LEGION: Visually compare modeling techniques for regression

Subhajit Das and Alex Endert

Abstract—People construct machine learning (ML) models for various use cases in varied domains such as in healthcare, finance, public-policy, etc. In doing so they aim to improve a models' performance by adopting various strategies, such as changing input data (data augmentation), tuning model hyperparameters, performing feature engineering that includes feature extraction, feature augmentation or feature transformation. However, how would users know which of these model construction strategies to adopt for their problem? Following any or all of these approaches allows the construction of a gigantic set of models, from which users may select model(s) suited to their data analytic task. This problem of model selection is non-trivial because in real-world use cases many of the best performing models (in relation to a specified metric) may appear to serve users' goal but often exhibits nuances and tradeoffs (e.g, may weight features differently, varying compute times to train, or may predict relevant data instances differently etc.). This paper aims to solve the problem of how to construct models and how to select a preferred modeling strategy by allowing users to compare the differences and similarities between multiple regression models, and then learn not only about the model but also about their data. This learning further empowers them to select model(s) that more precisely suit their analysis goals. We present LEGION, a visual analytic tool that helps users to compare and select regression models constructed either by tuning their hyperparameters or by feature engineering. We also present two use cases on real world datasets validating the utility and effectiveness of our tool.

Index Terms—Visual comparison, Visual Analytics, Regression Task, Model selection, Multi-model system, Feature engineering.

1 Introduction

Machine learning (ML) has changed how people solve problems in various domains. For example healthcare practitioners use ML to predict disease diagnostics [1], [2], financial analysts model and forecast future housing prices [3], etc. In such scenarios, people construct models that are expected to not only be highly accurate, but also should satisfy other relevant user-specified criteria. These may include models that are easier to explain or reason, models that can better characterize missing values in the training data, models that are faster to train and deploy in a production setting etc. To achieve these goals users can construct better models by augmenting data, tuning hyperparameters, or adopting feature engineering techniques. As such users should question: (1) What modeling strategies (e.g., hyperparameter tuning or feature engineering) should be used to construct models?, and (2) How should models be selected from a set of candidate models? To resolve the model construction problem we present a visual technique that aids in comparison of models that are constructed using feature engineering versus models that are constructed using hyperparameter tuning. We are motivated not only to help users select models but more importantly to help users decide on an appropriate model construction strategy for their domain-specific problem, an important distinction from previous systems.

The second question that this paper addresses pertains to model selection in visual analytics (VA), which often is a multi-objective optimization problem, where users may have a varying set of domain-specific goals that depends on the dataset and the problem at hand e.g., accuracy per class label on an imbalanced data [4]. In an interactive model selection process (e.g., RegressionExplorer [5], Xclusim [6], etc.) when users are presented with multiple models that solve the same task, some candidate models may serve a subset of their goals, but these may also have critical tradeoffs that needs to be presented to users as they select their preferred model. Recently, VA researchers have looked at visual interfaces that provide access to multiple machine learning models. For example, ClusterVision allows interactive construction of multiple clustering models [7], Boxer shows an approach to compare discrete-choice classifiers in relation to subset data items using multiple coordinated views [8]. Similarly, Hypermoval [9] and BEAMES [10] are systems that allow interactive modeling of multiple regression models. While these tools are effective and useful to help users find or compare preferred models, they fail to help users decide on what modeling strategy is suitable for their problem. Most of these systems, are implemented using one of the modeling strategies (i.e., either hyperparameter tuning, or feature engineering, etc.). For example, the tool Squares [11], shows model performance visualization for multi-class classifiers, but fails to clarify the distinction or commonality between models in relation to how they were constructed. Another tool Gamut allows construction and interpretation of a very specific ML model (GAM models for regression), but is not designed to answer what modeling strategy would solve the users' problem [12].

We seek to express incoherencies and similarities between models in relation to various modeling strategies. Different models show subtle nuances in relation to data instances and features present in the data, that exposes the tradeoffs between them, and further helps users to learn more about the model and their data. To that end, MLCube is a VA system [13] that is the closest solution to this problem (that we know of). MLCube compares multiple models in relation to aggregation statistics and evaluation metrics over data instance subsets. However, MLCube does not compare models in relation to the underlying modeling technique. Furthermore, past VA systems (e.g., BEAMES [10], FeatureExplorer [14]) expect

• *, Subhajit Das, and Alex Endert are with Georgia Institute of Technology. E-mail: das, endert@gatech.edu*

users to specify feature weights or select features when they construct models. This is effective when the user is either a domain expert or the data has such features that are more relevant than others. However, what if the user is not certain about which features to select, or if the data contains noise and does not reflect the users expectation. In such cases, we need systems that show various modeling possibilities by using various combination of features.

In this paper, we present LEGION, a VA tool that allows comparison of regression models across two modeling strategies: (1) Hyperparameter Tuning, and (2) Feature Engineering seeking to assist users in knowing which modeling strategy is suited to their problem. Thus it helps users to select an appropriate modeling strategy in addition to selecting preferred models. While the first type of models are constructed using hyperparameter tuning (with HP JS library [15]) of Tensorflow based linear regression models [16], the other type is implemented as a custom designed feature engineering module on a traditional boosted ridge regression model built using the Machine Learning JS library [17]. It visually represents models from these two categories by placing them in separate bins side by side in a *Stacked Horizontal Bar View* (refer Figure 1-A). While the color represents the model category (and a model performance metric), the size of the bars represent a user specified performance metric such as adjusted $R^2 - Score$, Root Mean Squared Error (RMSE) etc. Users can also see system recommended top 'k' models' prediction output per data instance, through the *Instance Unit View* (refer Figure 1-B) highlighting prediction error per data instance. Furthermore, users can compare high performing models from each category by exploring feature correlation, variance etc. (see Figure 4) and hyperparameter settings (see Figure 1-C, and Figure 3) that led to the construction of their preferred models. LEGION visualises this information through a series of stacked unit visualizations (see Figure 3) and small multiples of scatterplot charts. To continue their analysis process, users can specify new hyperparameter settings (i.e., hyperparameter names and value ranges to construct new models), and a set of features (based on the learning about the models and the data in the analysis process) to further train new models that better support their expectations.

Incremental construction of multiple ML models, and analysing them leading to the selection of a specific modeling strategy and a set of preferred model(s) is a very complex task that require technical expertise in both software skills and elementary data science. For instance, current multi-model systems [7], [18], require users to adapt to the steep learning curve of interacting with a complex user interface with a complex workflow, in addition to be able to comprehend model metrics, hyperparameters etc. While we are motivated to aid comparison of multiple candidate models to help users learn about them and the underlying data, we also seek to simplify the complex user experience of interacting with multiple models prevalent in current systems. To that end, we aid comparison of models in relation to prediction on relevant data instances or explaining a models' reasoning process through the visualization of top weighted features (by the model) and their correlation with the target label in the data. As such, if a user notices that a high performing model weights features that are more relevant to their domain, they may feel more confident about it than a similar black-boxed model. Likewise, if a user sees high performing models fail to correctly predict relevant data instances, in comparison to other relatively lower performing models that correctly predicted these data instances, they may understand that there is a tradeoff between the overall accuracy and the accuracy over a subset of data instances. Furthermore, if the user sees models from one category (e.g., hyperparameter tuning) are consistently performing better, they learn which model construction strategy is more suitable for their problem (and the data). In LEGION, if users are not satisfied with any of the presented models, they can also explore models that are constructed using an hybrid approach.

We also present two use cases with real world datasets of Cancer Mortality Rate prediction [19] and US Stock Price variation prediction [20]. Through these use cases we explain how LEGION helps users: (1) in deciding on modeling strategy that is appropriate for their problem, (2) to learn about features in the data that drive the construction of preferred models, and (3) to retrieve various hyperparameter settings of constructed models that specifically solved their analytic goals. Our contributions are:

- A visual analytic system LEGION that allows users to compare models constructed using either hyperparameter tuning or feature engineering approach.
- Two use cases on real world datasets that explain the utility of the tool and validate that LEGION is effective in teaching users about model tradeoffs and about the underlying data.

2 Related Work

2.1 Interactive modeling in VA

In visual analytics there are many systems that have looked at interactive construction of models [21], [22], [23], [24], [25], [26], [27]. These include models for classification task [28], [29], [30], interactive labeling [31], [32], dimensionality reduction [33], [34], clustering [7], [18], metric learning [35], etc. These systems are driven by the concept that users can visually adjust model parameters by direct interaction with graphical visual interface elements, that drive the underlying model adjustment/steering process to adhere to their expectations. Using this approach users can bypass the need to know specific details on model hyperparameters, model performance metrics and other mathematical constructs, the burden of which is off-loaded to the system. For example, the system ModelTracker visualizes model performance metrics in relation to data instances and their similarity in the feature space to help model developers debug models [36]. Using a similar interaction approach, in this paper we are specifically looking at regression tasks in ML. Regression is a process that aids prediction of a continuous value given a set of training examples (e.g., rows and columns in a table). For instance, regression allows predicting the price of a house (given a set of features such as size, rooms, location, etc.) [3]. In VA there are many interactive regression tools e.g., Piringer et al. demonstrated a multiple regression modeling system that uses comparison of multiple model outputs to help users select a preferred model [9]. BEAMES, another regression modeler allows users to interactively inspect, select, and steer multiple regression models [10]. While these tools are effective to solve a regression task, they do not allow comparison of multiple models with respect to how they were constructed, an aspect of modeling that we are specifically exploring in this paper.

2.2 AutoML and Hyperparameter Tuning

Model selection is a pivotal problem in making ML accessible to a wide range of people, use cases, and domains. This model selection problem can be framed as, given a set of learning algorithms A, a set of associated hyperparameters H, a dataset D (split into train and test set), a very specific ML task (e.g., regression) T, the goal is to select a model m (from a large set of model options M such

Fig. 1. LEGION - multi-regression modeller. A. Stacked Horizontal Bar View showing regression models constructed using two techniques in separate bins (blue and red bars). B. Instance Unit View. C. Parameter Comparison View. D. Model description in text. E. Radial View showing top 'k' models' RMSE on train and test set (color coded by modeling strategy used). F. Buttons to toggle new models and control panel.

that it maximizes or minimizes a pre-defined performance metric *P*. As this task often requires significant technical expertise, recently researchers demonstrated applications that automate this process of model selection by the invention of AutoML platforms such as AutoWeka [37], [38], SigOpt [39], TPOT [40], HyperOpt [41], [42], or Auto-sklearn [43]. Other tools have also looked at visually tuning hyperparameters to help users decide on optimal models [44], [45]. These systems apply techniques such as Bayesian optimization [43], evolutionary algorithms [40], deep reinforcement learning [46], or other custom-defined evaluation metrics [47] to select a model for the given problem case. However, these tools fail to incorporate subjective preferences of end-users, that drives the motivation to include humans-in-the-loop of model construction and selection [48]. To bridge the gap between AutoML and the conventional process of manually selecting models, we are motivated to present a visual analysis tool that automatically drives model construction and recommendation of preferred models for selection (guided by users), based on comparative analysis of candidate models.

2.3 Multi-model systems in VA

VA literature has seen the surge of multi-model based VA systems that provide users access to many ML models. These span various ML problems e.g., Clustervision assisted users to find better performing cluster models [7], Starspire showed building text analytic models [49] using multiple models, Prospect facilitated adjusting data properties as users interact with multiple ML models [50]. As we are interested in regression task, we looked at similar multi-model interfaces for regression modeling such as Hypermoval [9], RegressionExplorer [5], BEAMES [10] etc. A similar system Gamut is a regression modeler using GAM models that allows interpreting model output in relation to data instances and features [12]. Many of these systems also allow model comparison such as MLCube [13] that provides a tabular list view to see characteristics of different models. Similarly StackGenVis assists users to decide between different model options by managing data instances, selecting relevant feature combinations, and finally selecting highly performant models [51]. Mühlbacher et al. showed a technique to present trade-off analysis, model validation, and comparison of many regression models [52]. As effective as these and many other [53], [54], [55], [56], [57], [58] multi-model tools are, only a subset of them provide direct model comparison. Even the tools that deploy model comparison, most of them compare models of a specific type or class e.g., boosting models or random forests etc. More importantly, none of them clarify the distinction between models in relation to how they are constructed. Thus to answer users' question: What modeling strategy should they choose?, we seek to build a visual analysis tool by showing direct comparison of regression models with respect to how they were constructed.

To summarize, from the literature review we understood that we need to design a visual interface that expands current multi-model tools and model comparison approaches by allowing model comparison with respect to their construction techniques. Our technique should explain incremental model incoherencies or similarities in relation to data instances and feature correlations.

3 System: UI and Technique

Here we describe our visual analytic prototype - LEGION, designed to interactively construct multiple regression models. We first describe a set of design guidelines and tasks that we derived based on extensive literature review, and our past efforts in the space of multi-model based VA systems.

3.1 Design Guidelines and Tasks

DG1 : Visually show model incoherencies based on their construction technique. There are many regression model

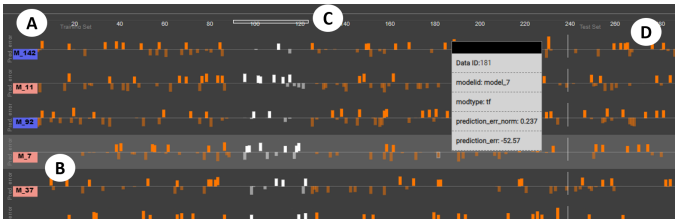

Fig. 2. Instance Unit View - A. Training instances shown as orange bars, encode prediction errors (negative errors are shown by upside down bars. B. Each row is a models' prediction output, showing its name and color coded by the modeling technique. C. Users can brush to select data instances (shown in white), LEGION highlights preferred models. D. Test Instances.

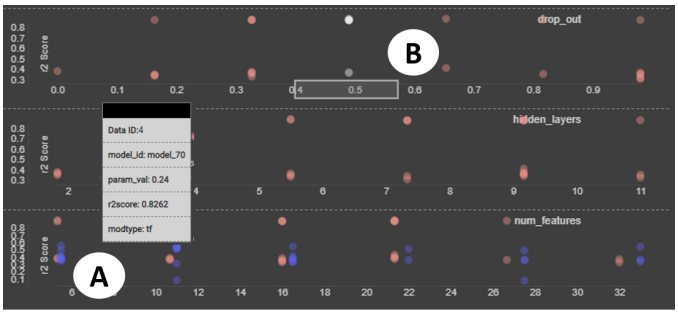

Fig. 3. Parameter Comparison View - A. Hovering on a model based on the hyperparameter value. B. User brushes mouse on the x-axis to highlight chosen models on the *Horizontal Stacked Bar View*.

construction techniques, each leads to models that may be are similar or different from one another. Users should be able to perceive these similarities and incoherencies amongst models as they compare a set of candidate models.

DG2 : Present model implication on the data, and the use of specific features: Users should be able to view a models' implication on the data (at the instance level) in terms of prediction output value. Furthermore, they should be able to learn about features in the data that are more relevant to the construction of better performing models; be able to adjust the selection and the use of these features as they incrementally construct models.

DG3 : Guide users with model recommendations: There can be many regression models constructed using either the hyperparameter tuning or the random selection of feature technique. The system should guide the users to look at specific models that may better suit their data analytic goals. There should be a system guided recommendation of models that assists users to inspect specific models as they interact with the visual interface.

DG4 : Interactive feedback driven construction and selection of new models: System should allow the possibility of interactive adjustment of models performance, and future construction and selection of new models by the user. As such, users interactive feedback should directly adjust the construction of new models and the underlying search process of the system, as it samples new model using the discussed construction techniques.

Based on these guidelines, we set forth the following set of tasks for our proposed system.

T1 : Interactive construction of models using two strategies: Users should be able to interactively construct regression models using two strategies: hyperparameter tuning and random selection of features. Users should be able to incrementally construct these models by adjusting their input preferences through interaction.

T2 : Compare models to select an appropriate modeling strategy in relation to subset data instances: Users should be able to compare models with each other with the goal to decide on a preferred modeling strategy. In doing so, they should learn similarities and differences between them, and other critical tradeoffs as they decide on selecting their preferred model and modeling strategy for the task.

T3 : Allow users to interactively navigate within the space of sampled candidate regression models. Users should be able to provide feedback to the system by directly manipulating visual encoding of various entities represented in the system or through control panel style input widgets.

3.2 User Interface

LEGION contains these main views: (1) Stacked Horizontal Bar View, (2) Radial View, (3) Instance Unit View, (4) Feature Explorer View, and (3) Parameter Comparison View, as explained below:

Stacked Horizontal Bar: This view shows regression models as horizontal bars placed in separate bins; each representing models constructed using a specific modeling technique such as hyperparameter tuning or random selection of features. In this prototype two such bins can be seen (Figure 1-A), however more can be added. Every row in the bin contains a set of bars (each a regression model). While the size of the bars encode a specified model metric such as RMSE value, the color encodes another metric such as residual error or adjusted $R^2 - Score$. Inspired from the abacus design, this view is designed to cluster models such that better performing models can be easily visually perceived from the large collection of models. Within each bar a horizontal black line encodes number of features used by the model in its training process. In addition, users can toggle to see models from all the bins in one bin vertically stacked with each other, allowing users to directly compare models' prediction metrics.

On a typical laptop screen size each bin can hold approx. 70-80 models, and if more models needs to be added, this view can scroll to scale. The bars that are placed per row (in a bin) are optimized to accommodate models as they are trained. To visually cluster high performing models together, they are grouped with relatively poor performing models per row. Here performance is indicated by the length of the bar; higher performing models are placed on the left. By default, the vertical ordering of the models are guided by the order of model construction, however, users can sort the models vertically by a specified metric e.g., residual error, RMSE, $R^2 - Score$, number of correct predictions on the test set, etc. (Figure 5-D). The visual layout helps in comparison of models within a bin and also with models from the adjacent bin. Furthermore, this design adapts progressive visual analytic principles [59], meaning as models are constructed they are placed per row, without the need to re-calculate the layout as we see in some other visual arrangement techniques e.g, force directed graph layouts etc.

Radial View: This view places two stacked radial bar charts next to each other, each representing a set of recommended models' performance metric on training and test set (see Figure 1-E). When a user clicks on a bar on the *Stacked Horizontal Bar View*, a set of 'k' recommended models are shown in this view, while the clicked model is highlighted with a white stroke. The color of the bars are synced with the *Stacked Horizontal Bar View* to ensure direct comparison of models can be made in relation to construction techniques. This view facilitates an overview of which model types performed better or worse on the training and test set. It is accom-

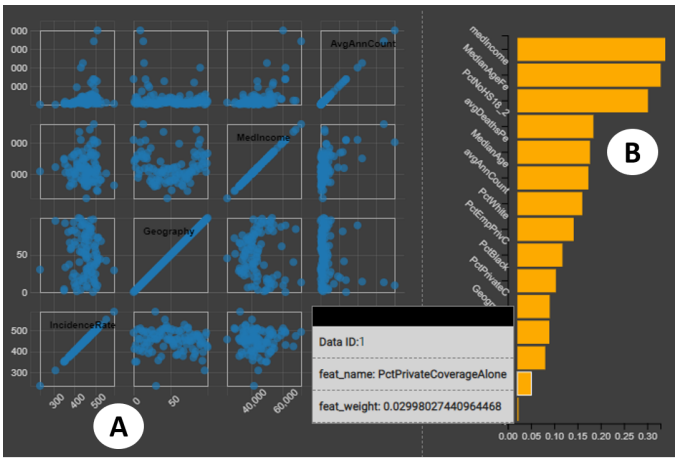

Fig. 4. Feature Explorer - A. Scatterplot matrix showing highly variant features. B. Horizontal bar chart showing top weighted features in a model.

panied by a textuall description of the characterisic of a selected model and the total collection of models as seen in Figure 1-D.

Instance Unit View: This view lists prediction output from 'k' recommended models as horizontal bar plots, where the x axis plots data instances and y axis plots the prediction error (Figure 2-B). Each models' prediction error per data instance on the training and validation set are encoded as vertical bars. If the error is positive then the bars are on top of the x axis, while if negative then the bars are on the bottom. The color ramp further emphasizes this concept. The model type (HPT or FST) is shown by a colored badge on its left. Users can brush over the x axis to highlight a subset of data instances. In response, LEGION highlights recommended models on the *Stacked Horizontal Bar View* that performed better on the chosen data instances.

Feature Explorer: When models are selected by users or when new models are constructed then the top performing features are shown in this view (Figure 4). They are shown as: (1) a set of 'k' (k=3, can be adjusted) scatterplot charts showing correlation between the feature and the target variable, (2) a scatterplot matrix of top 4 highly variant features, and (3) a horizontal bar chart listing highly weighted features in the model. Users can brush over the correlation scatterplots or the scatterplot matrix to highlight data instances in the *Instance Unit View*, prompting LEGION to recommend and highlight models in the *Stacked Horizontal Bar View* that performed well on these data instances.

Parameter Comparison View: In this view a set of unit visualizations shown that are stacked vertically (Figure 3). Each such view encodes R2 score (on test set) on its y axis, while the x axis shows a model hyperparameter such as *learning-rate*, *drop-out-value*, etc. Models are encoded as circles where the color encode the model type (HPT or FST). Using this view the user learns about hyperparameter settings that led to the construction of better performing models and their types. Users can brush on the x axis (Figure 3-B) to highlight the models on the *Stacked Horizontal Bar View*. One of the views in this set encodes number of features on the x axis, communicating how many feature combinations led to the construction of better performing models (e.g, $R^2 - Score$).

3.3 Technique

The underlying multi-model construction and selection of models adopts a methodical and systematic process further discussed here.

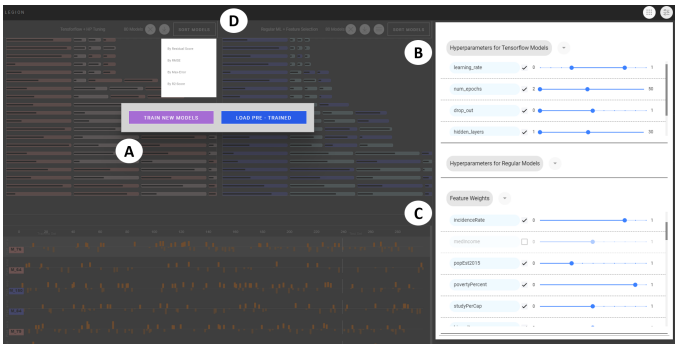

Fig. 5. A. Users can train new models, or load pre-trained models. B. Control panel to specify hyperparameter range values. C. Control panel to specify selected features and their weightings. D. Models can be sorted by a performance metric value.

Model Construction Technique: Our technique builds two types of models. The first type U, represents models constructed using Tensorflow JS simulating a linear regression model. In our implementation we adapted this example [60] and further modified the architecture of the network to adapt to the tested datasets. We setup the base Tensorflow model with hyperparameters such as *num-hidden-layers* $= 3$, *nodes-in-hidden-layer* $= 20$, *drop-out-rate* $=0.01$, *optimizer* $=$ '*sgd*', *num-epochs* $= 20$. The other model type V is a custom-designed boosted ridge-regression model with 'k' (a hyperparameter) underlying regression models v_i. We utilised ridge-regression model from the Machine Learning JS Library [61]. Overall, V takes these hyperparameters: *num-epochs* $= 10$, *learning-rate* $= 0.12$, *l2-weight* $= 0.03$, *loss-func* $=$ '*leastsquares*', *k-models* $= 5$. These hyperparameters are set by running the model with randomly assigned hyperparameters $n = 10$ (value can be adjusted) times, and the best performing models' hyperparameter settings were used to construct V, and never adjusted when user interactively constructed models. We also considered using the same learning algorithm for both U and V, however, tuning hyperparameters for a Tensorflow model seemed a more common practice by ML developers [62].

Input Data and Model overfitting: Our technique consumes the input data D, splits it into a training set T, a set of m vallidation sets S, and a test set R. As users iteratively construct models, T is used to train new regression models (U and V), however, per iteration a new validation set from $S = S_1, S_2, S_3 ....S_m$ is used to test the model. Thus the trained model (U or V) never sees the validation data instances before. Multiple sets of validation data is used to ensure the model does not overfit and always is tested on unseen data instances. Finally users can validate the trained model on R as they export any model.

Hyperparameter Tuning: Furthermore, as discussed earlier, models of type U are optimized using the hyperparameter tuning technique. To that end, we utilised the HPJS library [15], to tune hyperparameters of the Tensorflow models (U). To HPJS, our technique provides the model U, a set of hyperparameters H to tune, and a domain range d (within which HPJS samples hyperparameter values) to construct new regression models. We also provide a loss metric ω = 'meanSquaredError' to drive the hyperparameter optimization process guided by HPJS.

Feature Engineering: Feature engineering ensures that predictors (features) from the input data are encoded in a manner to maximize a models' performance. Within the scope of feature engineering, there is feature transformation and feature selection.

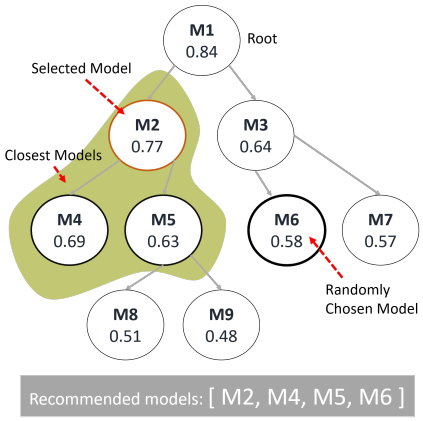

Fig. 6. Flow-diagram of the max-heap data structure to store models for recommendation. Numbers represent a score computed for each model.

While feature selection limits number of features utilised to train the models, feature transformation includes various family of algorithms that performs operations on the input features, such as scaling, normalizing, random projections etc. to ensure the model accurately gets informed from the data. Our technique allows random selection of features F and application of a preset list of feature transformation rules A (randomly chosen) to these features before constructing models. Let's say input data has l features , $F = F_1, F_2, F_3 .... F_l$, then our modeling engine, samples multiple models e,g, $V = V_1, V_2, V_3 ...$, where each V_i is trained using j ($j <= l$) randomly chosen features e.g., $F = F_3, F_4, F_{10} ....$ Similarly, a subset of the features in F are transformed using a random selection of feature transformation rules in A. For this prototye A includes, *variance scaling*, *standardization*, *mean removal*, and *discretization* (can be extended in future).

Algorithm 1 Algorithm to recommend models

```
INPUT: u_inst = user selected data, num = number of models
PARAMS: model_list, data                    ▷ stored on the backend
function BUILD_MODEL_HEAP
    for each item in data do
        m_hashmap ← create_heap(model_list, data[item])
    end for
    return m_hashmap ▷ dict of model heap per data instance
end function
function FIND_RECOMM_MODELS(u_inst, num)
    m_hashmap ← build_model_heap()
    model_arrs ← []                 ▷ list to store closest models
    for each item in u_inst do
        m_recom ← get_recom_models(m_hashmap, item, num)
        m_others ← get_rand_models(item, 1)
        model_arrs.concat(m_recom, m_others)
    end for
    recom_models ← get_overlap_mods(model_arrs, num)
    if recom_models.length == 0 then
        recom_models ← model_arrs.slice(0, num)
    end if
    return recom_models          ▷ list of recommended models
end function
```

Model Recommendation Technique: User interaction in LEGION includes: (1) selecting data instances from either the *Instance Unit View* or the *Feature Explorer View*, and (2) selecting models from the *Stacked Horizontal Bar View*. As they interact, a set of high performing models are recommended for them to consider. To implement this workflow we designed our model recommendation engine inspired by the previous work of Vainshtein et al. [63] and Shapira et al. [64]. In this technique we represent various characteristics of models (U, V) by extracting their meta features in M. It contains information such as model hyperparameters H, model metrics W (containing RMSE, $R^2 - Score$, Adjusted $R^2 - Score$, Max-Error, and Residual-Score), feature weights F and user feedback E. E may include number of times users exported a model, clicked on it, or if they liked it. We weight H, W, F and E by $\phi = \phi_h, \phi_w, \phi_f$, and ϕ_e, respectively. These weights are hand chosen by us by repeated trial and error with various datasets. Finally, we rank the models using M and ϕ; a process that generates a rank-score (Q_u, and Q_v) to each model in U and V respectively. Next we describe how LEGION recommends models for the two types of user interaction.

Using Q_u, and Q_v, we store the models in two max-heap data structures (HP_u and HP_v), where the model with the maximum rank-score (e..g, Q_{uk}) is stored at the root (see Figure 6). In the first interaction type, when users select a model (say U_i with rank-score Q_{ui}) our technique queries HP_u for top k closest models to Q_{ui}. The max-heap responds with closest models C_u with a similar rank-score (see Algorithm 1) and then visualizes it. As the rank score is derived from various model characteristics, (as summarized in its meta data M), such as hyperparameters, model metrics, number of features and user feedback, the set of models in C_u are expected to be similar to U_i. Besides showing closest models to Q_{ui}, we also include a handful set of randomly chosen models to allow users the opportunity to explore a diverse set of models.

The second interaction type is when users select data instances. We use trained models to predict labels for training set T and test set S_i as L_t and L_s respectively. Next we store a hash-map (e.g., a dict in python) data structure D, where the key is the data instance id, and the value is a max-heap of 'K' models that performed better than others in the collection. When new models are trained we look up D; for each data instance (both train and test set) we check if that model performed better than any other model already stored in the max-heap. If yes, we replace it with this model. However, if the max-heap is empty then, if the model satisfies a threshhold value α (a hyperparameter) then it gets stored in the max-heap. Constructing the hash-map D takes $O(vlogK)$ time, where v is the number of data instances in the training or test set and K is the number of recommended models we want to store. As users select specific data instances, the system refers D and recommends K models (either U or V) that performed better on these instances.

4 Usage Scenarios

4.1 Death Rate Prediction

LEGION splits the data into a training set (2100 rows), and a set of 3 validation sets (each 300 rows, utilised per iteration of model construction). When loaded, the system asks Taylor to train new models or load previously trained models (Figure 5-A). They decide to train a set of new regression models. After few minutes ($15 - 18$) of training, LEGION automatically saves the weights of these models for future use. Next Taylor explores the **Stacked Horizontal Bar View** (see Figure 1-A) to see two bins which stores: (1) 36 linear regression models constructed using Hyperparameter Tuning (HPT, implemented using TensorFlow), and (2) 42 Boosted Regressors using random selection and

transformation of features (FST) technique. Seeing the length of the bars in each bins Taylor notices that most of the models in the FST category are better performers in relation to the default metric - 'RMSE' (other metrics can be specified through a control panel). They hover over one such model Model $M-12$ to see that its $R^2-Score$ is 0.231 and its *RMSE* is 56.3, which is still high, but better than the one shown by the base line model. On clicking the model $M-4$ from the FST category, Taylor sees a recommendation of 20 (6 models from HPT category) other similarly high performing models on the **Radial View** (see Figure 1-E). Based on the color overlays representing model types on the two Radial Bar plots, they see that the models trained using HPT performed better on the test set with a lower RMSE of 53.71.

Inquisitive to learn more about models trained by HPT, They click on the model $M-28$ (*RMSE* = 58.73, $R^2-Score$ = 0.382, on validation set) and open the **Parameter Comparison View** to see its hyperparameters (Figure 1-C). From this view, they learn that there are about 8 high performing models in HPT category trained using *learning-rate* in the range of $0.05-0.08$, *num-hidden-layers* as 3, and $drop-out-rate$ in the range of $0.25-0.33$. Furthermore, Taylor learns that 10 out of 34 features (e.g., *median-age-male*, *avg-household-size*, *birth-rate*, etc.) were weighted heavily by these models. They open the **Feature Explorer View** (see Figure 4) to inspect the scatterplot matrix and the correlation plot to learn that there is a strong correlation between some of these features e.g., *median-age-male*, and *median-income* and the dependent variable. However, these models did not include the features *percent-married*, and *incidence-rate* that Taylor considered to be very strong predictors of the *Death-Rate*. To improve the models' performance further, Taylor opens the **Control Panel**, adjusts the hyperparameters and selects a subset of total 18 features (choices made based on the interaction so far, and their prior knowledge). Next, they trigger LEGION to construct new models.

As LEGION responds, they inspect new models shown in the **Stacked Horizontal Bar View** (Figure 1-A). Based on the bar size, and the color (encoding a higher R2 Score), they click on Model $M-17$ (with RMSE, 12.23) from the FST category and inspect the **Instance View** to see the prediction errors per data instance (on both train and validation set). They understand that the model overfitted as they find many long error bars on the validation set (see Figure 2). They drag their mouse on the training instance rows from $120-180$ (where the error seemed highest, see (Figure 2-C). In response LEGION highlights recommended models (on the **Stacked Horizontal Bar View**) that performed better on data these instances From the recommended models, Taylor clicks on Model $M-21$ (RMSE = 10.31, $R^2-Score$ = 0.875 on validation set, with FST technique); they notice that this model did not overfit (as seen in the **Radial Views**, Figure 1-E). Finally Taylor inspects the model hyperparameters and the features used in the model training from the **Parameter Comparison View** (Figure 3) before exporting it to Jupyter notebook for further analysis. Through this use case, we demonstrate how Taylor was able to explore two different regression modeling strategies to compare candidate models. LEGION assisted them in model construction and further, to select a generalizable model that trained on features that Taylor trusted.

4.2 Stock Price Prediction

Consider Andy is a stock market analyst and wants to predict future stock price variations to recommend which stocks to buy or sell. They have taken an elementary online course on Machine Learning, and using this knowledge is motivated to construct a regression model on the US Stock Price dataset [20]. This data contains 22000 rows of stocks from the year $2014-2018$. It has 225 columns including *revenue*, *inventory-growth*, *avg-payable*, *debt-to-assets*, *debt-to-equity*, etc. The target variable is the column *price-variation*, which indicates how much the stock price may vary (expressed in percentage, can be negative). If the value is positive then Andy would recommend to buy the stock, while if it is negative then their recommendation will be to sell the stock. As the data contains a plethora of financial indicators as features, Andy is not certain which of these are stronger predictor of the target variable than others. To continue analysis, they load the data in LEGION and construct many regression models.

LEGION first splits the data into a training set of 16000 rows, and 3 validation sets (each 2000 rows), then responds with 72 regression models built with hyperparameter tuning (HPT) and 83 regression models built using random selection of features (FST) technique. On the **Radial View**, Andy notices top 20 recommended regression models' with their $R^2-Score$ on train and validation set. From this set, they click on Model $M-32$ of FST type with RMSE as 132.2, and $R^2-Score$ as 0.541 on validation set. Furthermore, in the **Stacked Horizontal Bar View** (Figure 1-A), they see a short black line indicating that this model used only 32 (out of 225) features from the data to map its characteristic. On inspecting the Scatterplot Matrix and the Feature Weight bar plot (Figure 4), they notice that this model left out many important indicators of stock price variation such as *current-ratio*, *price-to-sales-ratio*, *earnings-yield* etc. On the **Stacked Horizontal Bar View**, they click on another model $M-43$ of similar RMSE (112.23) and $R^2-Score$ of 0.592 (perceived based on the length of the bar and its color encoding). LEGION updates the **Radial View** (Figure 1-E) with a set of new recommended models and the **Feature Explorer View** with the features that this model trained on. From the correlation plot in the **Feature Explorer**, they find that this model used more relevant features than the other and thus seem more useful for their analysis. Next they open the **Instance Unit View** (Figure 2) to further learn that the most of the high performing models used atleast 130 features (out of 225). To retrieve a model with a much lower RMSE Andy inspects models from the HPT type. They brush their mouse on the **Parameter Comparison View** (Figure 3) in the neighborhood of *learning-rate* with value 0.003 and *hidden-layer* with value 3. In response, LEGION highlights high performing models (on the **Stacked Horizontal Bar View**); from which Andy clicks on Model $M-12$ with $RMSE = 89.24$ and inspects the **Instance Unit View** to see its implication on the data.

They find that this model performed poorly on the validation set, though showed relatively lower RMSE. They brush their mouse on a set of important stocks from the training set where the prediction error was higher (see Figure 2-C). In response LEGION highlights a set of model of both types (HPT and FST) that performed better on these instances. From the recommended models Andy (as seen on the **Stacked Horizontal Bar View**, Figure 1-A), they click on Model $M-74$ (RMSE = 81.2) to understand that this model used most of the relevant features from the data (over 143 features). In order to further find better performing models Andy opens the control panel (Figure 5-B,C), adjusts the hyperparameter settings based on the values they learned from the **Parameter Comparison View**, and discards 30 features that have missing values and are outliers (learned from the correlation plots and the scatterplot matrix). Next they trigger LEGION to construct new models.

After an hour of training, Andy continues their model con-

struction and model exploration process. LEGION shows new models and recommends better performing models on the **Radial View**. Andyotices that the avg. $R^2 - Score$ has gone up to 0.962 (compared to 0.541 before). Based on the recommendations, Andy clicks on Model $M-11$ (RMSE = 12.23, FST type) and inspects the **Instance Unit View**. They learn that this model overfitted and thus brushes on a subset of incorrectly predicted validation data instances. As LEGION highlights other recommended models on the **Stacked Horizontal Bar View**, they inspect Models $M-5$ to $M-13$, all similarly high performing models of HPT type. They see from the horizontal black line that some of these models weighted 64 features very highly most of which were very relevant to predict the price variation of the stock. Next, Andy explores these models on the **Parameter Comparison View** to learn about their hyperparameter settings (Figure 3). They choose to select Model $M-12$ as it exhibited a simpler model architecture (based on the hyperparameter values). Finally, Andy exports the model to continue analysis and report results to their manager to recommend stock buying options to their clients.

5 Discussion and Limitations

Broader implication of model comparison: In this paper, we were driven by the intuition that model developers spend extensive time in tuning models to achieve a desired goal and that entails exploring and experimenting with numerous modeling strategies. We hypothesize that knowing about the merits of different modeling strategies early on, in the model selection process may save time, effort, and compute resources. This paper specifically extends current research on model selection, by empowering users not only to select a model, but also facilitating the opportunity to select a preferred modeling strategy for their problem. While model selection means completing a data analytic task, selecting a modeling strategy helps users to learn and discover something novel about the modeling process.

That being said, this paper demonstrated visual comparison of two of the common modeling strategies. However, there are many other strategies such as using data augmentation, changing input data size (e.g, analysis of learning curves in ML [65]), etc. to construct models. The visual design of the (*Stacked Horizontal Bar View*) is designed to scale with the number of comparison one needs to make; for example to compare four modeling techniques, the current design would present four adjacent bins, each holding bars or sequence of cell grids to encode models. In the future we plan to test the current visual design further, with more than three modeling strategies side by side.

Tradeoffs in modeling strategies: From the use cases (refer Section 4.1 and 4.2), we learned that for both the datasets the modeling problem was not so straight forward, and thus needed extensive exploration of the model space. Furthermore, through these use cases we demonstrated how LEGION assisted users in not only comparing models in relation to construction techniques, number of features, and hyperparameter settings applied, but also facilitating learning the data and the useful predictors (features) in it. For example, using our system users may filter similarly performing 'k' models and then compare them in relation to how simple or complex they are, how interpretable they can be, or what features they use to map the data; knowledge about these aspects may guide users to make informed choices as they select models. LEGION assists users in adapting the 'Occam's Razor' concept [66] that advocates selecting simpler models from a set of candidate models, given all of them perform highly/similarly on a specified performance metric. In the future, we plan to extend this functionality by further recommending adjustments users can make, that may reduce complexity of a black box model without losing its performance.

Advent of Feature Engineering in LEGION: Feature engineering presents ways to represent input data by applying operations that transforms raw data into a format that better characterizes the underlying information, directly impacting models' predictive performance. As part of feature engineering one can either select features, apply specific transformative operations on chosen features (e.g, scaling, normalizing, etc.), or can extract/construct new features e.g., create new features from existing ones. In this paper we have shown implication on models' prediction when constructed using random selection of features and random application of transformative operations. However, at present LEGION does not support feature extraction/construction [67], which we plan to prototype and test in the near future. Another approach we may consider is to apply feature augmentation using new data sources [68] to construct new models for comparison.

Model agnosticity, and scalability: LEGION is designed to work with tabular data to build regression models. However, the backend is also flexible to support other ML tasks such as classification with little adjustments to the current code base. Furthermore, currently the system supports two modes of model construction: (1) real-time model training, and (2) pre-training models and then loading these models to continue analysis. The first approach is suitable for medium size datasets (rows < 10k, columns < 30), wherein models are visually rendered as they finish training (similar to progressive visual analytic style rendering of views [59]). The second approach is designed to support larger datasets (tested with dataset with row <= 100k, column <= 600), wherein users can train models overnight and then later analyse and compare models when the training is complete.

Furthermore, the *Horizontal Stacked Bar View*, is designed to scale with the number of models' it needs to render. It can be transformed into a heatmap view, where pixel(s) instead of bars can visually encode a model, the size or area of which can represent a performance metric. On the engineering side, LEGION is designed with Tensorflow JS and MachineLearning JS, both of which features client-side model computation, off-loading the need for powerful servers. On the frontend the visualizations are implemented using a game design framework Phaser JS [69], that exploits GPU rendering on a browser. Client-side computation is designed to take advantage of better compute devices many ML practitioners have and further reduce the system design complexity on the server-side.

6 Conclusion

In this work, we demonstrate that multi-model based visual analytic systems can visually represent models by their construction technique, leading to more in-depth comparison to assist users in model selection. With our prototype LEGION, we present how users can be guided as they decide on which model construction technique to adopt and how models can be selected given a plethora of tradeoffs and model incoherencies that exists in numerous ML problems. To that end, with two real world datasets we show how LEGION can help users to interctively construct regression models, comparing their tradeoffs, and providing interactive feedback to further adjust these models to suit their specific data analytic goals.

REFERENCES

[1] M. A. Ahmad, C. Eckert, and A. Teredesai, "Interpretable machine learning in healthcare," in *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*, ser. BCB '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 559–560. [Online]. Available: https://doi.org/10.1145/3233547.3233667

[2] S. M. Wiens J, Saria S, "Do no harm: a roadmap for responsible machine learning for health care," in *Proceedings of the 2018 ACM International Conference on Bioinformatics, Computational Biology, and Health Informatics*. Nat Med. 2019, 2019, p. 1337–1340.

[3] B. Park and J. K. Bae, "Using machine learning algorithms for housing price prediction: The case of fairfax county, virginia housing data," *Expert Systems with Applications*, vol. 42, no. 6, pp. 2928 – 2934, 2015. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0957417414007325

[4] Y. SUN, A. K. C. WONG, and M. S. KAMEL, "Classification of imbalanced data: A review," *International Journal of Pattern Recognition and Artificial Intelligence*, vol. 23, no. 04, pp. 687–719, 2009. [Online]. Available: https://doi.org/10.1142/S0218001409007326

[5] D. Dingen, M. van't Veer, P. Houthuizen, E. H. J. Mestrom, E. H. H. M. Korsten, A. R. A. Bouwman, and J. van Wijk, "Regressionexplorer: Interactive exploration of logistic regression models with subgroup analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 246–255, 2019.

[6] S. L'Yi, B. Ko, D. Shin, Y.-J. Cho, J. Lee, B. Kim, and J. Seo, "Xclusim: a visual analytics tool for interactively comparing multiple clustering results of bioinformatics data," *BMC Bioinformatics*, vol. 16, no. 11, p. S5, Aug 2015. [Online]. Available: https://doi.org/10.1186/1471-2105-16-S11-S5

[7] B. C. Kwon, B. Eysenbach, J. Verma, K. Ng, C. D. Filippi, W. F. Stewart, and A. Perer, "Clustervision: Visual supervision of unsupervised clustering," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 142–151, Jan 2018.

[8] M. Gleicher, A. Barve, X. Yu, and F. Heimerl, "Boxer: Interactive Comparison of Classifier Results," *Computer Graphics Forum*, 2020.

[9] H. Piringer, W. Berger, and J. Krasser, "Hypermoval: Interactive visual validation of regression models for real-time simulation," in *Proceedings of the 12th Eurographics / IEEE - VGTC Conference on Visualization*, ser. EuroVis'10. Chichester, UK: The Eurographs Association & John Wiley & Sons, Ltd., 2010, pp. 983–992. [Online]. Available: http://dx.doi.org/10.1111/j.1467-8659.2009.01684.x

[10] S. Das, D. Cashman, R. Chang, and A. Endert, "Beames: Interactive multimodel steering, selection, and inspection for regression tasks," *IEEE Computer Graphics and Applications*, vol. 39, no. 5, pp. 20–32, Sep. 2019.

[11] D. Ren, S. Amershi, B. Lee, J. Suh, and J. D. Williams, "Squares: Supporting interactive performance analysis for multiclass classifiers," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 61–70, Jan 2017.

[12] F. Hohman, A. Head, R. Caruana, R. DeLine, and S. M. Drucker, "Gamut: A design probe to understand how data scientists understand machine learning models," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*. ACM, 2019.

[13] M. Kahng, D. Fang, and D. H. P. Chau, "Visual exploration of machine learning results using data cube analysis," in *Proceedings of the Workshop on Human-In-the-Loop Data Analytics*, ser. HILDA '16. New York, NY, USA: Association for Computing Machinery, 2016. [Online]. Available: https://doi.org/10.1145/2939502.2939503

[14] J. Zhao, M. Karimzadeh, A. Masjedi, T. Wang, X. Zhang, M. M. Crawford, and D. S. Ebert, "Featureexplorer: Interactive feature selection and exploration of regression models for hyperspectral images," *2019 IEEE Visualization Conference (VIS)*, pp. 161–165, 2019.

[15] "Hp js, hyperparameter optimization library in js, howpublished = https://hyperjs.herokuapp.com/, note = Accessed: 2020-16-07."

[16] "Tensorflow js, howpublished = https://www.tensorflow.org/js, note = Accessed: 2020-16-07."

[17] "Machine learning js, howpublished = https://www.machinelearnjs.com/api/linear_model.sgdregressor.html, note = Accessed: 2020-16-07."

[18] M. Cavallo and Demiralp, "Clustrophile 2: Guided visual clustering analysis," *IEEE Transactions on Visualization and Computer Graphics*, pp. 1–1, 2018.

[19] "Cancer mortality rates for us countiess, howpublished = https://data.world/nrippner/ols-regression-challenge, note = Accessed: 2020-16-07."

[20] "Financial indicators of us stocks, howpublished = https://www.kaggle.com/cnic92/200-financial-indicators-of-us-stocks-20142018?select=2015_financial_data.csv, note = Accessed: 2020-16-07."

[21] E. T. Brown, J. Liu, C. E. Brodley, and R. Chang, "Dis-function: Learning distance functions interactively," in *Proceedings of the 2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*, ser. VAST '12. Washington, DC, USA: IEEE Computer Society, 2012, pp. 83–92. [Online]. Available: http://dx.doi.org/10.1109/VAST.2012.6400486

[22] L. Bradel, C. North, L. House, and S. Leman, "Multi-model semantic interaction for text analytics," in *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*, Oct 2014, pp. 163–172.

[23] A. Endert, P. Fiaux, and C. North, "Semantic interaction for visual text analytics," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '12. New York, NY, USA: ACM, 2012, pp. 473–482.

[24] S. C. Leman, L. House, D. Maiti, A. Endert, and C. North, "Visual to parametric interaction (v2pi)," *PloS one*, vol. 8, no. 3, p. e50474, 2013.

[25] A. Endert, C. Han, D. Maiti, L. House, S. C. Leman, and C. North, "Observation-level Interaction with Statistical Models for Visual Analytics," in *IEEE VAST*, 2011, pp. 121–130.

[26] A. Endert, L. Bradel, and C. North, "Beyond Control Panels: Direct Manipulation for Visual Analytics," *IEEE Computer Graphics and Applications*, vol. 33, no. 4, pp. 6–13, 2013.

[27] S. Gratzl, A. Lex, N. Gehlenborg, H. Pfister, and M. Streit, "Lineup: Visual analysis of multi-attribute rankings," *IEEE Transactions on Visualization and Computer Graphics (InfoVis '13)*, vol. 19, no. 12, pp. 2277–2286, 2013.

[28] M. Kahng, P. Y. Andrews, A. Kalro, and D. H. Chau, "Activis: Visual exploration of industry-scale deep neural network models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, pp. 88–97, 2017.

[29] H. Li, S. Fang, S. Mukhopadhyay, A. J. Saykin, and L. Shen, "Interactive machine learning by visualization: A small data solution," in *2018 IEEE International Conference on Big Data (Big Data)*, Dec 2018, pp. 3513–3521.

[30] M. Liu, S. Liu, H. Su, K. Cao, and J. Zhu, "Analyzing the noise robustness of deep neural networks," *2018 IEEE Conference on Visual Analytics Science and Technology (VAST)*, pp. 60–71, 2018.

[31] Y. Sun, E. Lank, and M. Terry, "Label-and-learn: Visualizing the likelihood of machine learning classifier's success during data labeling," in *Proceedings of the 22Nd International Conference on Intelligent User Interfaces*, ser. IUI '17. New York, NY, USA: ACM, 2017, pp. 523–534. [Online]. Available: http://doi.acm.org/10.1145/3025171.3025208

[32] M. Chegini, J. Bernard, P. Berger, A. Sourin, K. Andrews, and T. Schreck, "Interactive labelling of a multivariate dataset for supervised machine learning using linked visualisations, clustering, and active learning," *Visual Informatics*, vol. 3, no. 1, pp. 9 – 17, 2019, proceedings of PacificVAST 2019. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S2468502X19300178

[33] H. Kim, J. Choo, H. Park, and A. Endert, "Interaxis: Steering scatterplot axes via observation-level interaction," *IEEE Transactions on Visualization and Computer Graphics*, vol. 22, no. 1, pp. 131–140, Jan 2016.

[34] D. Sacha, L. Zhang, M. Sedlmair, J. A. Lee, J. Peltonen, D. Weiskopf, S. C. North, and D. A. Keim, "Visual interaction with dimensionality reduction: A structured literature analysis," *IEEE Transactions on Visualization and Computer Graphics*, vol. 23, no. 1, pp. 241–250, 2017.

[35] E. T. Brown, J. Liu, C. E. Brodley, and R. Chang, "Dis-function: Learning distance functions interactively," in *2012 IEEE Conference on Visual Analytics Science and Technology (VAST)*, Oct 2012, pp. 83–92.

[36] S. Amershi, M. Chickering, S. Drucker, B. Lee, P. Simard, and J. Suh, "Modeltracker: Redesigning performance analysis tools for machine learning," in *Proceedings of the Conference on Human Factors in Computing Systems (CHI 2015)*, April 2015.

[37] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-weka: Combined selection and hyperparameter optimization of classification algorithms," in *Proceedings of the 19th ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2013, pp. 847–855.

[38] L. Kotthoff, C. Thornton, H. H. Hoos, F. Hutter, and K. Leyton-Brown, "Auto-weka 2.0: Automatic model selection and hyperparameter optimization in weka," *Journal of Machine Learning Research*, vol. 17, pp. 1–5, 2016.

[39] S. Paparizos, J. M. Patel, and H. Jagadish, "Sigopt: Using schema to optimize xml query processing," in *Data Engineering, 2007. ICDE 2007. IEEE 23rd International Conference on*. IEEE, 2007, pp. 1456–1460.

[40] T. T. Le, W. Fu, and J. H. Moore, "Scaling tree-based automated machine learning to biomedical big data with a feature set selector," *Bioinformatics*, vol. 36, no. 1, pp. 250–256, 2020.

[41] J. Bergstra, D. Yamins, and D. D. Cox, "Hyperopt: A python library

for optimizing the hyperparameters of machine learning algorithms," in *Proceedings of the 12th Python in Science Conference*, 2013, pp. 13–20.

[42] B. Komer, J. Bergstra, and C. Eliasmith, "Hyperopt-sklearn: automatic hyperparameter configuration for scikit-learn," in *ICML workshop on AutoML*, 2014.

[43] M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Advances in Neural Information Processing Systems*, 2015, pp. 2962–2970.

[44] L. T., G. Convertino, W. Wang, and H. Most, "Hypertuner: Visual analytics for hyperparameter tuning by professionals," *Machine Learning from User Interaction for Visualization and Analytics, IEEE VIS 2018*, 2018.

[45] M. K. J.-H. K. J. C. J.-W. H. N. S. Heungseok Park, Jinwoong Kim, "Visualhypertuner: Visual analytics for user-driven hyperparamter tuning of deep neural networks," *Proceedings of the 2nd SysML Conference*, 2019.

[46] I. Drori, Y. Krishnamurthy, R. Rampin, R. Lourenço, J. P. Ono, K. Cho, C. Silva, and J. Freire, "Alphad3m machine learning pipeline synthesis," 2018.

[47] L. Li, K. G. Jamieson, G. DeSalvo, A. Rostamizadeh, and A. Talwalkar, "Efficient hyperparameter optimization and infinitely many armed bandits," *ArXiv*, vol. abs/1603.06560, 2016.

[48] D. Sacha, M. Sedlmair, L. Zhang, J. A. Lee, J. Peltonen, D. Weiskopf, S. C. North, and D. A. Keim, "What you see is what you can change: Human-centered machine learning by interactive visualization," *Neurocomputing*, vol. 268, pp. 164–175, 2017.

[49] L. Bradel, C. North, L. House, and S. Leman, "Multi-model semantic interaction for text analytics," in *2014 IEEE Conference on Visual Analytics Science and Technology (VAST)*, Oct 2014, pp. 163–172.

[50] K. Patel, S. M. Drucker, J. Fogarty, A. Kapoor, and D. S. Tan, "Using multiple models to understand data," in *Proceedings of the Twenty-Second International Joint Conference on Artificial Intelligence - Volume Volume Two*, ser. IJCAI'11. AAAI Press, 2011, pp. 1723–1728. [Online]. Available: http://dx.doi.org/10.5591/978-1-57735-516-8/IJCAI11-289

[51] A. Chatzimparmpas, R. M. Martins, K. Kucher, and A. Kerren, "Stackgenvis: Alignment of data, algorithms, and models for stacking ensemble learning using performance metrics," *ArXiv*, vol. abs/2005.01575, 2020.

[52] T. Mühlbacher and H. Piringer, "A partition-based framework for building and validating regression models," *IEEE Transactions on Visualization and Computer Graphics*, vol. 19, pp. 1962–1971, 2013.

[53] X. Zhao, Y. Wu, D. L. Lee, and W. Cui, "iforest: Interpreting random forests via visual analytics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 25, no. 1, pp. 407–416, 2019.

[54] S. Liu, J. Xiao, J. Liu, X. Wang, J. Wu, and J. Zhu, "Visual diagnosis of tree boosting methods," *IEEE Transactions on Visualization and Computer Graphics*, vol. 24, no. 1, pp. 163–173, 2018.

[55] J. Talbot, B. Lee, A. Kapoor, and D. S. Tan, "Ensemblematrix: interactive visualization to support machine learning with multiple classifiers," in *CHI*, 2009.

[56] Y. Zhao, S. K. Tasoulis, and T. Roos, "Manifold visualization via short walks," in *EuroVis*, 2016.

[57] G. Sehgal, M. Rawat, B. Gupta, G. Gupta, G. Sharma, and G. Shroff, "Visual Predictive Analytics using iFuseML," in *EuroVis Workshop on Visual Analytics (EuroVA)*, C. Tominski and T. von Landesberger, Eds. The Eurographics Association, 2018.

[58] K. Zhao, M. O. Ward, E. A. Rundensteiner, and H. N. Higgins, "Lovis: Local pattern visualization for model refinement," *Comput. Graph. Forum*, vol. 33, pp. 331–340, 2014.

[59] C. D. Stolper, A. Perer, and D. Gotz, "Progressive visual analytics: User-driven visual exploration of in-progress analytics," *IEEE Transactions on Visualization and Computer Graphics*, vol. 20, pp. 1653–1662, 2014.

[60] "Tensorflow js regression modeling, howpublished = https://codelabs.developers.google.com/codelabs/tfjs-training-regression/index.html#0, note = Accessed: 2020-07-13."

[61] "Machine learning js , howpublished = https://www.machinelearnjs.com/api/linear_model.sgdregressor.html, note = Accessed: 2020-07-13."

[62] C. Liu, B. Zoph, M. Neumann, J. Shlens, W. Hua, L.-J. Li, L. Fei-Fei, A. Yuille, J. Huang, and K. Murphy, "Progressive neural architecture search," in *Proceedings of the European Conference on Computer Vision (ECCV)*, September 2018.

[63] R. Vainshtein, A. Greenstein-Messica, G. Katz, B. Shapira, and L. Rokach, "A hybrid approach for automatic model recommendation," *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, 2018.

[64] N. Cohen-Shapira, L. Rokach, B. Shapira, G. Katz, and R. Vainshtein, "Autogrd: Model recommendation through graphical dataset representation," *Proceedings of the 28th ACM International Conference on Information and Knowledge Management*, 2019.

[65] C. Perlich, F. Provost, and J. S. Simonoff, "Tree induction vs. logistic regression: A learning-curve analysis," *J. Mach. Learn. Res.*, vol. 4, no. null, p. 211–255, Dec. 2003. [Online]. Available: https://doi.org/10.1162/153244304322972694

[66] A. Blumer, A. Ehrenfeucht, D. Haussler, and M. K. Warmuth, "Occam's razor," *Information Processing Letters*, vol. 24, no. 6, pp. 377 – 380, 1987. [Online]. Available: http://www.sciencedirect.com/science/article/pii/0020019087901141

[67] H. Liu and H. Motoda, *Feature Extraction, Construction and Selection: A Data Mining Perspective*. USA: Kluwer Academic Publishers, 1998.

[68] "Datamart, a dataset search engine and data augmentation platform," https://docs.auctus.vida-nyu.org/, accessed: 2020-16-07.

[69] T. Faas, *An Introduction to HTML5 Game Development with Phaser.Js*, 1st ed. USA: A. K. Peters, Ltd., 2016.