‘Algorithm-Performance Personas’ for Siamese Meta-Learning and Automated Algorithm Selection

Bryan Tyrrell
Trinity College Dublin, Ireland

Edward Bergman
D-REAL, ADAPT Centre, Trinity College Dublin, Ireland

Gareth J. F. Jones
ADAPT Centre, School of Computing, Dublin City University, Ireland

Joeran Beel
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 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 \(S\) and a nearest-neighbours algorithm to assign to each unseen embedded instance \(s_i \in S\) an algorithm from a pool of algorithms \(A = \{A^{(1)}, \ldots, A^{(m)}\}\). 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 \(S\). For an unseen instance \(d_{\text{new}}\), to select an algorithm to run on said instance, we embed it with our SNN to get its embedding \(s_{\text{new}}\) and select the algorithm \(A^{(j)}\) 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 $D_{\text{train}}$ and evaluate the performance of every algorithm in $A$ to obtain a vector of performances $p_i = [p_i^{(1)}, \ldots, p_i^{(m)}]$ where $p_i^{(j)}$ is the performance of algorithm $A^{(j)}$ on instance $d_i$. We then train our SNN with instance-performance 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 dependent. 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 $\epsilon_i = \max(B_{\text{max}} - y_i, y_i - B_{\text{min}})$, the greatest possible difference a prediction can have from the label $y_i$ bounded in $[B_{\text{min}}, B_{\text{max}}]$.

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

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

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 $D_{\text{train}}$, Siamese Algorithm Selection will obtain its embedding $s_i$, consider its $k$-nearest neighbours in $S$ 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 $D_{\text{algo}}$ for training the algorithms with the remaining data randomly split 90/10 where $D_{\text{train}}$ consists of 1.872 million instances and $D_{\text{test}}$ with 208 thousand instances.

We considered 8 regression algorithms which were optimized and trained using $D_{\text{algo}}$. Each trained algorithm was then evaluated and their predictions recorded for every instance in $D_{\text{train}}$ and $D_{\text{test}}$. 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 $D_{\text{algo}}$ 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 $D_{\text{train}}$, 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 $D_{\text{test}}$ 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 $D_{\text{train}}$ and evaluate on $D_{\text{test}}$, the same as our SNN. We also consider an oracle baseline, a theoretically perfect algorithm with 100% selection accuracy on $D_{\text{test}}$. The oracle produces the lowest possible MAE for a perfect algorithm selection given our fixed set of algorithms $A$. 

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.

![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.](image)

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>MAE</th>
<th>Reduction in MAE</th>
<th>Selection Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>RF Regressor</td>
<td>0.549</td>
<td>−158.96%</td>
<td>8.5%</td>
</tr>
<tr>
<td>CatBoost</td>
<td>0.236</td>
<td>−11.32%</td>
<td>31.9%</td>
</tr>
<tr>
<td>MLP Regressor</td>
<td>0.212</td>
<td>0%</td>
<td>34.7%</td>
</tr>
<tr>
<td>SNN with 3-NN</td>
<td>0.211</td>
<td>0.47%</td>
<td>41.8%</td>
</tr>
<tr>
<td>SNN with 5-NN</td>
<td>0.201</td>
<td>5.18%</td>
<td>42.5%</td>
</tr>
<tr>
<td>SNN with 16-NN</td>
<td>0.194</td>
<td>8.49%</td>
<td>42.8%</td>
</tr>
<tr>
<td>SNN with 32-NN</td>
<td>0.186</td>
<td>12.26%</td>
<td>43.0%</td>
</tr>
<tr>
<td>SNN with 128-NN</td>
<td>0.180</td>
<td>15.09%</td>
<td>43.2%</td>
</tr>
<tr>
<td>RF baseline</td>
<td>0.176</td>
<td>16.98%</td>
<td>50.0%</td>
</tr>
<tr>
<td>Oracle Selection</td>
<td>0.088</td>
<td>58.49%</td>
<td>100.0%</td>
</tr>
</tbody>
</table>

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

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

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%.
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


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’.

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 $\alpha$, 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.
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 $D$ 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 ($-\text{target-predicted target}$—) 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 $dp_1$ 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 $d(a, n) > \alpha$.

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.

Figure 5: Potential problems when using clustering

In image classification, the selection of positive and negative training pairs is relatively straightforward. 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.

![Figure 6: Performance Space based on Absolute Error](image)

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 achieve 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.

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

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

<table>
<thead>
<tr>
<th>Algorithm</th>
<th>R-Squared Accuracy</th>
</tr>
</thead>
<tbody>
<tr>
<td>Lasso</td>
<td>0.96 %</td>
</tr>
<tr>
<td>SGD</td>
<td>0.95 %</td>
</tr>
<tr>
<td>MLP Regressor</td>
<td>0.99 %</td>
</tr>
<tr>
<td>CatBoost</td>
<td>0.99 %</td>
</tr>
<tr>
<td>Random Forest</td>
<td>0.97 %</td>
</tr>
<tr>
<td>AdaBoost</td>
<td>0.86 %</td>
</tr>
<tr>
<td>RANSAC</td>
<td>0.96 %</td>
</tr>
<tr>
<td>Gradient Boosting</td>
<td>0.84 %</td>
</tr>
</tbody>
</table>

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.

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

<table>
<thead>
<tr>
<th>Rank</th>
<th>Lasso %</th>
<th>SGD %</th>
<th>CatBoost %</th>
<th>MLP %</th>
<th>Random Forest %</th>
<th>Ada Boost %</th>
<th>RANSAC %</th>
<th>Gradient Boosting %</th>
</tr>
</thead>
<tbody>
<tr>
<td>1st</td>
<td>5.91</td>
<td>7.04</td>
<td>31.90</td>
<td>34.73</td>
<td>8.56</td>
<td>2.85</td>
<td>5.64</td>
<td>3.36</td>
</tr>
<tr>
<td>2nd</td>
<td>7.38</td>
<td>7.45</td>
<td>29.32</td>
<td>28.68</td>
<td>10.20</td>
<td>4.27</td>
<td>7.42</td>
<td>5.28</td>
</tr>
<tr>
<td>5th</td>
<td>25.12</td>
<td>12.22</td>
<td>6.43</td>
<td>5.53</td>
<td>17.25</td>
<td>6.35</td>
<td>22.43</td>
<td>4.67</td>
</tr>
<tr>
<td>6th</td>
<td>21.00</td>
<td>21.59</td>
<td>4.14</td>
<td>3.48</td>
<td>15.28</td>
<td>8.69</td>
<td>20.67</td>
<td>5.15</td>
</tr>
<tr>
<td>7th</td>
<td>9.14</td>
<td>22.83</td>
<td>2.94</td>
<td>1.71</td>
<td>7.64</td>
<td>29.07</td>
<td>16.91</td>
<td>16.65</td>
</tr>
<tr>
<td>8th</td>
<td>1.93</td>
<td>6.67</td>
<td>0.21</td>
<td>0.32</td>
<td>2.98</td>
<td>35.64</td>
<td>3.24</td>
<td>49.01</td>
</tr>
</tbody>
</table>

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