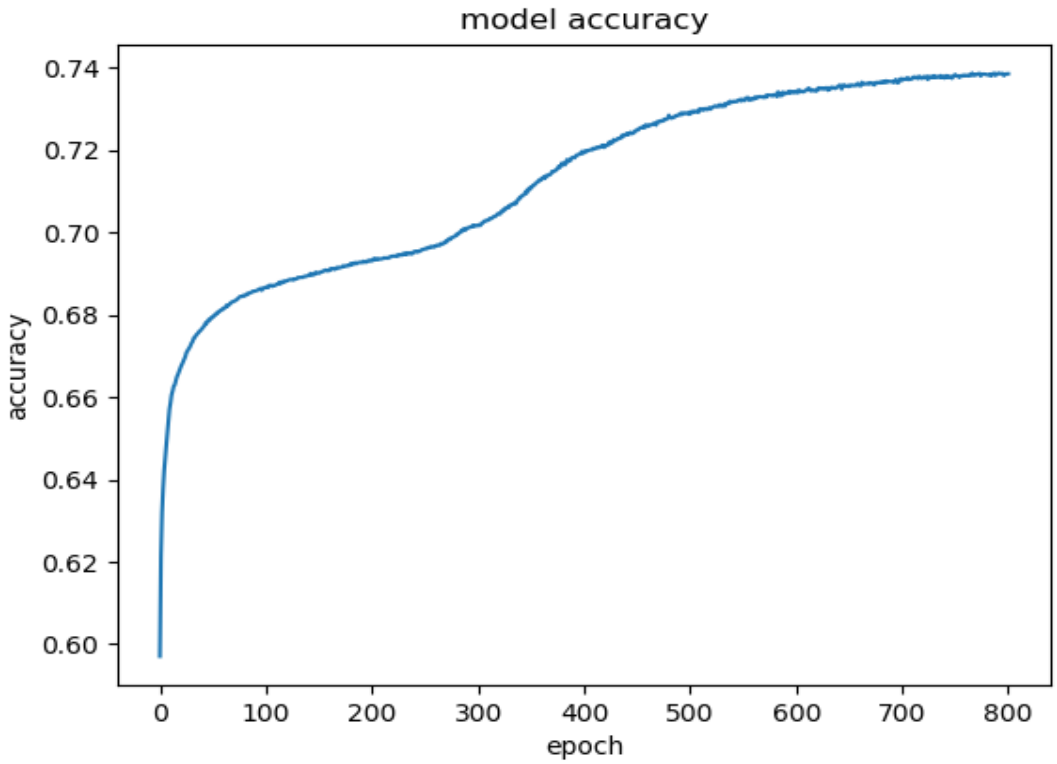


Figure 6.7: Siamese neural networks learning curve recorded over 800 epochs.

Using principal component analysis (PCA) to reduce the embedding space to two dimensions, the embedding space is plotted for epoch 1 and the final epoch to visualize how the SNN is embedding the training instances in the embedding space.

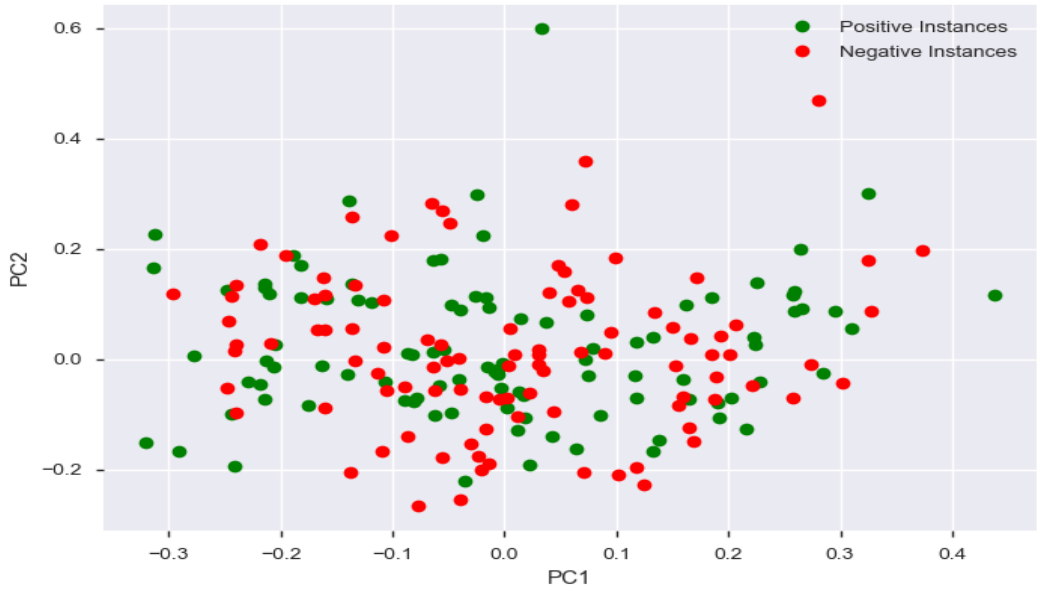


Figure 6.8: Siamese neural networks embedding space for 100 random points after 1 epoch with the distance pairing system.

Figure 6.9: Siamese neural networks embedding space for the same random 100 points after 800 epochs with the distance pairing system.

For the final embedding space shown in figure 6.9 above, small clusters of positive training instances, shown by green dots, can be identified. They seem to amalgamate to form a larger positive clustering of instances. This behaviour is not optimal as less differentiation between APP clusterings will inhibit different performance patterns from being represented.

This behaviour is due to the distance pairing system employed by the pipeline. The pairing system selects pairs of instances at random and taking their distance in the performance space, applies a positive or negative label. The SNN treats this label as a class moving negative instances away from the class and moving positive instances closer together. This will in turn move all positive instances closer, leading to the effect of a singular class as can be seen above.

The embedding space will still hold valuable information from the performance space as positive instances will embed closer and form small clusterings which represent APPs. The fear of the SNN fully amalgamating all smaller clusters into a singular cluster can be alleviated since the contrastive loss function employed by the SNN has a margin above which no further movement of embedding instances will occur, creating a final settled embedding space.

6.4 Evaluating Siamese algorithm selection

It is found that Siamese Algorithm Selection method works and successfully outperforms any single algorithm. In table 6.4 its performance is recorded, along with the three single best performing algorithms with the remaining 5 omitted due to worse performance. The performance of RF baseline and oracle baseline are also shown.

Algorithm	**MAE**	**Reduction in MAE**	**Selection Accuracy**
RF Regressor	0.549	−158.96%	8.5%
CatBoost	0.236	−11.32%	31.9%
MLP Regressor	**0.212**	**0%**	**34.7%**
Random Selection	0.695	−327.83%	12.23%
Mean Weighted Ensemble	0.605	−285.37%	*NA*%
RF baseline	**0.176**	**16.98%**	**50.0%**
SNN with 3-*NN*	0.211	0.47%	41.8%
SNN with 5-*NN*	0.201	5.18%	42.5%
SNN with 16-*NN*	0.194	8.49%	42.8%
SNN with 32-*NN*	0.186	12.26%	43.0%
SNN with 128-*NN*	**0.180**	**15.09%**	**43.2%**
Oracle Selection	**0.088**	**58.49%**	**100.0%**

Table 6.4: The mean absolute error (MAE), the percentage reduction in MAE from the best performing algorithm *MLP Regressor*, selection accuracy of our Siamese Algorithm Selection and baseline.

Its found that an oracle baseline, one which correctly selects the best performing algorithm for each instance, would lead to a 58% reduction in MAE over choosing the single best algorithm, *MLP Regressor*.

Siamese Algorithm Selection with 128 neighbours manages to achieve a 15% reduction in MAE, a definite improvement over the single best algorithm. The RF baseline performed slightly better than the Siamese Algorithm Selection approach, reducing possible MAE by 17% with a higher selection accuracy of 50%.

The RF baseline achieves a slightly lower MAE of 0.176 representing an additional reduction of 0.004 over SAS. This is on a par with the SAS approach but with a less resource intensive pipeline. While the RF baseline was proven not to be a global solution to per instance algorithm selection, the SAS approach, while promising will need further improvement to warrant selection over the RF baseline approach.

Two further approaches, "random selection" and "mean weighted ensemble" where also implemented. The random selection baseline achieves a poor performance of 0.695

MAE. Its selection accuracy of 12.23% is expected for a random selection from 8 ensemble algorithms. The mean weighted ensemble approach achieves a MAE of 0.605, while better than random ensemble, is still very weak compared to other approaches. Both approaches, in terms of MAE, rank around 4^{th} place within the single algorithm ensemble listed in table 6.3, and both are not suitable approaches to per instance algorithm selection.

The SNN has learned to identify some APPs with its selection accuracy from table 1, 8.5% above choosing the single best algorithm. While increasing neighbours, it was found that selection accuracy remains around 42% for $3, 5, 16, 32, 128$ neighbours but its MAE begins to improve. As part of the investigation as to why, the percentage that each single algorithm was selected is considered in Table 6.5.

Algorithm	**Oracle**	**RF Baseline**	**Siamese Algorithm Selection**
MLP Regressor	32.15 %	47.33 %	46.23%
CatBoost	31.96 %	31.95 %	42.94%
Random Forest	8.69 %	7.57 %	4.69%
Lasso	5.85 %	4.27 %	2.18 %
SGD	6.81 %	4.75 %	2.16 %
RANSAC	5.84 %	1.16 %	0.23 %
Gradient Boosting	4.74 %	1.46 %	1.33 %
AdaBoost	3.97 %	1.51 %	0.23 %

Table 6.5: A comparison between the algorithm selections of Siamese Algorithm Selection with 128 neighbours, the Random Forest baseline and an oracle selection.

From table 6.5, 89% of the total algorithm selection for Siamese algorithm selection consisted of *CatBoost* and *MLP Regressor*. While these algorithms should be selected most often due to their high performance, it shows an over reliance on stronger performing algorithms. This would explain why accuracy did little to improve but MAE decreased as these two algorithms generally provided a good prediction for each instance, even if not the best preforming algorithm for said instance.

In table 6.6 the selection share for each single algorithm is considered for all K neighbours.

An interesting pattern emerges from this data in 6.6, the selection share for smaller algorithms decreases as the number of neighbours increases. In contrast, the two best performing algorithms Catboost and MLP Regressor see an increase in their selection share at higher k neighbour values.

Algorithm	Oracle	SNN 3-*NN*	SNN 5-*NN*	SNN 16-*NN*	SNN 32-*NN*	SNN 128-*NN*
MLP Regressor	32.15 %	35.88 %	37.54 %	41.43 %	43.92 %	46.23 %
CatBoost	31.96 %	35.34 %	36.81 %	40.10 %	41.49 %	42.94 %
Random Forest	8.69 %	7.85 %	7.33 %	5.97 %	5.28 %	4.69 %
Lasso	5.85 %	5.34 %	4.84 %	3.65 %	2.95 %	2.18 %
SGD	6.81 %	5.31 %	4.75 %	3.57 %	2.90 %	2.16 %
RANSAC	5.84 %	4.50 %	3.82 %	1.98 %	1.06 %	0.23 %
Gradient Boosting	4.74 %	2.93 %	2.69 %	2.11 %	1.69 %	1.33 %
AdaBoost	3.97 %	2.85 %	2.23 %	1.18 %	0.69 %	0.23 %

Table 6.6: Selection share for each algorithm class chosen by the SAS pipeline for varying K neighbour values.

This is attributed to the pairing system employed in the pipeline. In figure 6.9, using the distance pairing approach, small clusters of positive training instances are surrounded by a larger positive cluster of instances. These smaller clusters will give accurate algorithm classification for low numbers of neighbours, as the number of neighbours increases other APP clusters are counted in the neighbour circumference. MLP regressor and catboost have the largest selection share of best performing algorithms, this will give both algorithms larger numbers of APP clusters. As neighbours are increased, MLP regressor and catboost have stronger influence on algorithm selection meaning their percentage share of selection increases at the expense of weaker performing algorithms.

Since MLP regressor and Catboost are both the globally strongest performing algorithms, selecting these algorithms, even if incorrect in doing so, can lead to a reduction in MAE due to these algorithms having a low errors even when not the strongest performing algorithm.

While an over-reliance on MLP regressor and Catboost has been established, it should be noted that the selection share is 8.5% above the single best algorithm. This is a very promising finding and shows SAS is correctly selecting an algorithm for a large proportion of instances, for both low and high numbers of neighbours.

6.4.1 Pairing Approaches

Two separate pairing approaches where employed in the pipeline, **distance pairing** and **clustering pairing**. Through ablation testing of the pipeline, both pairing systems where tested for 5 runs with 3 nearest neighbours with the results shown in table 6.7 below.

Distance pairing outperforms clustering pairing by an MAE 0.058 over 5 runs. Distance pairing is the simpler of approaches, picking pairs of instances based on thresholds. These thresholds can be estimated by sampling feature and performance space to select thresholds and as such, is an easy approach to optimize.

Run	Distance Pairing (MAE)	Clustering Pairing (MAE)
1st run	0.213	0.267
2nd run	0.211	0.277
3rd run	0.210	0.267
4th run	0.210	0.268
5th run	0.214	0.272
average	**0.212**	0.270

Table 6.7: Comparison of Pairing Approaches for 5 separate runs of the pipeline.

Clustering pairing is a more complex approach, creating pairs for individual clusters along with inter algorithm performance pairing. This pairing system has more hyperparameters and requires much finer tuning to optimize the approach. Both approaches perform well achieving strong MAE recordings, but distance paring records stronger MAE making it the pairing approach of choice for this pipeline.

6.4.2 *k*-Nearest Neighbour Approaches

Two nearest neighbour approaches where tested in the pipeline, **weighted K nearest neighbours** and **standard K nearest neighbours**, with a distance pairing system. Over 5 runs with unique embeddings, both approaches where tested with 3, 5 and 16 neighbours and averaged shown in table 6.8 below.

Run	standard *k*-NN (MAE)			Weighted *k*-NN (MAE)		
	3-NN	**5**-NN	**16**-NN	**3**-NN	**5**-NN	**16**-NN
1st run	0.211	0.202	0.193	0.223	0.212	0.202
2nd run	0.214	0.203	0.196	0.227	0.212	0.203
3rd run	0.212	0.201	0.194	0.225	0.210	0.201
4th run	0.213	0.203	0.195	0.226	0.212	0.204
5th run	0.212	0.201	0.194	0.225	0.211	0.201
average	**0.212**	**0.202**	**0.194**	0.225	0.211	0.202

Table 6.8: Comparison of clustering Approaches for 5 separate runs of the pipeline with varying k neighbours.

The Standard KNN algorithm outperforms distance weighted KNN for all number of neighbours in table 6.8 above. The MAE of distance weighted KNN lags standard KNN, but does increase with increasing neighbours, suggesting it takes more neighbours for distance weighted KNN to record the same MAE as standard KNN.

Closer examination of the embedding space in figure 6.9 shows that multiple clusters of instances are located in close proximity. Using a KNN algorithm, clusters located close to the target cluster will influence the outcome of both KNN algorithms. The distance weighed algorithm will weight these neighbouring clusters with less importance and will need more neighbours to override the target cluster, which is an desired behaviour.

With larger neighbour numbers, strong ensemble algorithms seem to override the original prediction, which will lead to a low MAE even if incorrect but stagnate selection accuracy. A flexible K value should be considered in future work to offset stronger ensemble algorithms overriding smaller algorithm clusters.

7 Conclusion

This dissertation has focused on the per instance algorithm selection problem, a problem that exists across datasets and domains. While multiple solutions such as Auto Weka and Auto SK-learn have been implemented and proven to work consistently to tackle the global algorithm selection problem, approaches on the per instance level have not had the same degree of success. Similar approaches in the per instance domain such as Pulatov and Kotthof [43] have shown mixed and inconsistent results in their approach and as such the per instance algorithm selection problem remains to this day.

Siamese Algorithm Selection, consisting of a Siamese Neural Network trained with 'Algorithm Performance Personas' (APP) and a novel performance metric successfully identified APPs of instances, to reduce the MAE by 15% over that of the single best algorithm MLP regressor, considered a global solution.

This dissertation has shown that distance in the performance space with augmentation, can be leveraged for training a neural network to transform instance features into embeddings, where distance of embeddings correlates with distance in performance space.

While a 15% reduction in MAE has been achieved, there is still room from improvement with a maximum MAE reduction of 58% possible from an optimal per instance approach. Overall this approach has shown to be useful and a promising future direction for per-instance algorithm selection.

Below the key contributions of this dissertation are listed:

- Siamese Algorithm Selection, a novel approach to the per instance algorithm selection problem, has been implemented with success, showing greater accuracy compared to a global solution.
- An implementation of a novel performance metric, which allows for instance performance to be measured in terms of instance peer similarity and algorithm suite performance.
- This dissertation has successfully utilized performance space as a tool for prediction of future instance performance.
- A thorough search of the relevant literature yielded that this is the first attempt to utilize a Siamese Neural Network for to tackle the problem of per instance algorithm selection.
- All data and code written have been made available in open source to encourage reproducibility and build further on the results obtained in this dissertation.

8 Limitations and Future Work

While the results show promise on the Kaggle Loan dataset, a comprehensive look over multiple datasets are required to assess the generality of Siamese algorithm selection as a per instance selection solution. All single algorithms within the ensemble perform to a very high standard on the kaggle loan dataset with the lowest recorded R squared accuracy of 83.6%. A more difficult task should be included in future work to assess the generality of SAS with weaker performing ensemble algorithms. The OpenML dataset [60] would be a promising direction for both standardized datasets but also to incorporate more information regarding algorithm performances.

This dissertation implemented a new metric to normalize performances, accounting for both the maximum possible error and the relative sizes of error. In practice this has worked but it's recognized a potential divide by 0 error exists if an algorithm achieves no error. Improvements to the metric should be considered for future work.

APPs have been successfully used in the training of an embedding space to classify similar performing instances. APPs are a theoretical concept to which multiple implementations can be created. In this dissertation two separate pairing approaches have been implemented and discussed but these approaches are a small subset of the possible approaches that can be taken.

One promising idea for future work is with the contrastive loss function. In current pairing systems, a discrete label of one or zero is used to classify training instances. The constrastive loss function has the ability to accept continuous values in the range of 0 to 1. This would compliment the augmented performance space as similarity is also a scale rather than a discrete value. A promising line of future work is to create a similarity scale that can be used when labelling training data for the SNN.

For algorithm selection, a k-nearest neighbour algorithm with fixed neighbour sizes was used to determine APP clusters in the embedding space. A fixed k value may not be optimal to select APP clusters, as increasing k tended to favour stronger performing ensemble algorithms. A flexible k should be explored in future work to allow for differing APP cluster sizes to be captured.

Bibliography

[1] John R. Rice. The algorithm selection problem. *Advances in computers*, 15 (65-118):5, 1976.

[2] José Carlos Monteiro Rui Rodrigues. Review of the algorithm selection. *Eighth International Conference on Artificial Intelligence and Soft Computing*, 2008.

[3] Sebastian Raschka. Model evaluation, model selection, and algorithm selection in machine learning. *arXiv preprint arXiv:1811.12808*, 2018.

[4] Lars Kotthoff, Ian P Gent, and Ian Miguel. A preliminary evaluation of machine learning in algorithm selection for search problems. In *Fourth Annual Symposium on Combinatorial Search*, 2011.

[5] Joeran Beel, Corinna Breitinger, Stefan Langer, Andreas Lommatzsch, and Bela Gipp. Towards reproducibility in recommender-systems research. *User Modeling and User-Adapted Interaction (UMUAI)*, 26(1):69–101, 2016. doi: 10.1007/s11257-016-9174-x.

[6] Andrew Collins, Jöran Beel, and Dominika Tkaczyk. One-at-a-time: A meta-learning recommender-system for recommendation-algorithm selection on micro level. *CoRR*, abs/1805.12118, 2018. URL `http://arxiv.org/abs/1805.12118`.

[7] Joeran Beel, Akiko Aizawa, Corinna Breitinger, and Bela Gipp. Mr. dlib: Recommendations-as-a-service (raas) for academia. In *Proceedings of the 17th ACM/IEEE Joint Conference on Digital Libraries*, JCDL '17, pages 313–314, Piscataway, NJ, USA, 2017. IEEE Press. ISBN 978-1-5386-3861-3. URL `http://dl.acm.org/citation.cfm?id=3200334.3200389`.

[8] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

[9] Frank Hutter, Lars Kotthoff, and Joaquin Vanschoren, editors. *Automatic Machine Learning: Methods, Systems, Challenges*. Springer, 2019.

[10] Jane Bromley, Isabelle Guyon, Yann LeCun, Eduard Säckinger, and Roopak Shah. Signature verification using a" siamese" time delay neural network. In *Advances in neural information processing systems*, pages 737–744, 1994.

[11] Luca Bertinetto, Jack Valmadre, João F. Henriques, Andrea Vedaldi, and Philip H. S. Torr. Fully-Convolutional Siamese Networks for Object Tracking. 2016. URL `http://arxiv.org/abs/1606.09549`.

[12] Jonas Mueller and Aditya Thyagarajan. Siamese Recurrent Architectures for Learning Sentence Similarity. In *Thirtieth AAAI Conference on Artificial Intelligence*, 2016. URL `https://www.aaai.org/ocs/index.php/AAAI/AAAI16/paper/view/12195`.

[13] Ke Chen and Ahmad Salman. Extracting Speaker-Specific Information with a Regularized Siamese Deep Network. In J. Shawe-Taylor, R. S. Zemel, P. L. Bartlett, F. Pereira, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 24*, pages 298–306. Curran Associates, Inc., 2011. URL `http://papers.nips.cc/paper/4314-extracting-speaker-specific-information-with-a-regularized-siamese-deep-netw pdf`.

[14] W. Hayale, P. Negi, and M. Mahoor. Facial expression recognition using deep siamese neural networks with a supervised loss function. In *2019 14th IEEE International Conference on Automatic Face Gesture Recognition (FG 2019)*, pages 1–7, 2019.

[15] Gregory Koch, Richard Zemel, and Ruslan Salakhutdinov. Siamese neural networks for one-shot image recognition. In *ICML deep learning workshop*, volume 2. Lille, 2015.

[16] Xiu-Shen Wei, Peng Wang, Lingqiao Liu, Chunhua Shen, and Jianxin Wu. Piecewise classifier mappings: Learning fine-grained learners for novel categories with few examples. *CoRR*, abs/1805.04288, 2018. URL `http://arxiv.org/abs/1805.04288`.

[17] Xiu-Shen Wei, Peng Wang, Lingqiao Liu, Chunhua Shen, and Jianxin Wu. Piecewise classifier mappings: Learning fine-grained learners for novel categories with few examples. *IEEE Transactions on Image Processing*, 28:6116–6125, 2019.

[18] Shubham Panchal. Face recognition from scratch using siamese networks and tensorflow, Aug 2019. URL https://medium.com/predict/face-recognition-from-scratch-using-siamese-networks-and-tensorflow-df03e32f8cd0.

[19] Raia Hadsell, Sumit Chopra, and Yann LeCun. Dimensionality reduction by learning an invariant mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006.

[20] Sunghwan Kim, Taesung Lee, Seung-won Hwang, and Sameh Elnikety. List intersection for web search: algorithms, cost models, and optimizations. *Proceedings of the VLDB Endowment*, 12(1):1–13, 2018.

[21] Enrique Leyva, Yoel Caises, Antonio González, and Raúl Pérez. On the use of meta-learning for instance selection: An architecture and an experimental study. *Information Sciences*, 266:16–30, 2014.

[22] Lin Xu, Frank Hutter, Holger H Hoos, and Kevin Leyton-Brown. Satzilla: portfolio-based algorithm selection for sat. *Journal of artificial intelligence research*, 32:565–606, 2008.

[23] Matthias Feurer, Aaron Klein, Katharina Eggensperger, Jost Springenberg, Manuel Blum, and Frank Hutter. Efficient and robust automated machine learning. In C. Cortes, N. D. Lawrence, D. D. Lee, M. Sugiyama, and R. Garnett, editors, *Advances in Neural Information Processing Systems 28*, pages 2962–2970. Curran Associates, Inc., 2015.

[24] Felix Mohr, Marcel Wever, and Eyke Hüllermeier. Ml-plan: Automated machine learning via hierarchical planning. *Machine Learning*, 107(8-10):1495–1515, 2018. doi: 10.1007/s10994-018-5735-z. URL https://doi.org/10.1007/s10994-018-5735-z.

[25] Bruno Almeida Pimentel and André CPLF de Carvalho. A meta-learning approach for recommending the number of clusters for clustering algorithms. *Knowledge-Based Systems*, page 105682, 2020.

[26] Wei-Wei Tu. The 3rd automl challenge: Automl for lifelong machine learning. In *NIPS 2018 Challenge*, 2018.

[27] Murchhana Tripathy and Anita Panda. A study of algorithm selection in data mining using meta-learning. *Journal of Engineering Science & Technology Review*, 10(2), 2017.

[28] Joeran Beel and Lars Kotthoff. Preface: The 1st interdisciplinary workshop on algorithm selection and meta-learning in information retrieval (amir). In *Proceddings of The 1st Interdisciplinary Workshop on Algorithm Selection and Meta-Learning in Information Retrieval (AMIR)*, pages 1–9, 2019.

[29] Nicola Ferro, Norbert Fuhr, Gregory Grefenstette, Joseph A Konstan, Pablo Castells, Elizabeth M Daly, Thierry Declerck, Michael D Ekstrand, Werner Geyer, Julio Gonzalo, et al. From evaluating to forecasting performance: How to turn information retrieval, natural language processing and recommender systems into predictive sciences. *Dagstuhl manifestos*, 2018.

[30] Ben He and Iadh Ounis. Inferring query performance using pre-retrieval predictors. In *International symposium on string processing and information retrieval*, pages 43–54. Springer, 2004.

[31] Joel Mackenzie, J Shane Culpepper, Roi Blanco, Matt Crane, Charles LA Clarke, and Jimmy Lin. Query driven algorithm selection in early stage retrieval. In *Proceedings of the Eleventh ACM International Conference on Web Search and Data Mining*, pages 396–404, 2018.

[32] Joeran Beel, Alan Griffin, and Conor O'Shey. Darwin & goliath: Recommendations-as-a-service with automated algorithm-selection and white-labels. In *13th ACM Conference on Recommender Systems (RecSys)*, 2019.

[33] Il Im and Alexander Hars. Does a one-size recommendation system fit all? the effectiveness of collaborative filtering based recommendation systems across different domains and search modes. *ACM Trans. Inf. Syst.*, 26(1), #nov# 2007. ISSN 1046-8188. doi: 10.1145/1292591.1292595. URL `http://doi.acm.org/10.1145/1292591.1292595`.

[34] Mi Luo, Fei Chen, Pengxiang Cheng, Zhenhua Dong, Xiuqiang He, Jiashi Feng, and Zhenguo Li. Metaselector: Meta-learning for recommendation with user-level adaptive model selection. *arXiv preprint arXiv:2001.10378*, 2020.

[35] Dominika Tkaczyk, Paraic Sheridan, and Joeran Beel. Parsrec: A meta-learning recommender system for bibliographic reference parsing tools. In *Proceedings of the 12th ACM Conference on Recommender Systems (RecSys)*, 2018.

[36] Diana Gudu, Marcus Hardt, and Achim Streit. Combinatorial auction algorithm selection for cloud resource allocation using machine learning. In *European Conference on Parallel Processing*, pages 378–391. Springer, 2018.

[37] Laung-Terng Wang, Yao-Wen Chang, and Kwang-Ting Tim Cheng. *Electronic design automation: synthesis, verification, and test*. Morgan Kaufmann, 2009.

[38] Vivek Jain, Alex Tyrrell, Hud Wahab, Lars Kotthoff, and Patrick Johnson. In-situ raman investigation of laser-induced graphene using machine learning. *Bulletin of the American Physical Society*, 2020.

[39] Lionel Lobjois and Michel Lemaitre. Branch and Bound Algorithm Selection by Performance Prediction. page 6, 1998.

[40] Hilan Bensusan and Christophe Giraud-Carrier. Discovering Task Neighbourhoods through Landmark Learning Performances. In Djamel A. Zighed, Jan Komorowski, and Jan Żytkow, editors, *Principles of Data Mining and Knowledge Discovery*, Lecture Notes in Computer Science, pages 325–330. Springer, 2000. ISBN 978-3-540-45372-7. doi: 10.1007/3-540-45372-5_32.

[41] Javier Larrosa and Pedro Meseguer. Exploiting the use of dac in max-csp. In *International Conference on Principles and Practice of Constraint Programming*, pages 308–322. Springer, 1996.

[42] L. Xu, F. Hutter, H. H. Hoos, and K. Leyton-Brown. SATzilla: Portfolio-based Algorithm Selection for SAT. 32:565–606, 2008. ISSN 1076-9757. doi: 10.1613/jair.2490. URL `https://jair.org/index.php/jair/article/view/10556`.

[43] Damir Pulatov and Lars Kotthof. Utilizing software features for algorithm selection. In *COSEAL Workshop, co-located with the 15th ACM/SIGEVO Workshop on Foundations of Genetic Algorithms*, 2019b.

[44] Bernd Bischl, Pascal Kerschke, Lars Kotthoff, Marius Lindauer, Yuri Malitsky, Alexandre Fréchette, Holger H. Hoos, Frank Hutter, Kevin Leyton-Brown, Kevin Tierney, and Joaquin Vanschoren. ASlib: A Benchmark Library for Algorithm Selection. *Artificial Intelligence Journal (AIJ)*, (237):41–58, 2016.

[45] Damir Pulatov and Lars Kotthoff. Modelling Algorithmic Performance. 2019a.

[46] Michael Ekstrand and John Riedl. When recommenders fail: predicting recommender failure for algorithm selection and combination. In *Proceedings of the sixth ACM conference on Recommender systems*, pages 233–236. ACM, 2012.

[47] Jungtaek Kim, Saehoon Kim, and Seungjin Choi. Learning to warm-start bayesian hyperparameter optimization. In *NIPS 2017 Workshop on Bayesian Optimization*, 2017.

[48] Ljupco Todorovski, Hendrik Blockeel, and Saso Dzeroski. Ranking with predictive clustering trees. In *European Conference on Machine Learning*, pages 444–455. Springer, 2002.

[49] Andrew Collins, Dominika Tkaczyk, and Joeran Beel. One-at-a-time: A meta-learning recommender-system for recommendation-algorithm selection on micro level. *arXiv preprint arXiv:1805.12118*, 2018.

[50] F Maxwell Harper and Joseph A Konstan. The movielens datasets: History and context. *ACM Transactions on Interactive Intelligent Systems (TiiS)*, 5(4):19, 2016.

[51] DP Gaikwad and Ravindra C Thool. Intrusion detection system using bagging ensemble method of machine learning. In *2015 International Conference on Computing Communication Control and Automation*, pages 291–295. IEEE, 2015.

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

[53] Y. Taigman, M. Yang, M. Ranzato, and L. Wolf. Deepface: Closing the gap to human-level performance in face verification. In *2014 IEEE Conference on Computer Vision and Pattern Recognition*, pages 1701–1708. ISBN 1063-6919. doi: 10.1109/CVPR.2014.220.

[54] Nick Babich. Putting personas to work in ux design: What they are and why they're important. *Adobe Blog. https://theblog.adobe.com/putting-personas-to-work-in-ux-design-what-they-are-and-why-theyre-important/*, 2017.

[55] Wendy Kan. Lending club loan data, 2019. URL `https://www.kaggle.com/wendykan/lending-club-loan-data`.

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

[57] Martín Abadi, Ashish Agarwal, Paul Barham, Eugene Brevdo, Zhifeng Chen, Craig Citro, Greg S. Corrado, Andy Davis, Jeffrey Dean, Matthieu Devin, Sanjay Ghemawat, Ian Goodfellow, Andrew Harp, Geoffrey Irving, Michael Isard, Yangqing Jia, Rafal Jozefowicz, Lukasz Kaiser, Manjunath Kudlur, Josh Levenberg, Dan Mané, Rajat Monga, Sherry Moore, Derek Murray, Chris Olah, Mike Schuster, Jonathon Shlens, Benoit Steiner, Ilya Sutskever, Kunal Talwar, Paul Tucker, Vincent Vanhoucke, Vijay Vasudevan, Fernanda Viégas, Oriol Vinyals, Pete Warden, Martin Wattenberg, Martin Wicke, Yuan Yu, and Xiaoqiang Zheng. TensorFlow: Large-scale machine learning on heterogeneous systems, 2015. URL `http://tensorflow.org/`. Software available from tensorflow.org.

[58] Liudmila Prokhorenkova, Gleb Gusev, Aleksandr Vorobev, Anna Veronika Dorogush, and Andrey Gulin. Catboost: unbiased boosting with categorical features. In S. Bengio, H. Wallach, H. Larochelle, K. Grauman, N. Cesa-Bianchi, and R. Garnett, editors, *Advances in Neural Information Processing Systems 31*, pages 6638–6648. Curran Associates, Inc., 2018. URL `http://papers.nips.cc/paper/7898-catboost-unbiased-boosting-with-categorical-features.pdf`.

[59] Yingying Wang, Yibin Li, Yong Song, and Xuewen Rong. The influence of the activation function in a convolution neural network model of facial expression recognition. *Applied Sciences*, 10(5):1897, 2020.

[60] Bernd Bischl, Giuseppe Casalicchio, Matthias Feurer, Frank Hutter, Michel Lang, Rafael G. Mantovani, Jan N. van Rijn, and Joaquin Vanschoren. OpenML Benchmarking Suites. 2019. URL `http://arxiv.org/abs/1708.03731`.

A1 Appendix

A1.1 Codebase

This implementation of Siamese algorithm selection has been made public with an open source licence. The codebase is hosted on github and can be found using the link shared below.

https://github.com/BeelGroup/Algorithm-Performance-Personas