'Algorithm-Performance Personas' for Siamese Meta-Learning and Automated Algorithm Selection

Bryan Tyrrell TYRRELBR@TCD.IE
Trinity College Dublin, Ireland

Edward Bergman EBERGMAN@TCD.IE
D-REAL, ADAPT Centre, Trinity College Dublin, Ireland

Gareth J. F. Jones GARETH.JONES@DCU.IE
ADAPT Centre, School of Computing, Dublin City University, Ireland

Joeran Beel JOERAN.BEEL@UNI-SIEGEN.DE
University of Siegen, Department of Computer Science, Germany
& Trinity College Dublin, School of Computer Science and Statistics, ADAPT, Ireland

Abstract

We propose a new method of per-instance algorithm selection which can improve overall performance on machine learning tasks, surpassing traditional methods which often search and a select a single algorithm for use over an entire dataset. A prevailing challenge with algorithm selection however is to provide an accurate performance ranking of algorithms given some new instance or dataset with common methods involving meta-learning, classification and regression. Our proposed method, Siamese Algorithm Selection, is to train a Siamese Network to learn an embedding of instances, clustering according to both feature similarity and prior algorithm performances. These clusters which we dub Algorithm Performance Personas (APP) enable classic neighbourhood methods to then be used in generating an algorithm ranking for new instances. We find that our method works and successfully outperformed the best single algorithm by reducing MAE by 15%, on par with the performance of our baseline. We then investigate individual algorithm selection rates and conclude with directions for future work.

1. Introduction

The idea of meta-learning for automatic algorithm selection is by no means new and has shown to be effective in machine learning (Feurer et al., 2015; Mohr et al., 2018; Tu, 2018), information retrieval (Beel and Kotthoff, 2019; Ferro et al., 2018; He and Ounis, 2004; Mackenzie et al., 2018), recommender systems (Beel et al., 2019; Im and Hars, 2007; Luo et al., 2020), data mining (Tripathy and Panda, 2017) and material sciences (Jain et al., 2020). Often the problem is presented as a ranking one, given some new dataset, how do you predict the performance rankings of a pool of algorithms. Some early techniques worked to regress algorithm performances directly (Bensusan and Kalousis, 2001) with later work instead predicting relative rankings (Todorovski et al., 2002), incorporating dataset metafeatures (Matuszyk and Spiliopoulou, 2014) and learning dataset similarity with prior algorithm performances (Kim et al., 2017). Recently, there has been some work to instead focus on a per-instance approach which aims to make use of the fact that empirically, no single algorithm will perform best across all tasks (Kotthoff et al., 2015).

Our proposed method Siamese Algorithm Selection [1] contributes in three main ways, **(1)** A novel per-instance selection method incorporating Siamese Neural Networks to learn instance similarity trained on algorithm performances. **(2)** The concept of an 'Algorithm Performance Persona' (APP), which corresponds to clusters of similar performing instances. **(3)** A normalization of algorithm performances, accounting for cases of only small differences in their relative performances.

2. Related Work

One approach to per-instance selection we draw from is to incorporate algorithm characteristics. The work of Pulatov and Kotthof (2019a) investigates this in the domain of SAT solvers, training a regression model on algorithm source code and instance features to predict a solver for each instance. From this, they achieved a 95% reduction in time spent solving when compared to the single best algorithm, however, later experiments (Pulatov and Kotthoff, 2019b) found mixed and inconsistent results for other datasets, these inconsistencies motivate us to try a Siamese Architecture instead.

The prior performance of algorithms have also shown to be a useful characteristic as shown by Lobjois and Lemaitre (1998) and Bensusan and Giraud-Carrier (2000) who both model prior evaluations of algorithms to infer which algorithm is best for unseen instances. Xu et al. (2008) put this to great use in the 2008 SAT competition, training on historical algorithm performances from previous competitions to create a winning per-instance selection model. This is however quite domain specific, warranting further investigation.

The use of Siamese Neural Networks (SNN) was originally used to identify signature similarity (Bromley et al., 1994) but has since found its way with success in object tracking (Bertinetto et al., 2016), sentence similarity (Mueller and Thyagarajan, 2016) and speaker recognition (Chen and Salman, 2011). The work of Kim et al. (2017) trains a SNN to recognize dataset similarity by the performance of different acquisition functions, allowing them to warm-start algorithm configuration for new datasets. We extend this idea, consider the use of SNN for algorithm selection on a per-instance level rather than a per-dataset one.

3. 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 $\boldsymbol{A} = \{\mathcal{A}^{(1)}, \dots, \mathcal{A}^{(m)}\}$. We train our SNN 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, we embed it with our 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.

[1] We originally propose the idea in Beel et al. (2020) but this paper marks the first implementation and evaluation of this method.

3.1 Training

To train our SNN we first take each instance d_i in $\mathcal{D}_{train}$ and evaluate the performance of every algorithm in $\boldsymbol{A}$ to obtain a vector of performances $p_i = [\,p_i^{(1)}, \ldots, p_i^{(m)}\,]$ where $p_i^{(j)}$ is the performance of algorithm $\mathcal{A}^{(j)}$ on instance d_i. We then train our SNN with instance-performace pairs $(d_a, p_a), (d_b, p_b)$, with a contrastive loss function (Hadsell et al., 2006) to classify positive and negative pairs, positive pairs having their embeddings s_a, s_b brought closer while negative pairs have their embeddings drawn apart. This classification of pairs is determined by a margin of both their feature distance and performance vector distance.

3.2 Normalizing performances

Instead of directly using p_i when classifying points, we normalize them first. We combine what we refer to as relative intra-instance performance (RIIP) and max-possible relative error (MPRE). RIIP and MPRE are to account for relative algorithm performances and the possible scale of performances respectively, helping to separate performances with little difference between each algorithm. These metrics target error specifically and are reliant on a label for each instance but this can be dataset dependant. For an extended discussion on why the use of standard cosine distance and euclidean distance do not suffice, please refer to Appendix C.

The maximum possible error an algorithm can make for an instance d_i is factored in by MPRE and is defined as $p_i^{(j)}/\epsilon_i$ with the maximum possible error expressed as $\epsilon_i = \max(B_{max} - y_i, y_i - B_{min})$, the greatest possible difference a prediction can have from the label y_i bounded in $[B_{min}, B_{max}]$.

RIIP is calculated as $\min_k p_i^{(k)}/p_i^{(j)}$, scaling prediction errors to the range $[0, 1]$ with the minimum algorithm error at 1 and greater errors approaching 0. If we represent our final normalized performances as $\tilde{p}_i = [\,\tilde{p}_i^{(1)}, \ldots, \tilde{p}_i^{(m)}\,]$ we have that each performance is normalized according to

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

3.3 Learned APPs through Example

We train our SNN to take a pair of instances d_a, d_b to their embeddings s_a, s_b whose distance encode their APP similarity to one another. As training relies on pair comparisons (Appendix A), if we were to consider all n^2 possible pairs, training time would become cost prohibitive. We instead choose to identify 4 kinds of pairs for training, *easy-positive*, *hard-positive*, *easy-negative* and *hard-negative*. Using the distance between their normalized performance vectors $\|\tilde{p}_i - \tilde{p}_j\|$, we classify the pair as *positive* if they are close and *negative* if they are far. Depending on the distance distance of their features $\|d_i - d_j\|$, we similarly classify them as being *easy* if they are close and *hard* in far. These margins for far and close are considered hyperparemeters of the model and can also be used to reduce training time though a finer selection of pairs considered.

3.4 Per-Instance Algorithm Selection

We have trained our SNN to embed instances to their APPs such that they are in close proximity. If we reconsider our problem of per-instance algorithm selection, we have a straightforward method to evaluate algorithm selection for any unseen instance. For some unseen instance d_i in $\mathcal{D}_{train}$, Siamese Algorithm Selection will obtain its embedding s_i, consider its k-nearest neighbours in $\mathcal{S}$ and choose the algorithm which performed best over all neighbours, counting lower ranks in cases of a tie.

4. Methodology

For our implementation, we use the Lending Club Loan dataset (Kan, 2019). We chose this dataset due to its large number of columns and rows, 145 and 2.26 million respectively. As our prediction label we chose the interest rate. Our final processed dataset, consisted of 2.13 million rows and 74 columns, detailed preprocessing steps can be found in the provided code (Appendix E). We further select 50,000 instances to create $\mathcal{D}_{algo}$ for training the algorithms with the remaining data randomly split 90/10 where $\mathcal{D}_{train}$ consists of 1.872 million instances and $\mathcal{D}_{test}$ with 208 thousand instances.

We considered 8 regression algorithms which were optimized and trained using $\mathcal{D}_{algo}$. Each trained algorithm was then evaluated and their predictions recorded for every instance in $\mathcal{D}_{train}$ and $\mathcal{D}_{test}$. The performance of each algorithm was considered as the absolute error between the algorithms prediction and the label. We then normalize these performances according to our RIIP metric (eq. 1). the final performance ranks of all algorithms can be seen in table D of the appendix. Further details of algorithms used, the size of $\mathcal{D}_{algo}$ and their r-squared accuracies are listed in Appendix D.

Instances from similar performance personas where identified and paired as positive training pairs for our Siamese Neural Network. To achieve this, we implemented the idea discussed in section 3.3 of hard and easy positives by using distance thresholds.

The SNN architecture consisted of 4 layers with $40, 20, 20, 8$ respectively. All layers used a ReLU activation function with the exception of the final layer which is our output layer. The contrastive loss function, also known as pairwise ranking loss, was used to train the network's weights using the standard ADAM optimizer.

We then generated the embedding of each instance in $\mathcal{D}_{train}$, serving as points for our k-nearest neighbours algorithms. We also modify the amount of neighbours considered with $k \in \{3, 5, 16, 32, 128\}$. To choose the algorithm for instances of $\mathcal{D}_{test}$ we obtain their k-nearest neighbours and do a most often vote, choosing the algorithm that most often performed best across all of its neighbours. In the event of a tie, lower ranks were taken into consideration to select a suitable algorithm.

For comparison, our approach was measured against a Random Forest (RF) regression baseline which we implemented based on work from Pulatov and Kotthof (2019a). This baseline was trained to predict the error vectors directly, trained on the set $\mathcal{D}_{train}$ and evaluate on $\mathcal{D}_{test}$, the same as our SNN. We also consider an *oracle* baseline, a theoretically perfect algorithm with 100% selection accuracy on $\mathcal{D}_{test}$. The oracle produces the lowest possible MAE for a perfect algorithm selection given our fixed set of algorithms $\boldsymbol{A}$.

5. Results

We find that our Siamese Algorithm Selection method works and successfully outperforms any single algorithm. We compare and discuss its performance with the three single best performing algorithms with the remaining 5 omitted due to worse performance. We also compare this with the RF baseline and oracle baseline.

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%
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%
RF baseline	0.176	16.98%	50.0%
Oracle Selection	0.088	58.49%	100.0%

Table 1: 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.

We 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 our Siamese Algorithm Selection approach, reducing possible MAE by 17% with a higher selection accuracy of 50%.

Algorithm	**Oracle**	**RF Baseline**	**Siamese Algorithm Selection**
MLP Regressor	34.73 %	47.33 %	47.07 %
CatBoost	31.90 %	31.95 %	44.43 %
Random Forest	8.56 %	7.57 %	3.57 %
Lasso	7.04 %	4.27 %	1.85 %
SGD	5.91 %	4.75 %	1.36 %
RANSAC	5.64 %	1.16 %	1.07 %
Gradient Boosting	3.36 %	1.46 %	0.36 %
AdaBoost	2.85 %	1.51 %	0.25 %

Table 2: A comparison between the algorithm selections of Siamese Algorithm Selection with 128 neighbours, the Random Forest baseline and an oracle selection.

We can see that the SNN has learned to identify some APPs with its selection accuracy from table 1 of 8.5% above choosing the single best algorithm. While increasing neighbours we found that selection accuracy remains around 42% for $3, 5, 16, 32, 128$ neighbours but its MAE begins to improve. As part of our investigation as to why, we consider the percentage that each single algorithm was selected in Table 2.

From table 2, 91% 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.

6. Summary and Future Work

Our Siamese Algorithm Selection method, consisting of a Siamese Neural Network trained with 'Algorithm Performance Personas' (APP) and a novel normalization method successfully identified APPs of instances, to reduce the MAE by 15% over that of the single best algorithm. We have shown that distance in the performance space can be leveraged for training a neural network to transform instance features into embeddings, where distance of embeddings correlates with distance in performance space. This approach has shown to be useful and a promising future direction for per-instance algorithm selection.

While the results show promise for further optimization on our dataset, a comprehensive look over multiple datasets are required to assess the generality of our method. The OpenML dataset (Bischl et al., 2019) would be a promising direction for both standardized datasets but also to incorporate more information regarding algorithm performances.

We proposed a new metric to normalize performances, taking into account both the maximum possible error and the relative sizes of error. In practice this has worked for us but we recognize a potential divide by 0 in the case of an algorithm achieving no error. Improvements to the metric should be considered for future work.

We used a k-nearest neighbours with fixed neighbour sizes to determine APP clusters in the embedding space. APP clusters have no upper or lower bound on size and as such we would like to explore alternative approaches such as distance weighted k-nearest neighbours and variable sized clustering approaches.

Acknowledgments

This research was partly conducted with the financial support of the ADAPT SFI Research Centre at Trinity College Dublin. The ADAPT SFI Centre for Digital Media Technology is funded by Science Foundation Ireland through the SFI Research Centres Programme and is co-funded under the European Regional Development Fund (ERDF) through Grant #13/RC/2106. This work was also conducted with the financial support of the Science Foundation Ireland Centre for Research Training in Digitally-Enhanced Reality (D-REAL) under Grant No. 18/CRT/6224. We are further grateful for the support from and discussions with Andrew Collins and Shahad Nagoor.

References

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.

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.

Joeran Beel, Bryan Tyrell, Edward Bergman, Andrew Collins, and Shahad Nagoor. Siamese meta-learning and algorithm selection with 'algorithm-performance personas' [proposal]. *arXiv:2006.12328 [cs.LG]*, 2020.

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.

Hilan Bensusan and Alexandros Kalousis. Estimating the Predictive Accuracy of a Classifier. In Luc De Raedt and Peter Flach, editors, *Machine Learning: ECML 2001*, Lecture Notes in Computer Science, pages 25–36. Springer, 2001. ISBN 978-3-540-44795-5. doi: 10.1007/3-540-44795-4_3.

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`.

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`.

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.

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-network.pdf`.

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.

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.

R. Hadsell, S. Chopra, and Y. LeCun. Dimensionality Reduction by Learning an Invariant Mapping. In *2006 IEEE Computer Society Conference on Computer Vision and Pattern Recognition - Volume 2 (CVPR'06)*, volume 2, pages 1735–1742. IEEE, 2006. ISBN 978-0-7695-2597-6. doi: 10.1109/CVPR.2006.100. URL `http://ieeexplore.ieee.org/document/1640964/`.

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.

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`.

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.

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

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

Lars Kotthoff, Pascal Kerschke, Holger Hoos, and Heike Trautmann. Improving the State of the Art in Inexact TSP Solving Using Per-Instance Algorithm Selection. In Clarisse Dhaenens, Laetitia Jourdan, and Marie-Eléonore Marmion, editors, *Learning and Intelligent Optimization*, Lecture Notes in Computer Science, pages 202–217. Springer International Publishing, 2015. ISBN 978-3-319-19084-6. doi: 10.1007/978-3-319-19084-6_18.

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

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.

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.

Pawel Matuszyk and Myra Spiliopoulou. Predicting the Performance of Collaborative Filtering Algorithms. In *Proceedings of the 4th International Conference on Web Intelligence, Mining and Semantics (WIMS14)*, WIMS '14, pages 1–6. Association for Computing Machinery, 2014. ISBN 978-1-4503-2538-7. doi: 10.1145/2611040.2611054. URL `https://doi.org/10.1145/2611040.2611054`.

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`.

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`.

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*, 2019a.

Damir Pulatov and Lars Kotthoff. Modelling Algorithmic Performance. 2019b.

Ljupco Todorovski, Hendrik Blockeel, and Saso Dzeroski. Ranking with Predictive Clustering Trees. In Tapio Elomaa, Heikki Mannila, and Hannu Toivonen, editors, *Machine Learning: ECML 2002*, Lecture Notes in Computer Science, pages 444–455. Springer, 2002. ISBN 978-3-540-36755-0. doi: 10.1007/3-540-36755-1_37.

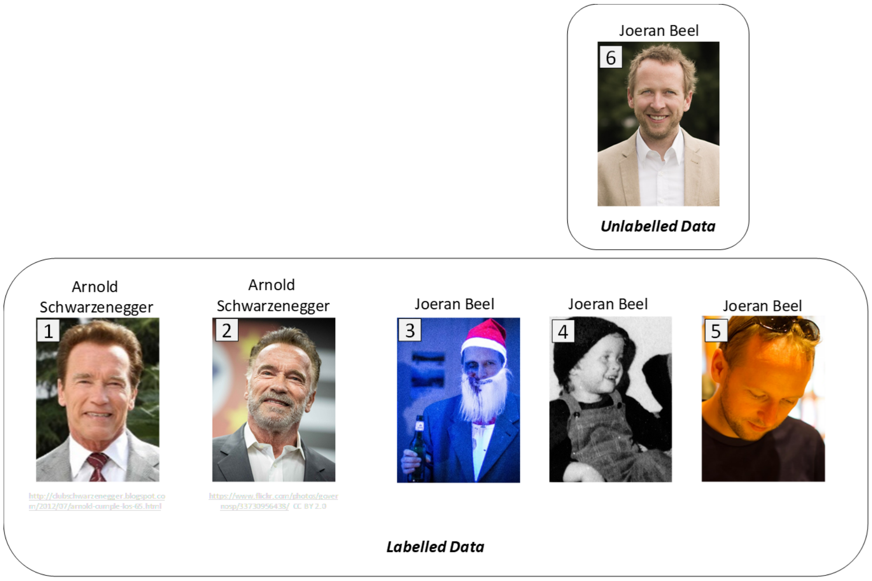
Murchhana Tripathy and Anita Panda. A study of algorithm selection in data mining using meta-learning. *Journal of Engineering Science & Technology Review*, 10(2), 2017.

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

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`.

Appendix A. Siamese Neural Networks for Few-Shot Image Classification

Siamese Neural Network architectures are successful in few-shot learning, particularly for metric learning for image classification. The goal of few-shot image classification is to classify images for which only few labelled samples exist – too few to train a model in the traditional way. Figure 1 illustrates this concept. The labelled dataset has images of the two persons 'Arnold Schwarzenegger' (photos 1 and 2) and 'Joeran Beel' (photos 3, 4, and 5), and there could be hundreds of classes more, each with only a few labelled images. A traditional (deep) machine learning algorithm (e.g. a CNN) would have difficulties in learning the classes with just few samples per class. In the example, the unlabeled image 6 likely would be misclassified as its features (pixels, color space, etc.) are more similar with photos of the class 'Arnold Schwarzenegger' then the correct class 'Joeran Beel'.



http://dubschwarzenegger.blogspot.com/2012/07/arnold-cumple-los-65.html

https://www.flickr.com/photos/governosp/33730956438/ CC BY 2.0

Figure 1: Illustration of a few-shot image classification dataset with only few images for each class.

To address this challenge, Siamese Neural Networks learn to calculate the similarity between any two data points, instead of learning to predict a class directly. This learned similarity function can be applied to any two datapoints including one unlabeled and one labelled data-point. If the distance between the unlabeled data point and the know data point is smaller than the margin α, the unabelled data point is assumed to be of the same class as the data point within close distance.

In the example, the Network would learn that images 1 and 2 are similar/identical and that images 3, 4 and 5 are similar – even if their actual features (color etc.) differ. Such a similarity function would more likely be able to predict that the unlabeled image 6 is similar to image 3, 4, and 5 and consequently assign the correct class to image 6. When trained on many classes, each having a few samples, Siamese Neural Networks achieve high accuracies.

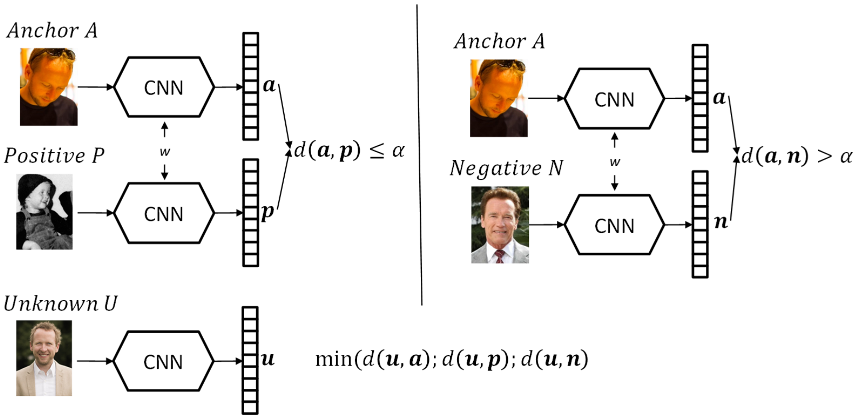

Anchor A
CNN
a
w
Positive P
CNN
p
$d(\boldsymbol{a},\boldsymbol{p}) \leq \alpha$
Anchor A
CNN
a
w
Negative N
CNN
n
$d(\boldsymbol{a},\boldsymbol{n}) > \alpha$
Unknown U
CNN
u
$\min(d(\boldsymbol{u},\boldsymbol{a}); d(\boldsymbol{u},\boldsymbol{p}); d(\boldsymbol{u},\boldsymbol{n}))$

Figure 2: Siamese Neural Network Illustration for Image Classification

The training of the Siamese Neural Network is conducted as follows. One network received an 'anchor' image and a 'positive' training sample as input. The other network receives the same anchor image and a negative sample as input. The two networks are then trained to output embeddings that are close to each other ($< \alpha$) in the embedding space for the positive example, and far from each other ($> \alpha$) for the negative example. For new unlabelled instances, the network transforms the image features into an embedding, and predicts the class based on the class of those image(s) being closest to the input image in the embedding space.

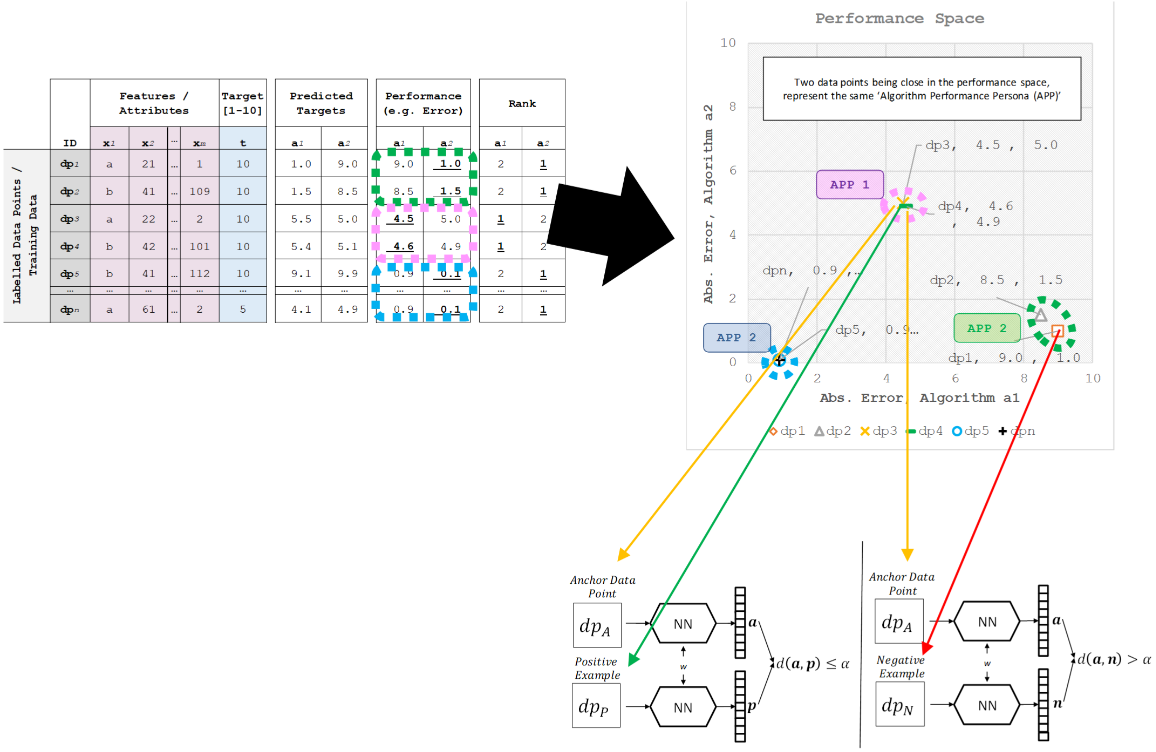
A.1 Illustration of the Architecture and Example

The overall idea of our approach is illustrated in 3. Given is a dataset $\mathcal{D}$ with n data points dp_i (figure 3, Top-Left). Each data point dp_i has m features and a target t (in our case a discrete number, e.g. the rating of a movie). Given is further the predicted targets of two algorithms a_1 and a_2, their corresponding absolute prediction errors (—target-predicted target—) and the rank, i.e. which of the two algorithms performed best on the given data point.

In the given example, data points dp_1 and dp_2 are not similar in terms of features (red), but the algorithms a_1 and a_2 perform alike on them (green). By "perform alike" we mean that the same algorithm performs best (a_2), and prediction errors of the two algorithms are similar (e.g. a_1 and a_2 have errors of 9 (a_1) and 1 (a_2) on dp1 and of 8.5 (a_1) and 1.5 (a_2) on dp_2). In contrast, dp_4 and dp_5 have similar features, but the algorithms perform different on them.

The overall goal is to make a network learn that e.d. dp_1 and dp_2 are similar regardless of their features. Or, in other words, given that, based on the performance dp_1 and dp_2 are similar by definition, the network shall learn to transform the not-similar features into embedding vectors that actually are similar.

All instances in the dataset are plotted in the performance space (3, Top-Right). In this example, the performance space has only two dimensions and uses the prediction error as metric. Instances that are close to each other in the performance space represent the same Algorithm-Performance-Persona. For instance, dp_3 and dp_4 are very close in the performance space and represent APP 1. Both these datapoints are then taken as inputs dp_A and dp_P for the Siamese Neural Network (3, Bottom-Left). Another datapoint (e.g. dp_1) is taken as negative input dp_N to the network. The network then learns to transform dp_A, dp_P, dp_N into the embeddings a,p,n so that $d(a,p) < \alpha$ and $d(a,n) > \alpha$.



		Features / Attributes				Target [1-10]	Predicted Targets		Performance (e.g. Error)		Rank	
	ID	x_1	x_2	…	x_m	t	a_1	a_2	a_1	a_2	a_1	a_2
Labelled Data Points / Training Data	dp_1	a	21	…	1	10	1.0	9.0	9.0	**1.0**	2	**1**
	dp_2	b	41	…	109	10	1.5	8.5	8.5	**1.5**	2	**1**
	dp_3	a	22	…	2	10	5.5	5.0	**4.5**	5.0	**1**	2
	dp_4	b	42	…	101	10	5.4	5.1	**4.6**	4.9	**1**	2
	dp_5	b	41	…	112	10	9.1	9.9	0.9	**0.1**	2	**1**
	…	…	…	…	…	…	…	…	…	…	…	…
	dp_n	a	61	…	2	5	4.1	4.9	0.9	**0.1**	2	**1**

Figure 3: Illustration of Algorithm Performance Persona

Appendix B. APP vs Cluster

On first glance, an Algorithm Performance Persona may seem similar or identical to a cluster. However, as we will show later, clustering is not a suitable technique to identify Algorithm Performance Personas. Also, we believe that a novel term is needed to express our idea as the term 'cluster' is associated to the machine learning clustering techniques, and we are optimistic that in the future many different concepts may be proposed to identify APPs. The approach we will propose is just one of potentially many.

Clustering may seem as obvious choice to identify Algorithm Performance Personas, and hence training pairs. The data points in the performance space could be clustered, and data points of the same cluster could be used as positive training pairs (figure 4). Data points from other clusters could be used as negative samples. We would expect though that clustering will not perform optimally for two reasons (5). First, there likely would be clusters with data points for which different algorithms perform best. This likely would not lead to precise prediction later about which algorithms will perform best for a new data

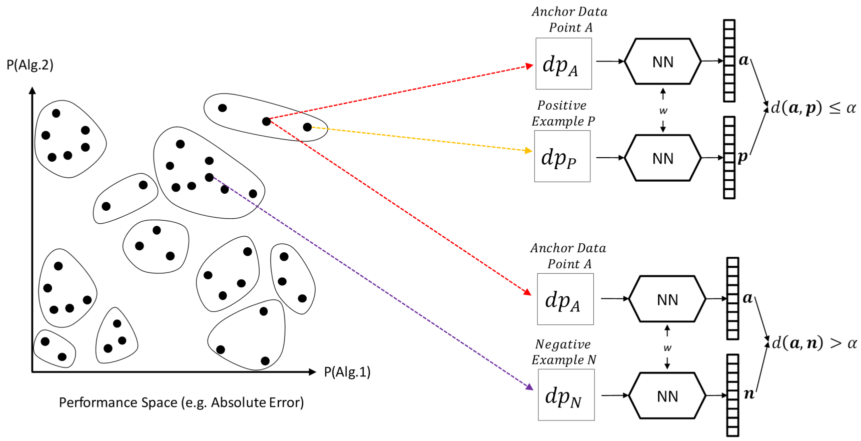
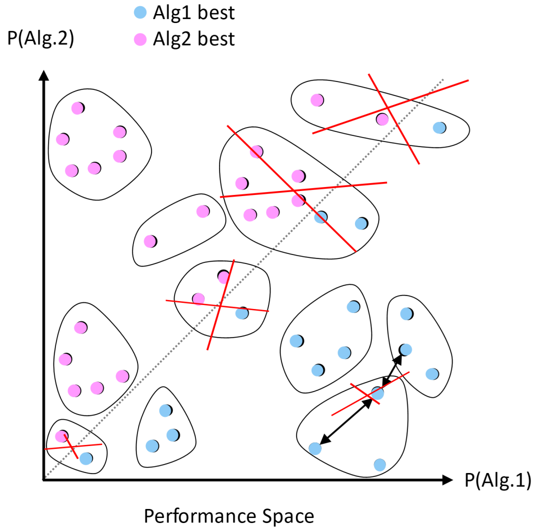


Figure 4: Clustering data points in the performance space to identify positive and negative training samples

point. Second, the distance between two data points in the same cluster might be larger than the distance between data points in different clusters. Treating data points in different clusters but close distance as different Personas is counter-intuitive.



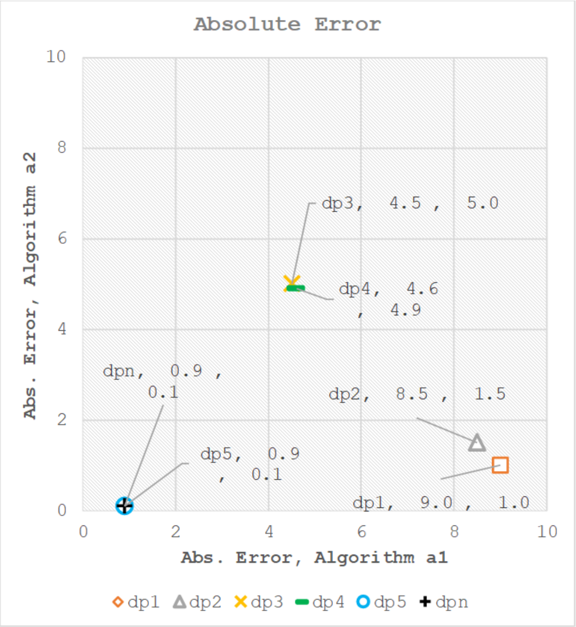
centering

Figure 5: Potential problems when using clustering

In image classification, the selection of positive and negative training pairs is relatively straight forward. Two photos either are of the same class, i.e. they show the same person (positive sample), or they are not (negative sample). For algorithm selection, the sample selection is not that clearly given, i.e. an equivalent is needed to a 'person' in image classification

Appendix C. MAE vs our Novel Metric

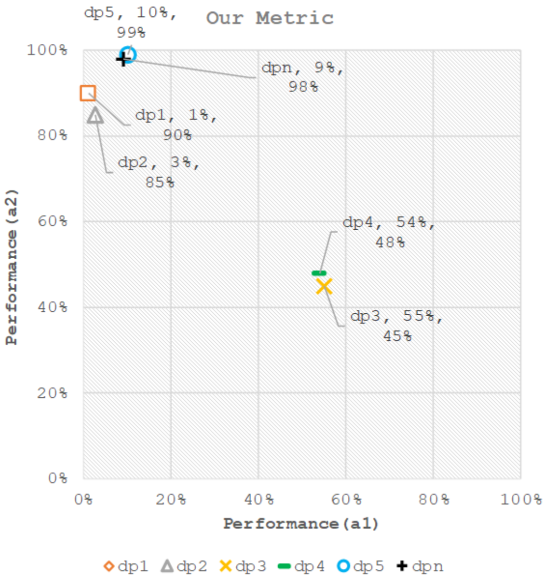
There are two rather obvious metrics for the performance space: the ranking of algorithms from 1 to n and the actual performance of an algorithm (e.g. MAE, Accuracy, Precision, ...). However, we consider both choices as suboptimal and will demonstrate in the following section why that is. We then present a novel metric. Since we define the performance space as a vector space, Cosine and Euclidian distance are two rather obvious choices to measure similarity.



		Features / Attributes				Target [1-10]	Predicted Targets		Performance (e.g. Error)		Rank			Which data points perform similar to dp*1*? (Intuition)	Similarity based on Abs. Error	
	ID	x*1*	x*2*	…	x*m*	t	a*1*	a*2*	a*1*	a*2*	a*1*	a*2*			Vector Space (Eucl.Dist.)	Vector Space (Cosine Dist.)
Labelled Data Points / Training Data	dp*1*	b	41	…	109	10	1.0	9.0	9.0	**1.0**	2	**1**				
	dp*2*	a	21	…	1	10	1.5	8.5	8.5	**1.5**	2	**1**	$\mathcal{S}(dp_1, dp_2)$	Very similar	Very Similar	Very Similar
	dp*3*	a	22	…	2	10	5.5	5.0	**4.5**	5.0	**1**	2	$\mathcal{S}(dp_1, dp_3)$	Not similar	Less than $\mathcal{S}(dp_1, dp_5)$	Quite Dissimilar
	dp*4*	b	42	…	101	10	5.4	5.1	**4.6**	4.9	**1**	2	$\mathcal{S}(dp_1, dp_4)$	Not similar	Less than $\mathcal{S}(dp_1, dp_5)$	Quite Dissimilar
	dp*5*	b	41	…	112	10	9.1	9.9	0.9	**0.1**	2	**1**	$\mathcal{S}(dp_1, dp_5)$	Somewhat Similar	Quite Dissimilar	Identical
	…	…	…	…	…	…	…	…	…	…	…	…		…	…	…
	dp*n*	a	61	…	2	5	4.1	4.9	0.9	**0.1**	2	**1**	$\mathcal{S}(dp_1, dp_n)$	Somewhat Similar	Quite Dissimilar	Identical

Figure 6: Performance Space based on Absolute Error

The relative intra-instance performance (RIIP) measures how well an algorithm performs compared to the other algorithms. For every data point, the best performing algorithms achieves a RIIP of 1 (or 100%). The other algorithms receive values between 0 and 1, indicating how close their original performance is to the best performing one. For instance, a_1 has an error of 8.5 on dp_2 while a_2 has an error of 1.5. This means, a_1 only is 18% as good as algorithm 2, and hence RIIP is 0.18 or 18%. RIIP is inspired by other similar metrics for pairwise comparisons and landmarkers, which are relatively commonly used for automated algorithm selection. However, to the best of our knowledge the metrics were used in different contexts, and typically for binary comparisons. Also, RIIP alone is not sufficient for our purpose.



		Features / Attributes				Target [1-10]	Predicted Targets		Performance (e.g. Error)		Rank			Which data points perform similar to dp1? (Intuition)	Similarity based on Abs. Error	
	ID	x_1	x_2	…	x_m	t	a_1	a_2	a_1	a_2	a_1	a_2			Vector Space (Eucl.Dist.)	Vector Space (Cosine Dist.)
Labelled Data Points / Training Data	dp_1	b	41	…	109	10	1.0	9.0	9.0	**1.0**	2	**1**				
	dp_2	a	21	…	1	10	1.5	8.5	8.5	**1.5**	2	**1**	$\mathcal{S}(dp_1, dp_2)$	Very similar	Very Similar	Very Similar
	dp_3	a	22	…	2	10	5.5	5.0	**4.5**	5.0	**1**	2	$\mathcal{S}(dp_1, dp_3)$	Not similar	Less than $\mathcal{S}(dp_1, dp_5)$	Quite Dissimilar
	dp_4	b	42	…	101	10	5.4	5.1	**4.6**	4.9	**1**	2	$\mathcal{S}(dp_1, dp_4)$	Not similar	Less than $\mathcal{S}(dp_1, dp_5)$	Quite Dissimilar
	dp_5	b	41	…	112	10	9.1	9.9	0.9	**0.1**	2	**1**	$\mathcal{S}(dp_1, dp_5)$	Somewhat Similar	Quite Dissimilar	Identical
	…	…	…	…	…	…	…	…	…	…	…	…		…	…	…
	dp_n	a	61	…	2	5	4.1	4.9	0.9	**0.1**	2	**1**	$\mathcal{S}(dp_1, dp_n)$	Somewhat Similar	Quite Dissimilar	Identical

Figure 7: Performance Space based on our novel metric

The Max-Possible Relative-Error (MPRE) takes into consideration that for different data points the scale may vary. This is true for many regression problems, particularly in the field of recommender systems. For instance, for movie recommendations, ratings are often made on a scale between 1 and 10 (or other bounded scales). If the actual target rating is 4, then the maximum possible error an algorithm could make is 6 (if the algorithm predicts 10). However, if the actual rating is 9, then the maximum possible error an algorithm can make is 8 (if 1 is predicted). In our example, the absolute errors for dp_5 and dp_n are identical (0.9 and 0.1 respectively). However, the target for dp_5 is 10, and for dp_n is 5. Hence, we would argue that the algorithms performed better on dp_5 than on dp_n. For dp_5, the algorithms are only 9% and 1% off the actual target, while for dp_n the algorithms are off

18% and 2% respectively. MPRE takes this into consideration by expressing an algorithm's performance based on the error relative to the maximum possible error. Of course, this metric is only relevant for scenarios where different data points have different maximum possible errors.

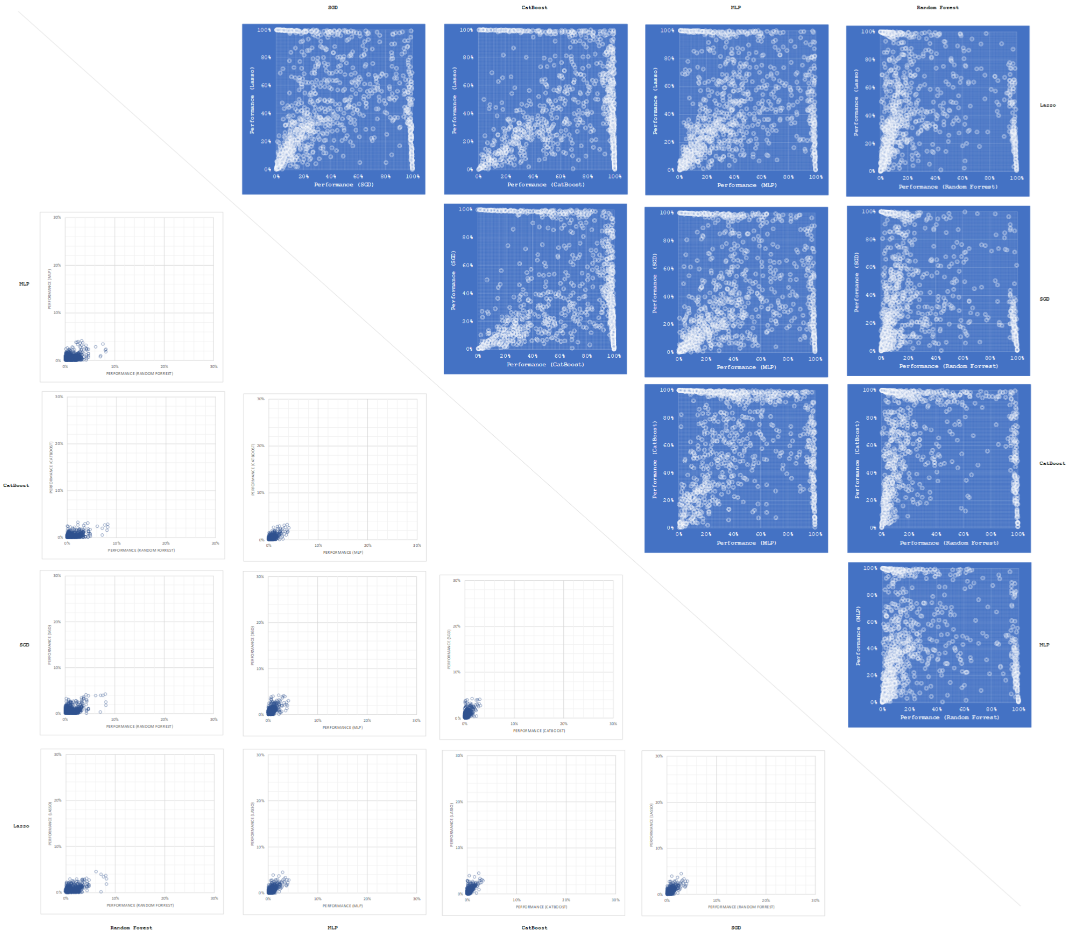


Figure 8: 1,000 Samples Plotted to the Performance Spaces (2 Dimensions Each); Mean Absolute Error

Appendix D. Algorithm Training

A varied suite of algorithms was selected to try 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 final performance of each algorithm was measured using R squared accuracy shown in table 3 below.

Algorithm	**R-Squared Accuracy**
Lasso	0.96 %
SGD	0.95 %
MLP Regressor	0.99 %
CatBoost	0.99 %
Random Forest	0.97 %
AdaBoost	0.86 %
RANSAC	0.96 %
Gradient Boosting	0.84 %

Table 3: The R-Squared accuracy of each algorithm on the 50,000 instances they were trained on.

It is clear that the *MLP Regressor* and *Catboost* perform considerably better compared to other regression algorithms in the suite. 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. Once each algorithm was trained, it was asked to predict a label for the remaining 1.8 million instances. The absolute error from each prediction was calculated and the performance rank of each algorithm measured against its peers is shown in Table 4.

Rank	Lasso %	SGD %	CatBoost %	MLP %	Random Forest %	Ada Boost %	RSNAC %	Gradient Boosting %
-1st	5.91	7.04	31.90	34.73	8.56	2.85	5.64	3.36
2nd	7.38	7.45	29.32	28.68	10.20	4.27	7.42	5.28
3rd	10.85	9.84	16.36	16.14	19.05	6.41	12.16	9.19
4th	18.66	12.37	9.60	9.41	19.04	6.71	17.54	6.67
5th	25.12	12.22	6.43	5.53	17.25	6.35	22.43	4.67
6th	21.00	21.59	4.14	3.48	15.28	8.69	20.67	5.15
7th	9.14	22.83	2.04	1.71	7.64	29.07	10.91	16.65
8th	1.93	6.67	0.21	0.32	2.98	35.64	3.24	49.01

Table 4: Performance of each algorithm at each rank, with **1st** being the optimal rank.

Appendix E. Codebase

The codebase along with results for the most recent run presented in the paper is available at the

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