Theme Article: Visual Data Science

BEAMES: Interactive Multimodel Steering, Selection, and Inspection for Regression Tasks

Subhajit Das
Georgia Institute of Technology

Dylan Cashman
Tufts University

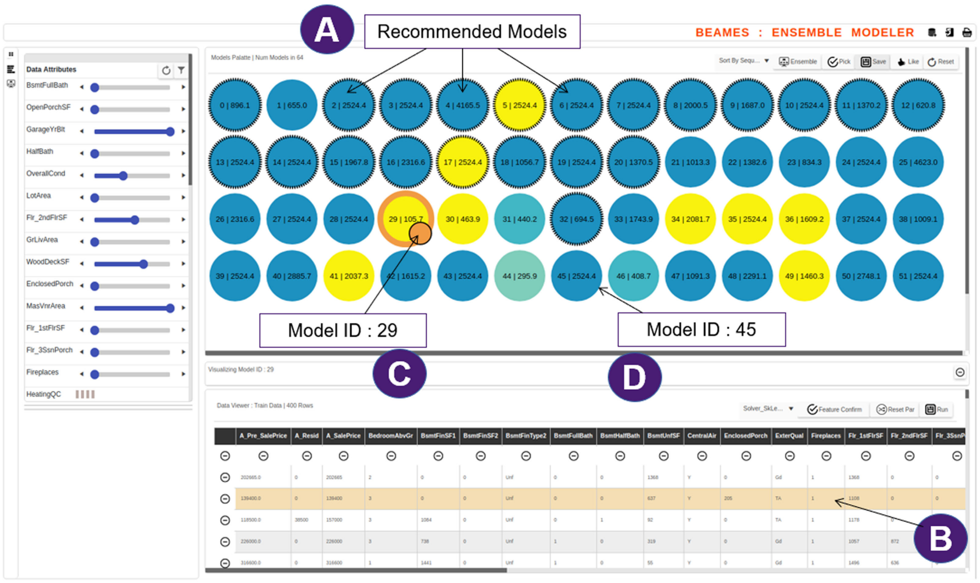
Remco Chang
Tufts University

Alex Endert
Georgia Institute of Technology

***Abstract*—Interactive model steering helps people incrementally build machine learning models that are tailored to their domain and task. Existing visual analytic tools allow people to steer a single model (e.g., assignment attribute weights used by a dimension reduction model). However, the choice of model is critical in such situations. What if the model chosen is suboptimal for the task, dataset, or question being asked? What if instead of parameterizing and steering this model, a different model provides a better fit? This paper presents a technique to allow users to inspect and steer multiple machine learning models. The technique steers and samples models from a broader set of learning algorithms and model types. We incorporate this technique into a visual analytic prototype, BEAMES, that allows users to perform regression tasks via multimodel steering. This paper demonstrates the effectiveness of BEAMES via a use case, and discusses broader implications for multimodel steering.**

DOMAIN EXPERTS USE interactive visual analytic systems to solve real problems spanning a broad set of domains. Many such systems incorporate machine learning models to help analyze the data. Users interact with such systems to explore their data by changing various parameters of the models, which in turn produce different results. This process, generally referred to as *model steering*, is complex. Users need to understand the

Digital Object Identifier 10.1109/MCG.2019.2922592
Date of publication 12 June 2019; date of current version 21 August 2019.



A_Pre_SalePrice	A_Resid	A_SalePrice	BedroomAbvGr	BsmtFinSF1	BsmtFinSF2	BsmtFinType2	BsmtFullBath	BsmtHalfBath	BsmtUnfSF	CentralAir	EnclosedPorch	ExterQual	Fireplaces	Flr_1stFlrSF	Flr_2ndFlrSF	Flr_3SsnP
202665.0	0	202665	2	0	0	Unf	0	0	1368	Y	0	Gd	1	1368	0	0
139400.0	0	139400	3	0	0	Unf	0	0	637	Y	205	TA	1	1108	0	0
118500.0	38500	157000	3	1084	0	Unf	0	1	92	Y	0	TA	1	1178	0	[illegible]
226000.0	0	226000	3	738	0	Unf	1	0	319	Y	0	Gd	1	1057	872	[illegible]
316600.0	0	316600	1	1441	0	Unf	1	0	55	Y	0	Gd	1	1496	636	[illegible]

Figure 1. BEAMES user interface (UI) for multimodel steering, selection, and inspection for regression tasks. The model view (A) shows circular glyphs representing regression models color coded by residual error. The data table (B) shows training, test, and application datasets. The control panel (C) allows users to filter models and critical instances, and change feature weights (D).

parameters of the models (and the general characteristics of the models) to properly convey their intention or domain expertise to the system, improve the models, and in turn gain insight.

Prior work has looked at this fundamental usability problem. For example, Endert *et al.* proposed *semantic interaction* as a method to couple model steering operations with native user interaction on the data. Interaction, such as highlighting phrases or text, performing a search, or grouping documents steer clustering and natural language processing algorithms. Daee *et al.* showed with a user study user feedback on feature relevance enhanced sparse linear regression models in a sentiment analysis task.[1] Yang *et al.* studied design implications for such a system, where nonexperts can interact with complex models to solve real life problems.[2]

While the advances of these works are impactful, the complexity of algorithmic support in visual analytic systems and machine learning continue to increase. The act of model steering is no longer limited to adjusting the parameters of a single algorithm. For example, systems like Interaxis,[3] Dis-Function,[4] AxisSketcher,[5] and others rely on interactive updating of a loss function based on user interaction to find optimal attribute weights of a single model that closely matches the domain expertise of the users. These single-model steering systems allow users to intuitively steer a model, without requiring knowledge of the underlying model and its parameters (e.g., References 3,4,6) However, single-model steering becomes less effective when the model chosen no longer fits the task or data characteristics. An incorrectly chosen model is hard to steer to get acceptable results. We call this problem *multimodel steering*, where interaction steers multiple models from a set of model types, and users ultimately select a best model for the task and domain.

Multimodel steering is a complex process requiring a considerable amount of technical expertise. A model is defined by a learning algorithm. Each algorithm relies on a set of hyperparameters. While a model trains on a dataset, it learns an optimal set of parameters and weights to precisely characterize the structure of the data, so that it can generalize well on an unseen dataset. Setting the correct learning algorithm and the right combination of hyperparameter values is critical to building an optimal model for a given problem type (i.e., classification, regression, clustering, etc.). For users without formal data science training, specifying each of these parameters may be difficult.

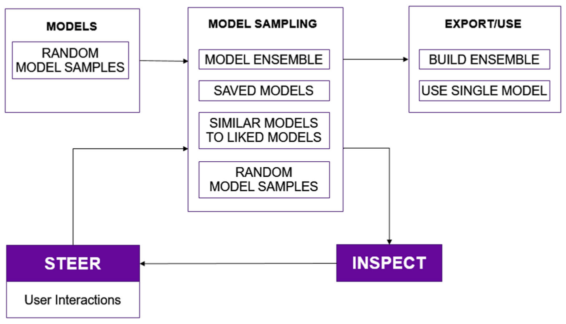
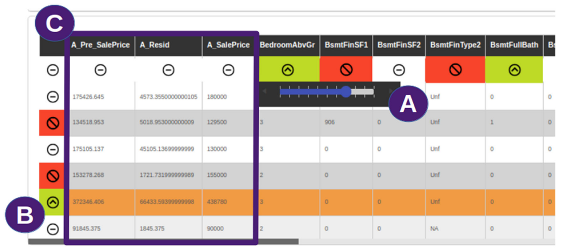


Figure 2. Conceptual diagram of our iterative technique for multimodel steering and inspection. Users interactively inspect and steer models, which are generated and sampled using a variety of techniques described in the System Description: BEAMES section. When satisfied, users can export either a model ensemble or a single model.



	A_Pre_SalePrice	A_Resid	A_SalePrice	BedroomAbvGr	BsmtFinSF1	BsmtFinSF2	BsmtFinType2	BsmtFullBath	Bs
⊖	⊖	⊖	⊖	🟩 ⌃	🟥 ⊘	⊖	🟥 ⊘	🟩 ⌃	
⊖	175426.645	4573.3550000000105	180000				Unf	0	0
🟥 ⊘	134518.953	5018.953000000009	129500	3	906	0	Unf	1	0
⊖	175105.137	45105.13699999999	130000	3	0	0	Unf	0	0
🟥 ⊘	153278.268	1721.731999999989	155000	2	0	0	Unf	0	0
🟩 ⌃	372346.406	66433.59399999998	438780	3	0	0	Unf	0	0
⊖	91845.375	1845.375	90000	2	0	0	NA	0	0

Figure 3. (A) Data table view showing the toggle switches (green, white, and red) for features. The sliders allow users to add weights to the data samples and attributes. (B) Similar toggle switches and sliders for data samples. (C) From left: Column 1 is the predicted output, Column 2 is the residual error, Column 3 is the ground truth value to predict, in this case sale price of a house.

In this paper, we describe a technique that allows users to inspect model outputs, give feedback, and in turn steer and select from multiple models. (See Figure 2). In this case, multiple models refers to a set of models formed by using different learning algorithms, where each algorithm is defined by using a range of values for its hyperparameters (e.g., Model 1 is LogisticRegression(alpha = 0.2), Model 2 is LogisticRegression (alpha = 5), Model 3 is BayesianRegression(alpha = 40), etc.).

The visual analytic technique presented in this paper allows domain experts to inspect models by checking a model's predicted output on the data (i.e., checking critical data instances). Accurate prediction of critical data instances can increase user's trust in the model. Our technique also allows people to steer and inspect multiple models. Further, our technique assists the inspection process by recommending models (from the collection of models) that successfully make predictions on the critical data instances with zero or relatively low error value. Showing a wide spectrum of models for the given regression problem can be beneficial to domain experts who otherwise would not be aware of the many possibilities and permutations of models. Being able to filter the data by instances and filter models by their performance (by simple double range sliders and toggling switches for categorical items), users can drill down to models that are successful and can validate them by checking their results on critical data instances. However, since this process is iterative, our technique provides an interactive visual interface to iteratively refine the critical instances and other user input to continue model steering and inspection. Further, our technique enables domain experts to add knowledge to the model building process. Users can add knowledge about which features may be more important than others, or which data instances are more important to correctly predict.

The technique presented in this paper has three primary components: first, interactive weighting of critical data instances, second, interactive feature selection with weights, third, interactive model selection, and fourth, building model ensembles. Users can steer multiple models simultaneously by increasing the weights on critical data instances (using the on-demand sliders shown in Figure 3), indicating that accuracy on these data items is more important. In addition, they can select features (by toggling checkbox type buttons) and specify their weights (by dragging sliders) to specify their relative importance. Users can also perform interactive model selection by "liking" one or more models, from which BEAMES generates a new set of similar models to inspect. Finally, they can select exporting a model that best fits their task, or generate an ensemble of models to use.

To summarize, our technique searches the model space for models that more closely adhere to data items and attributes the user is interested in. The set of models is refined at each step, where more models are generated and less relevant models are pruned. This human-in-the-loop process allows domain experts to explore a myriad of models for their regression task, and add domain expertise into the model building to produce models that adhere to both subjective and objective preferences of the users. The main contribution of this paper are as follows.

1. An interactive technique for domain experts to select an optimal model using multimodel steering aided by: interactive weighting of data instances; interactive feature selection and weighting; and building model ensembles.
2. A visual analytic system called BEAMES, which instantiates our technique to help people select a regression model using multimodel steering.
3. A usage scenario demonstrating the intended usage and interaction with the system.

RELATED WORK

Single Model Steering Systems

Interaction-based single model driven visual analytic systems have been around for a while, helping nontechnical users build and change model parameters by control panels or user interface (UI) elements that enable them to interactively demonstrate the feedback. The spectrum of the problem types these systems solve is adequately wide. It includes ranking,[6] metric learning,[4] dimension reduction,[7] feature selection,[8] etc. For instance, Podium[6] uses a single linear support vector machine (SVM) model with the goal to compute attribute weights based on the subjective preference of multiattribute data items.

In all of these examples the model infers parameters based on users demonstration by direct manipulation of graphical widgets. A relevant system is the work of Kim *et al.*, InterAxis,[3] which showed how users can drag data objects to the high and low locations on both axes of a scatterplot to help them interpret, define, and change axes with respect to a linear dimension reduction technique. Our work is distinct from existing work in that we enable users to steer (by interaction based user feedback) multiple machine learning models as opposed to single models. Also, our technique allows users to inspect multiple models simultaneously, leveraging them to evaluate and select an optimal model.

Liere *et al.* have defined computational steering as a process to enable users to change parameters of simulations on the fly.[9] Their paper emphasizes the concept that simulations run over many iterations, where users may need to update parameters before completion. In this paper, we ground our concept of model steering in the prior work, and refer to it as a process in which a model's parameters are changed to iteratively produce updated results, and multimodel steering as a process in which a model's hyperparameters and parameters are changed. It is similar to computational steering in which multiple iterative cycles are computed, and user input can change parameters or hyperparameters at any iteration. This also aligns well with how model steering has been implemented in visual analytic systems discussed above. Additionally, BEAMES also supports interactive model selection by allowing users to select a subset of models and generate a new set of similar models.

Multimodel Interaction and Model Ensembles

While single model systems contain a predefined machine learning (ML) model with a carefully selected learning algorithm and set of hyperparameters, multimodel visual analytic (VA) systems include multiple ML models each using a different set of learning algorithms and hyperparameters. Thus, we define multimodel steering as the interactive change of these models' hyperparameters. Similarly, Endert *et al.* introduced a related method for users to provide feedback called semantic interaction, which enables users to directly interact with models using visual interface components/elements (i.e., dragging visual data elements, such as dots in a scatterplot to specify similarity and brushing on a node-layout to define regions of interest).[10]

For instance, the work of Bradel *et al.*[11] lies in the multimodel steering space. Using semantic interaction, they allow users to steer multiple text analytic models. While effective, their system is scoped to text analytics and handling text corpora at multiple levels of the scale. Shneider *et al.* showed visual integration of data and model space, by allowing users identify effective component models on data items from a classification model ensemble.[12] Patel *et al.* showed an example technique to work with multiple model systems helping users understand relationship between data, models, and features.[13] Piringer *et al.* showed an interactive visual analytic system helping multiple regression model comparison and validation in an interactive fashion.[14] Their technique specifically uses comparison of multiple model outputs to help users select the best model. Kwon *et al.*[15] showed a tool to visually identify and select an appropriate cluster model from multiple clustering algorithms and parameter combinations. However, their work targeted data scientists as the user, whereas we are aiming to build techniques for domain experts without formal data science training.

The topic of model ensembles is related to our work, as one of the interactions provided to the user is to build ensemble models from a set of models they like. Model ensembles increase model performance by fusing multiple model's strength. Different strategies yield different kinds of model ensemble, for example, Potter *et al.*[16] showed an interactive ensemble model to simulation outcome exploration. In our case, users select models manually and combine them to build a model ensemble.

Automated Model Selection

Model building is a nontrivial task for nonexpert users, as it involves complexity including, selecting a good combination of learning algorithms and hyperparameters. One solution is to use existing automated model selection tools, such as AutoWeka,[17] a VA system Propsector,[18] or others.[19, 20,21,22] These tools follow numerous optimization procedures internally to find the right combination of algorithms and hyperparameters for an optimal model suited to the given dataset and task. The BEAMES leverages user feedback on model outputs to incrementally steer and select models that can be mapped to this comprehensive model.

SYSTEM DESCRIPTION: BEAMES

This section describes BEAMES, a visual analytic system for multimodel steering to solve a regression task. The BEAMES enables users to inspect outputs of multiple models, specify critical data instances, steer multiple models to increase or refine their performance, and finally select one model (or an ensemble of models) when satisfied. It assists the user to select an optimal regression model from a collection of models constructed by various combinations of regression algorithms and their hyperparameters. Some examples that the algorithms tested includes Bayesian regression, linear regression, and logistic regression (shown in Table 1).

The design rationale of the BEAMES strives to provide a simple user experience to select models without the multimodel complexity. To that end, the UI design encourages users to iteratively steer and select models by inspecting a subset of presented models without exploring the aspects of model construction and evaluation. The system should also show users aspects of regression models, such as residual scores, mean squared errors, or different types of regression methods. In addition, BEAMES shows an overview of the data instances and attributes so that users can input domain knowledge in the modeling pipeline at the level of data instances and data attributes. To support model selection at different scales, BEAMES shows models at different levels of detail to inspect models (i.e., to review model accuracy on the whole data or on specific data items) and to select models (i.e., to filter models by defined criteria or inspect recommended models).

User Interface

The user interface (shown in Figure 1) consists of four primary views: a data table, a model view, a control panel, and a model detail view.

Data Table Users can see the loaded dataset with every attribute in the data table view [see Figure 1(b)]. By default, the view shows the training data but users can toggle to view test and application dataset (test data required to validate models). We define application dataset as the final dataset without labels/output values. The view uses a standard spreadsheet metaphor, representing data items as rows and attributes as columns. Users can add, update, or delete training data (both rows and columns). The columns (attributes) of the loaded data shows three state toggle switches allowing users to emphasize, deemphasize, or discard an attribute using a slider

Table 1. Model Samples and Hyperparameters Used by the Learning Algorithm.

Model	Learning algorithm	Hyperparameters
M1	Linear regression	**fitIntercept** = 'true', **normalize** = 'false'
M2	Linear regression	**fitIntercept** = 'false', **normalize** = 'false'
M3	Linear regression	**fitIntercept** = 'true', **normalize** = 'true'
M4	Logistic regression	**fitIntercept** = 'true', **penalty** = 'l1', **dual** = 'true', **tol** = '10.55', **C** = '1.854', **maxIter** = '10'
M5	Logistic regression	**fitIntercept** = 'false', **penalty** = 'l2', **dual** = 'true', **tol** = '0.12', **C** = '100.854', **maxIter** = '20'
M6	Logistic regression	**fitIntercept** = 'true', **penalty** = 'l2', **dual** = 'false', **tol** = '-5.32', **C** = '55.4', **maxIter** = '50'
M7	Bayesian regression	**fitIntercept** = 'true', **normalize** = 'false', **alpha1** = '0.85', **alpha2** = '-5.32', **lamba1** = '55.4', **lambda2** = '50.524', **computeScore** = 'true'
M8	Bayesian regression	**fitIntercept** = 'false', **normalize** = 'true', **alpha1** = '20.85', **alpha2** = '-51.112', **lamba1** = '155.422', **lambda2** = '-30.24', **computeScore** = 'false'
M9	Bayesian regression	**fitIntercept** = 'true', **normalize** = 'true', **alpha1** = '8.65', **alpha2** = '1.102', **lamba1** = '-5.45', **lambda2** = '50.24', **computeScore** = 'true'
M9	Ensemble regressor	**compModels** = '['m1', 'm3', 'm4']', **maxFeatures** = '10', **maxSamples** = '200', **randomState** = '45', **numIter** = '30'
M10	Ensemble regressor	**compModels** = '['m4', 'm2', 'm8', 'm3']', **maxFeatures** = '500', **maxSamples** = '100', **randomState** = '45', **numIter** = '100'

to adjust the weights (shown in Figure 3). Likewise, users can discard a data instance or increase the relative importance of critical instances. Increased weighting tells the model that it should learn more from this instance than others, and emphasize the accuracy of these instances more. Users can also add domain expertise to the data via new attributes, if needed. Hovering over any row on the table triggers the system to recommend models from the model view to the user to inspect (i.e., models that correctly predicted the data instance). Recommended models are shown with a border stroke on the circular glyphs (representing a model) in the model view (see Figure 5).

Model View The model view shows each model as a circular glyph as shown in Figure 1(a). They are color coded by residual errors on the data loaded on the data table for the given regression problem. Yellow represents lower residual error, whereas dark blue represents higher residual error. The text on the glyphs describe the residual error value.

These circular glyphs are interactive. Hovering shows the model output details like the number of instances correctly predicted, the model's learning algorithm, etc. Users can inspect a model by clicking a visual glyph that adds a column to the data table, showing the predicted value (e.g., housing price on a housing dataset). The prediction column is next to the ground truth column, allowing users to compare how close the model's prediction is to the ground truth for each data instance. Inspecting a model also opens the model detail view (explained below). Furthermore, users can select multiple models to build model ensembles, or save a single model, to preserve the model across iterations. Users can also like a model, specifying that the next iteration should sample models similar to the one's the user liked (See Figure 4).

Model Detail View The model detail view shows up when the user inspects a model by clicking on it. It draws a line chart showing the predicted model output and ground truth data [shown as dots, shown in Figure 6(a)]. Visually it tells users how accurately the model fits the data. In the same view, users can see a residual bar chart, depicting the amount of error in prediction by the inspected model on both training and test

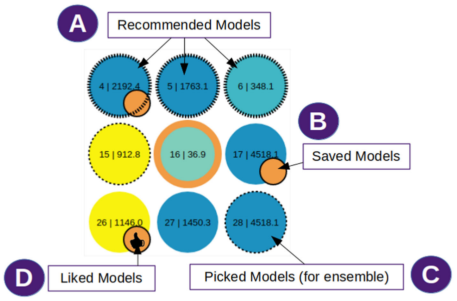


Figure 4. Shows the model view in detail. (A) Recommended models with strong border stroke. (B) Saved models. (C) Picked models for model ensembles. (D) Liked models.

sets (See Figure 7). The third view (bottom-left) shows correlation between two selected attributes from the data to the user. It helps users understand relationships between attributes in order to know which ones to emphasize, de-emphasize, or discard.

Control Panel The control panel [see Figure 1(c)] contains frequently-used operations, such as filtering data instances and attributes. Data instances can be filtered by both quantitative and categorical attributes. Similarly, models can be filtered by dragging the sliders that specify a threshold range of model accuracy, residual scores, or desired number of correctly predicted instances.

Technique

In this section, we will describe the underlying techniques applied to enable multimodel inspection and steering.

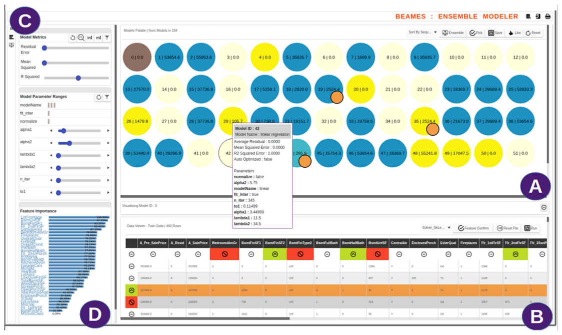


Figure 5. View showing Amy loads BEAMES to find 64 regression models. (A) Recommended models. (B) Amy hovers over a critical data instance. (C) Amy clicks model 29 and saves it. (D) Amy clicks on model 45 and inspects the output.

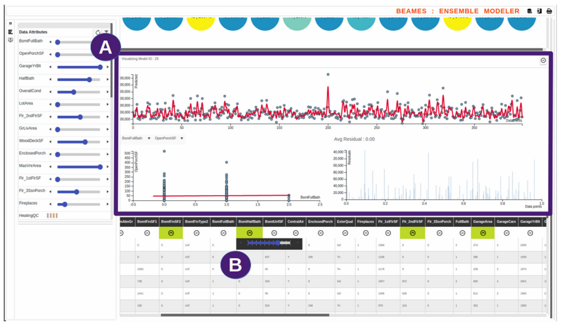


Figure 6. (A) Shows the model detail view. (B) Amy emphasizes features using the slider.

Data We define our full dataset as C (contains $N + K + B$ instances), which is then split in training, test and application dataset. Users train models on the training set D containing N samples, then validate on test set T containing K samples. When they find an acceptable model they export it or use it on application dataset H containing B samples. For notation descriptions, see Table 2.

Model Sampling We define a *model*, M_i as a function $f : \mathcal{X} \mapsto \mathcal{Y}$, mapping from the input space $\mathcal{X}$ to the prediction space $\mathcal{Y}$. Here, the prediction space is $\mathbb{R}$ and each model m_i is a regression model. For the modeling algorithms we used Scikit Learn's machine learning package.[23] Each model is sampled by combining a learning algorithm l_k from a set of J algorithms (hand picked by us for the regression task). Tested algorithms include Linear Regression, Logistic Regression, and Bayesian Regression. Each learning algorithm comes with their own set of hyperparameters λ_m. Examples of sampled models can be found in table 1. A sampled model is defined as

$$m_i \mapsto \text{Model}(l_k, [\lambda_{k1}, \lambda_{k2}, \lambda_{k3}, \ldots,]).$$

The system initiates with randomly sampled S models. We would like the sampling

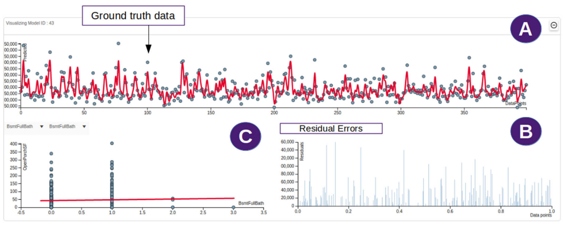


Figure 7. Amy exploring the model detail view showing low residual error for most of the instances.

distribution to be uniform across algorithms such that users can inspect a wide spectrum of model outputs for the given regression problem. For that reason, we initialize probability p_k to sample a learning algorithm l_k (for a model m_i) from J possible models as $1/J$.

Updating Training Data Users can load training data D on the data table. Every data sample is initially set to an equal weight of $\omega_i = 0.5$. However, users can interactively set weights on the samples between 0 and 1. 0 meaning to discard the data sample in training, whereas 1 is to place highest strength to the learning from the sample.

$$a_{ij} = \begin{cases} x & \text{user added value to the subset data instances} \\ 0 & \text{initialized value for the rest of the dataset} \end{cases} \quad (1)$$

where a_{ij} represents data at the ith column and the jth row.

User Driven Feature Engineering Similar to weighting of data instances, users can emphasize, discard or weight quantitative features. Using the UI toggle buttons [See Figure 3(a)], users can specify if they want to emphasize or discard a feature for model training. Discarding a feature removes it from the set **A**. Emphasizing a feature reveals a weight slider, which the user can toggle between -1 and 1. Setting a weight of -1 enforces the model to place higher emphasis on lower values of the attribute than others. As with instance weighting, all attributes default to have user weights of 0.5 before any interaction takes place.

The instance and attribute weights assigned by the user directly affect the computation of the y dependent variable. Loss for the models is a weighted least squares loss. Thus, the different regression models solve the following regression problem:

$$\min \sum_{i=0}^{N} \omega_i * (\hat{y}_i - y_i)^2$$

where

$$\hat{y}_i = b_0 + \sum_{i=0}^{M} b_i * x_i * w_i$$

where ω_i is the user defined weights for data instance i, b_0 is the intercept, b_i is the coefficients of the attributes learned by the model, and w_i are the user's attribute weights.

Table 2. Notation and Definitions Used to Define Our Technique.

Notation	Description
$C = D \cup T \cup H$	Full dataset, comprising of training, test and application dataset
$D = d_1, d_2, \ldots, d_N$	Training dataset of size N
$T = t_1, t_2, \ldots, t_K$	Test dataset of size K
$H = h_1, h_2, \ldots, h_B$	Application dataset of size B
$A = a_1, a_2, \ldots, a_P$	Set of attributes in the data
P	Cardinality of data attributes
$W = w_1, w_2, \ldots, w_N$	Set of attribute weights
$\Omega = \omega_1, \omega_2, \ldots \omega_P$	Set of training data weights
$M = m_1, m_2, \ldots, m_S$	Set of models of size S
$L = l_1, l_2, \ldots, l_J$	Set of J learning algorithms
$P = p_1, p_2, \ldots, p_J$	Probability distribution to pick J learning algorithm
$\Lambda_k = \lambda_{k,1}, \lambda_{k,2}, \ldots,$	Set of hyperparameters for model k

User Interactions User interactions in BEAMES are designed to update the underlying models via both interactive model steering and selection. This section describes these interactions, and details how the models interpret and update accordingly.

Save models If the user saves a model M_i, the system saves its learning algorithm L_i and the set of hyperparameter combination represented as $[\lambda_1, \lambda_2, \lambda_3, ..., \lambda_m]$. At each iteration of model sampling, BEAMES keeps saved models in the set of models shown to the user.

Like models If the user toggles the Like button in the interface on model m_d, then the probability p_d of the learning algorithm that produced that model is increased by a factor r_f. We randomly set the value of r_f by using a threshold ϵ. With trial and error, we found $\epsilon = 0.1$ showed promising results. Likewise, the hyperparameters λ_i of that algorithm are sampled from within a threshold region of the hyperparameters used in the liked model. This ensures that a large share of the new sampled models are from the neighboring regions of the models users liked.

However, the technique still ensures randomly sampling a smaller share of other models in the collection such that the user can still get an overview of model output from a wide array of choices. The process of liking models to regenerate a new set of models that are similar helps users perform interactive model selection.

Adjust data instance or attribute weight Users can adjust the weights of data instances and attributes by adjusting the respective sliders. As a result, all the available M models are retrained using N training instances with user specified features A_k where $A_k \subset A$ and user specified feature and data instance weights, W and Ω, respectively.

Export model Once users are satisfied with a model, they can simply export the single model. Additionally, model ensembles can be created and exported. For instance, users can select (by the pick interaction) G models to build a model ensemble. The system uses each component model m_j to build a model ensemble E. Our technique differs from theirs in which BEAMES provides users the flexibility to select component models to build an ensemble.

The BEAMES uses a bagging technique to sample from training data (sampling with replacement). However, data instances d_i that the users have increased the weights ω_i get higher probability to be sampled than other instances. This is to make sure models learn from these samples as users have intended, and the ensemble model emphasizes accuracy on the critical data instances d_i. Given the regression problem, the system finds the final predicted output by averaging the output received from component models. However, if any of the component models m_i are saved or liked by the user, then the output computed is weighted, and the weights of the saved or liked models are higher than that of the unsaved models. The final ensemble model E's output is the weighted average of the predictions of the models in the ensemble.

USAGE SCENARIO

We describe our tool BEAMES with a usage scenario, where a domain expert uses the system to perform data exploration and predict future housing prices. Amy is a real estate agent who reviews existing and new properties to analyze their market prices and potential change in the future due to changing conditions in the city. Amy has years of experience in her field including field knowledge of the city's various neighborhoods, upcoming city projects, infrastructure changes, and other city planning activities. She has a good grasp of the changing demographics of the city and the rising demand for housing. She usually explores data using tools like MS Excel, to decide on which properties to buy or sell. However, not being a data scientist, she is not conversant with complex modeling techniques, which can help her accurately predict property prices, property demands, or ratings in future.

Amy begins by importing two datasets into BEAMES [the dataset is available here (https://www.kaggle.com/c/house-prices-advanced-regression-techniques/data/)]. The first dataset is for the properties she knows the prices of at the current state of the market (called input data), and the second has properties she wants to predict future prices of (called the application dataset). The BEAMES splits the input data into training and test sets for model training and validation. The training data have over 750 samples with 36 attributes comprised of both categorical and quantitative types. It has a target attribute namely $SalePrice$, containing the property price of each house. Every row in the data is a property(a house) described by attributes, such as property size, fireplaces, year built, and number of bedrooms. The application dataset has more than 800 unlabeled samples.

After importing the data, BEAMES builds 64 regression models, each randomly sampled using a combination of learning algorithms (linear, logistic, ridge, and Bayesian regression) and hyperparameter values (alpha, lambda, tol, etc.). The list of sample models with hyperparameter values is shown in Table 1. As Amy is not formally trained in the specifics of the models, she begins her exploration by how well specific models predict property sale prices. In the model view (Figure 5), she sees the collection of models as circular glyphs color coded by their residual error scores. Yellow represents better models with lower residual error, whereas blue glyphs are models with high residual errors. On the bottom in the data table, Amy sees the training data in a tabular format. Browsing the colors of the circular glyphs (representing models), Amy decides to start inspecting a few further as they have lower error values.

Clicking the circle, Amy sees that the training data has a new column representing predicted price. She sees that the predictions are quite close to some of the actual prices in the data. She clicks on model 45 [see Figure 5(d)], as it has accurately predicted more than 300 entries on her training data and has some errors on the rest of the entries from the training data. She sees that the residual error of the current model is more than 500. Next, she notices a few models with residual error score close to 100 (colored yellow). She clicks one of the lower scored models [model id 29, as seen in Figure 5(c)] and finds that most of the data instances are predicted accurately. However, to double-check if some of the known data instances were correctly predicted, she uses the filter panel on the left. She sees the currently selected model (with a low residual error of 105.7) did not correctly predict most of the these critical data instances.

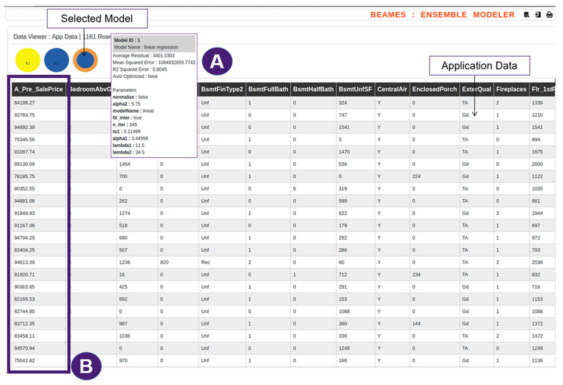
By hovering over these critical data instances, [see Figure 5(b)] the system shows models that Amy should inspect, as they made correct predictions on those instances [See Figure 5(a)]. Amy reviews a few of the suggested models. She finds that the recommended models performed better for the critical instances, though they had higher overall residual errors. Next, Amy toggles the three state toggle button on these data instances to emphasize them using the slider [shown in Figure 6(b)]. In addition, Amy thinks the property price should be most strongly defined by the $numberofbedrooms$, $garageArea$, and the $2ndfloorarea$ attributes. She again uses the slider to increase their weight, while reducing the weight on $frontPorchSize$ and $DrivewayQuality$. Next she presses the build new model button for BEAMES to recompute all the models.

The BEAMES updates the model view with newly computed models. Amy sees the color encoding changed for the collection of models, as new models have different residual error output. Amy quickly looks over the collection to find that many models have scores close to 0, meaning that they have high performance on training data. However, still being interested in her critical data instances, she hovers over the rows to see recommended models from the model view. She clicks on few of the recommended models and finds two of them performed quite well, as it correctly predicted 7 out of the 8 critical instances. Clicking on these models, Amy finds the prediction on the the test data is bit off. For example, property id 104 shows a predicted price of 141 345, whereas the true price is 99 322.

Confused to find relatively poor performance on the test set, Amy uses the control panel to see the importance of the features in the horizontal bar chart. She sees the relatively strong weight on the $numberofbedrooms$ attribute (as she intended previously). She adds few other relevant attributes and ups the weight factor for those, i.e, $overallPropertyRating$, $numberOfFloors$, and $distanceToTransit$. Next, she discards a few training data samples thinking those properties are not relevant anymore. She saves two models that she found have good potential and likes a few others based on their performance. In addition she picks a few models to create an ensemble from these component models and generates more models.

Amy browses the newly computed models. She sees brown colored circular glyphs [See Figure 1 (c)], representing an ensemble model build from the models she picked. She clicks on it and finds that it shows a very accurate prediction on the training data. Amy confirms the same on the model detail view, as the line fits the set of points (representing actual ground truth values), as shown in Figure 7. Similarly, she sees almost 0 residual error from the bar chart shown in Figure 7.

At this point, she also uploads the test dataset in the data table. She hovers over the rows, to see system recommended models (visually represented with an outer stroke line). She is happy to see that the models recommended include the ensemble model and the one she saved. She clicks on the recommended models and finds that the results improved. The predictions were very close and in some instances were almost the same as the ground truth. At this point, Amy is happy with the models and saves a few (including the model ensemble). She loads the application dataset. The interface changes to show a full-screen spreadsheet, with a horizontal scrollable



Data Viewer : App Data | 1161 Rows

Model ID : 1
Model Name : linear regression
Average Residual : 3401.0303
Mean Squared Error : 1084832659.7743
R2 Squared Error : 0.8045
Auto Optimized : false

Parameters
normalize : false
alpha2 : 5.75
modelName : linear
fit_inter : true
n_iter : 345
to1 : 0.11499
alpha1 : 3.44999
lambda1 : 11.5
lambda2 : 34.5

A_Pre_SalePrice	BedroomAbvG			BsmtFinType2	BsmtFullBath	BsmtHalfBath	BsmtUnfSF	CentralAir	EnclosedPorch	ExterQual	Fireplaces	Flr_1stF
84188.27				Unf	1	0	324	Y	0	TA	2	1336
92783.75				Unf	0	0	747	Y	0	Gd	1	1210
94892.39				Unf	0	0	1541	Y	0	Gd	1	1541
75346.56				Unf	1	0	0	Y	0	TA	0	894
91067.74				Unf	0	0	1470	Y	0	TA	1	1675
89130.09		1464	0	Unf	1	0	536	Y	0	Gd	0	2000
78185.75		700	0	Unf	1	0	0	Y	224	Gd	1	1122
80352.55		0	0	Unf	0	0	319	Y	0	TA	0	1035
94881.06		262	0	Unf	0	0	599	Y	0	TA	0	861
91848.93		1274	0	Unf	1	0	622	Y	0	Gd	3	1944
91167.06		518	0	Unf	0	0	179	Y	0	TA	1	697
94704.29		680	0	Unf	1	0	292	Y	0	TA	1	972
83404.25		507	0	Unf	1	0	286	Y	0	TA	1	793
94613.39		1236	820	Rec	2	0	80	Y	0	TA	2	2036
81920.71		16	0	Unf	0	1	712	Y	234	TA	1	832
90363.65		425	0	Unf	1	0	291	Y	0	Gd	1	716
82169.53		692	0	Unf	1	0	153	Y	0	Gd	1	1153
82744.85		0	0	Unf	0	0	1088	Y	0	Gd	1	1088
83712.35		987	0	Unf	1	0	360	Y	144	Gd	1	1372
83458.11		1036	0	Unf	1	0	336	Y	0	TA	2	1472
94570.94		0	0	Unf	0	0	1249	Y	0	TA	0	1249
75641.82		970	0	Unf	1	0	166	Y	0	Gd	1	1136

Figure 8. Amy loads the application dataset to apply the saved models and see predicted output. (A) Horizontal panel storing models saved by Amy. (B) Predicted output ("sale price") when a model is selected.

panel on top listing the saved models [Figure 8(a)]. She clicks on the saved model glyphs to see predictions on the application dataset. As she clicks, BEAMES adds a prediction column to the application dataset [Figure 8(b)].

At this point, Amy has models (with the help of BEAMES) to help her predict house prices, each emphasizing different characteristics of the data.

DISCUSSION

Spectrum of automatic, semiautomatic, and manual model selection. The work described in this paper is a semiautomatic technique for exploring multiple models and their respective parameterizations. This technique takes a different approach from automatic hyperparameter tuning model selection, as adopted by some of the existing systems like AutoWeka and Hyperopt. Our technique differs from manual model-building techniques where users are required to specify most (if not all) of the model parameters. Within this model selection spectrum, we explore how to bring domain experts in the loop of model building by performing model validation and selection based on not only objective metrics (i.e., residual errors) but also data instance validation (i.e., "I like model A because it predicts instances 1,5,10 correctly, though it has relatively higher residual error."). We call this approach semiautomatic, because some parts of model sampling and model building are still automatic.

Model output inspection and model interpretability. Model interpretability and comprehensibility is discussed in the work of Gleicher.[24] Gleicher discusses model interpretability as a means to understand the learning of the model. However, in our work, we are aiming to help users understand a model by inspecting its output and steering it toward their own sense of the importance of familiar and critical data instances. Our visual technique helps users understand the model output against ground truth, without having to interpret the internal complexities of the models. While our work begins to explore this space, there is more work to understand how users understand, interpret, and trust models by observing outputs.

LIMITATION AND FUTURE WORK

Model space sampling. The number of models that can be sampled for a given problem/task and dataset is large (potentially infinitely so). Given the plethora of options for learning algorithms, each with a distinct group of hyperparameters, which can take values within a set domain range, the size of the model candidate space grows rapidly. Any multimodel optimization technique searching from such a large model space can result in many model to choose from. In BEAMES, we set an upper limit on the number of model options the system randomly samples to initiate the loop of model inspection and steering. Another feature assisting the user is the technique to recommend models to inspect. The recommendations are driven by how well the models perform on data instances, the user care about. However, given the iterative nature of the technique, it might lead to a substantial amount of user interaction until they find an acceptable model, leading to fatigue or frustration. In the future, we are interested in exploring additional forms of guidance for multimodel steering.

Model overfitting. Externalizing a domain experts knowledge to train a model can often lead to the problem of overfitting. Though BEAMES has a provision of using a test dataset to train model, we understand that the overuse of the testing data might lead to an overfitted model. The current interface of BEAMES does not have explicit functionality to avoid model overfitting. In future work, we look forward to overcome model overfitting while still using expert knowledge in the model training process.

Model comparison. BEAMES in its current state can help users compare models by inspecting their outputs. Though helpful, there are open research questions about how best to compare multiple models directly. However, these are not fully addressed in the current design.

CONCLUSION

In this paper, we described a technique for interactive multimodel steering and inspection. The technique emphasizes the importance of moving beyond existing, single-model steering techniques where the initial model choice greatly impacts the quality of the results. Instead, our technique steers and samples from multiple model types, learning algorithms, and hyperparameters. Our visual analytics prototype, BEAMES, allows people to specify interest in models, data instances, and attributes to iteratively build, combine, and select models to perform prediction tasks. We describe our technique and demonstrate the use of BEAMES through a use case scenario, highlighting the model steering and inspection process resulting from user interactions. Finally, we discuss broader challenges in the area of multimodel steering and illuminate valuable areas for future work.

ACKNOWLEDGMENTS

This work was supported by the Defense Advanced Research Projects Agency under Grant FA8750-17-2-0107. The views and conclusions contained in this document are those of the authors and should not be interpreted as representing the official policies, either expressed or implied, of the U.S. Government.

REFERENCES

1. P. Daee, T. Peltola, A. Vehtari, and S. Kaski, "User modelling for avoiding overfitting in interactive knowledge elicitation for prediction," in *Proc. 23rd Int. Conf. Intell. User Interfaces*, 2018, pp. 305–310. [Online]. Available: http://doi.acm.org/10.1145/3172944.3172989
2. Q. Yang, J. Suh, N.-C. Chen, and G. Ramos, "Grounding interactive machine learning tool design in how non-experts actually build models," in *Proc. Des. Interactive Syst. Conf.*, 2018, pp. 573–584. [Online]. Available: http://doi.acm.org/10.1145/3196709.3196729
3. H. Kim, J. Choo, H. Park, and A. Endert, "InterAxis: Steering scatterplot axes via observation-level interaction," *IEEE Trans. Vis. Comput. Graph.*, vol. 22, no. 1, pp. 131–140, Jan. 2016. [Online]. Available: doi.ieeecomputersociety.org/10.1109/TVCG.2015.2467615
4. E. T. Brown, J. Liu, C. E. Brodley, and R. Chang, "Dis-function: Learning distance functions interactively," in *Proc. IEEE Conf. Vis. Analytics Sci. Technol.*, Oct. 2012, pp. 83–92.
5. B. C. Kwon, H. Kim, E. Wall, J. Choo, H. Park, and A. Endert, "AxiSketcher: Interactive nonlinear axis mapping of visualizations through user drawings," *IEEE Trans. Vis. Comput. Graph.*, vol. 23, no. 1, pp. 221–230, Jan. 2017. [Online]. Available: doi.ieeecomputersociety.org/10.1109/TVCG.2016.2598446
6. E. Wall, S. Das, R. Chawla, B. Kalidindi, E. T. Brown, and A. Endert, "Podium: Ranking data using mixed-initiative visual analytics," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 288–297, Jan. 2018. [Online]. Available: doi.ieeecomputersociety.org/10.1109/TVCG.2017.2745078
7. A. Endert, C. Han, D. Maiti, L. House, S. C. Leman, and C. North, "Observation-level interaction with statistical models for visual analytics," in *Proc. IEEE Conf. Vis. Analytics Sci. Technol.*, 2011, pp. 121–130.
8. D. H. Jeong, C. Ziemkiewicz, B. Fisher, W. Ribarsky, and R. Chang, "iPCA: An interactive system for PCA-based visual analytics," *Comput. Graph. Forum*, vol. 28, no. 3, pp. 767–774, 2009.
9. R. van Liere, J. D. Mulder, and J. J. van Wijk, "Computational steering," *Future Gener. Comput. Syst.*, vol. 12, no. 5, pp. 441–450, 1997. [Online]. Available: http://www.sciencedirect.com/science/article/pii/S0167739X96000295
10. A. Endert, P. Fiaux, and C. North, "Semantic interaction for visual text analytics," in *Proc. SIGCHI Conf. Human Factors Comput. Syst.*, 2012, pp. 473–482. [Online]. Available: http://doi.acm.org/10.1145/2207676.2207741
11. L. Bradel, C. North, L. House, and S. Leman, "Multi-model semantic interaction for text analytics," in *Proc. IEEE Conf. Vis. Analytics Sci. Technol.*, Oct. 2014, pp. 163–172.
12. B. Schneider, D. Jäckle, F. Stoffel, A. Diehl, J. Fuchs, and D. A. Keim, "Visual integration of data and model space in ensemble learning," in *Symp. Visualization Data Science (VDS)*, 2017, Phoenix, AZ, USA.
13. K. Patel, S. Drucker, J. Fogarty, A. Kapoor, and D. Tan, "Using multiple models to understand data." in *Proc. 22nd Int. Joint Conf. Artif. Intell.*, Jul. 2011,

pp. 1723–1728. [Online]. Available: https://www.microsoft.com/en-us/research/publication/using-multiple-mod els-understand-data/

14. H. Piringer, W. Berger, and J. Krasser, "Hypermoval: Interactive visual validation of regression models for real-time simulation," *Comput. Graph. Forum*, vol. 29, pp. 983–992, 2010.
15. B. C. Kwon *et al.*, "Clustervision: Visual supervision of unsupervised clustering," *IEEE Trans. Vis. Comput. Graph.*, vol. 24, no. 1, pp. 142–151, Jan. 2018.
16. K. Potter *et al.*, "Ensemble-Vis: A framework for the statistical visualization of ensemble data," in *Proc. IEEE Int. Conf. Data Mining Workshops*, Dec. 2009, pp. 233–240.
17. C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *Proc. 19th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, 2013, pp. 847–855.
18. J. Krause, A. Perer, and K. Ng, "Interacting with predictions: Visual inspection of black-box machine learning models," in *Proc. CHI Conf. Human Factors Comput. Syst.*, 2016, pp. 5686–5697. [Online]. Available: http://doi.acm.org/10.1145/2858036.2858529
19. J. Zhang, Y. Wang, P. Molino, L. Li, and D. S. Ebert, "Manifold: A model-agnostic framework for interpretation and diagnosis of machine learning models," *IEEE Trans. Vis. Comput. Graph.*, vol. 25, no. 1, pp. 364–373, Jan. 2019.
20. Automl 2014 Workshop @ ICML. 2018. Accessed May 29, 2019. [Online]. Available: https://sites.google.com/site/automl2018icml/
21. H. Jin, Q. Song, and X. Hu, "Auto-Keras: Efficient neural architecture search with network morphism," Jun. 27, 2018, *arXiv cs.LG/1806.10282*.
22. M. Feurer, A. Klein, K. Eggensperger, J. Springenberg, M. Blum, and F. Hutter, "Efficient and robust automated machine learning," in *Proc. Adv. Neural Inf. Process. Syst.*, 2015, pp. 2962–2970. [Online]. Available: http://papers.nips.cc/paper/5872-efficient-and-robust-automated-machine -learning.pdf
23. L. Buitinck *et al.*, "API design for machine learning software: Experiences from the scikit-learn project," in *Proc. ECML PKDD Workshop, Lang. Data Mining Mach. Learn.*, 2013, pp. 108–122.
24. M. Gleicher, "A framework for considering comprehensibility in modeling," *Big Data*, vol. 4, no. 2, pp. 75–88, Jun. 2016. [Online]. Available: http://graphics.cs.wisc.edu/Papers/2016/Gle16

Subhajit Das is currently working toward the Ph.D. degree in computer science from the School of Interactive Computing, Georgia Institute of Technology, Atlanta, GA, USA. His research interests include interactive machine learning, model optimization/selection, and human-in-the-loop based visual analytic systems. He has previously explored design and development of intelligent interactive systems and prototype testing with industry partners including Autodesk, Microsoft, and GTRI. Contact him at das@gatech.edu.

Dylan Cashman is currently a Ph.D. candidate with the Department of Computer Science, Tufts University, Somerville, MA, USA. His research interests include human-centered model selection, interpretability in machine learning, and visual encodings for deep learning architectures. He received the M.Sc. degree in computer science from Tufts University and the Sc.B. degree in mathematics from Brown University, Providence, RI, USA. He is a Student Member of the ACM. Contact him at dylan.cashman@tufts.edu.

Remco Chang is currently an Assistant Professor with the Computer Science Department, Tufts University, Somerville, MA, USA. His current research interests include visual analytics, information visualization, and human–computer interactions. His research has been funded by NSF, DHS, MIT Lincoln Lab, and Draper. He received the B.S. degree in computer science and economics from Johns Hopkins University, Baltimore, MD, USA, in 1997, the M.Sc. degree from Brown University, Providence, RI, USA, in 2000, and the Ph.D. degree in computer science from the University of North Carolina at Charlotte (UNC), Charlotte, NC, USA, in 2009. Prior to his Ph.D., he worked for Boeing developing real-time flight tracking and visualization software, followed by a position with UNC Charlotte as a Research Scientist. Contact him at remco@cs.tufts.edu.

Alex Endert is currently an Assistant Professor with the School of Interactive Computing, Georgia Tech., Atlanta, GA, USA. He directs the Visual Analytics Lab, where he and his students explore novel user interaction techniques for visual analytics. His lab often applies these fundamental advances to domains including text analysis, intelligence analysis, cyber security, decision making, and others. He received the Ph.D. degree in computer science from Virginia Tech, Blacksburg, VA , USA, in 2012. Contact him at endert@gatech.edu.