Interactive Clustering with a High-Performance ML Toolkit

Biye Jiang
Computer Science Division
UC Berkeley
Berkeley, CA 94720
bjiang@cs.berkeley.edu

John Canny
Computer Science Division
UC Berkeley
Berkeley, CA 94720
jfc@cs.berkeley.edu

ABSTRACT
Clustering is a class of machine learning algorithms which has important applications in many different fields. Users often use clustering to find hidden structures from data for those domain specific problems. However, evaluating clustering results is always a hard problem. In many and perhaps most of these applications, users need to trade off competing goals and encode prior knowledge into the model to define what is the best result. The learning algorithm however has evolved around the optimization of a single, usually narrowly-defined criterion, which may not obtain satisfactory results. In most cases, an expert makes trade-offs between different criteria which requires high-level (human) intelligence. This motivates us to provide interactive customization and optimization so that the expert can incorporate secondary criteria into the model-generation process in an interactive way.

In this demo paper we will demonstrate the techniques we developed to do customized and interactive model optimization for clustering algorithms. The keys to the approach are (i) high-performance training so that non-trivial models can be trained in real-time (using roofline design and GPU hardware), (ii) a machine learning architecture which is modular, and supports primary and secondary loss functions, and (iii) highly-interactive visualization tools that support dynamic creation of visualizations and controls to match the bespoke criteria being optimized.

Keywords
Interactive, Machine learning, Clustering, GPU

1. INTRODUCTION
Machine Learning is now at the center of data analysis in many fields across the sciences, business, health care and other realms. Among those ML algorithms, clustering is a class of unsupervised learning algorithms which is often used to find hidden structures of the data. In contrast to supervised learning which always uses accuracy to measure model quality, in general it is hard to evaluate clustering results. In practice users often need to trade off competing goals. Latent variable models such as K-Means clustering, or LDA[3] or NMF[18] find latent factors which maximize the likelihood of the observed data, but which may have secondary desiderata such as uniform cluster size, independence of factors, or coherence of topics.

The learning algorithms have evolved around the optimization of a single, usually narrowly-defined criterion, but users often find it hard to represent their criteria as a single objective function. The lack of evaluation methods also make it hard for analyzing algorithm behavior as well as debugging. In many cases, using clustering algorithms requires high-level (human) intelligence to make trade-offs between these criteria and examine the results manually.

Because ML models today are not flexible enough to incorporate all these criteria, secondary constraints are often applied after model training (by overriding the model’s choices) in a way that is inevitably sub-optimal. Furthermore, the effects of downstream interventions require live testing to quantify. By contrast, interactive customization and optimization allows the analysts to incorporate secondary constraints into the model-generation process in an interactive way. There are several benefits to this:
• Models can be fully optimized given a suitable mixture of the criteria.
• Families of models can be trained to deal with variability in the application context.
• Analysts can explore the effects of particular trade-offs instantly, without waiting for a live test.
• Through this exploration, an analyst can gain intuition for the effects of various criteria, and make better trade-offs in the long run.

In this paper we develop the techniques to do customized and interactive model optimization, and demonstrate the approach on several examples. The keys to the approach are (i) high-performance training so that non-trivial models can be trained in real-time (using roofline design and GPU hardware), (ii) a machine learning architecture which is modular, and support primary and secondary loss functions, and (iii) highly-interactive visualization tools that support dynamic creation of visualizations and controls to match the bespoke criteria being optimized.

KDD 2015 Workshop on Interactive Data Exploration and Analytics (IDEA'15) August 10th, 2015, Sydney, Australia.
Copyright is held by the owner/author(s)
2. RELATED WORK

2.1 Interactive clustering

Interactive clustering is an active area. Since clustering is widely used to simplify the interpretation of large datasets, and since the natural metrics for a new domain may be difficult to articulate, interactive exploration [8, 24, 29] is a natural and powerful approach. In [8, 24] authors used visualization to rapidly explore the results of a clustering algorithm, and these approaches have become important tools in computational biology [8]. AverageExplorer [29] allow users to explore and summarize large collection of images by interactively posing constraints.

Recently, there has been much interest in using visualization to support the refinement of topic models [26, 12, 7]. Since the latent topics extracted by the algorithm are not always semantically meaningful [22, 6], different constrained topic models [20, 2, 21] have been developed. Systems like [26, 12] also allow users to iteratively refine the model based on their preference. However, those models always require solving a complicated optimization problem with some very specific constraint. And few systems have demonstrated real-time interaction with large scale dataset.

2.2 Interactive model refinement

In the context of supervised learning, one early influential paper on interactive machine learning was Fail and Olsen’s paper [9], which describes partially-supervised learning with a user supplying some (sparse) labelled data to help an ML algorithm label the rest. A number of other works have followed this route, by focusing on manipulation of the training data rather than internals of a particular algorithm. Other work focused on human-assisted feature selection (rather than algorithm training) [23]. Amershi et al. [1] provides a detailed summary of the work in this area. Much of these works attempt to improve only the accuracy of a machine learning problem by adding a human in the loop, which is quite different from our work. Perhaps the closest to ours is [13] which integrates a human-assisted optimization strategy with the design of multi-class classifiers. But in this paper we focus on clustering and the optimization algorithms are different.

2.3 Large scale machine learning system

There has been a great deal of work recently on tools for Big Data. But much of these works emphasize scalability on clusters [14, 25, 10] without applying single-node acceleration (CPU and GPU-specific acceleration libraries). Those systems are typically optimized for scalability rather than latency, which is more important for interactive modeling.

3. SYSTEM DESIGN

3.1 BIDMach: high-performance, customized machine learning

The first key to interactive, customized machine learning is an architecture which supports it. BIDMach [5] is a new machine learning toolkit which has demonstrated extremely high performance with modest hardware (single computers with GPUs), and which has the modular design shown in Fig 1. BIDMach use minibatch updates, typically many per second, so that models are being updated continuously.

This is a good match to interactive modeling, since the effects of analysts actions will be seen quickly. Rather than a single model class, models comprise first a primary model (which typically outputs the model loss on a minibatch and a derivative or other update for it). Next an optimizer is responsible for updating the model given gradients. Several are available including simple SGD, ADAGRAD, and Preconditioned CG. Finally, mixins represent secondary constraints or likelihoods. Gradient-based primary models and mixins are combined with a weighted sum. In our interactive context, these weights are set interactively.

![Figure 1: BIDMach’s Architecture](image)

Beyond the architecture of Fig 1, BIDMach has two layers. A general-purpose matrix layer called BIDMat, and BIDMach which includes the machine learning classes from figure 1. This organization shortens development time by providing high-level primitives for writing learning algorithms, and also allows us to leverage recent gains in the performance of GPU hardware. BIDMat is completely agnostic about matrix type: both CPU and GPU matrices (and sparse or dense and single or double precision) can be used and code is written generically.

BIDMach contrasts with most high-performance machine learning systems [14, 25, 10] in its emphasis on optimizing single-machine performance first, and then scale-up if needed. BIDMach uses roofline design to optimize computational kernels toward hardware limits. On a large collection of benchmarks it has proved to be typically two orders of magnitude faster than other single-machine toolkits (when BIDMach is used with a GPU), and one to two orders of magnitude faster than cluster toolkits running on 10-100 nodes. Part of the difference is due to BIDMach’s complete suite of GPU primitives. Almost all computation is done on the GPU; CPU/GPU transfers are minimized, and custom kernels give close to the theoretically optimal GPU performance. GPUs typically achieve an order-of-magnitude speedup in dense matrix operations vs. mid-range CPUs. Less well known is their advantage in main memory speed, which (at 300 GB/s) is nearly an order-of-magnitude faster than recent quad-channel CPUs (at around 40 GB/s). This memory speed gap also gives GPUs a similar advantage for sparse matrix operations which are central to most real-world ML applications. These differences explain one order of magnitude of the performance gap that we observe with BIDMach. The balance is due to the fact that most other systems are not close to their (CPU) rooflines. Because of this BIDMach
High performance is very important for interactivity. BIDMach has reduced the running time of many non-trivial ML tasks from hours to minutes. And even for models that take minutes to train fully, the effects of parameter changes are typically visible in seconds. We will see this in the examples later.

3.2 Client-server architecture
We use a client-server architecture with 3 components as shown in Fig 2: a computing (BIDMach) engine, a web server, and a web based front end. BIDMach is implemented in the Scala language which supports concurrency with high level "actor" primitives. Another thread runs in the same Scala runtime, communicating with the web server. This thread receives parameter updates from the web server, and updates the corresponding model training parameters. As the model is trained, primary and mixin cost functions are evaluated on minibatches, providing regular updates which are passed to the web server.

![Visualization Architecture](image)

**Figure 2: Visualization Architecture**

In the client side, we implement a web based interface which uses D3.js[4] for data visualization. D3 is widely used, has very powerful graphics elements, and good support for animation. As a browser-based system, it runs transparently with a local or remote server. We will discuss the interface design in detail in the next section.

The communication between client and server is bi-directional, with both client and server initiating transfers. We therefore use WebSockets instead of e.g. a one-way RESTful web service. For simplicity and extensibility, we use JSON as the over-the-wire exchange format.

3.3 Secondary criteria as Mixins
Model customization is useful for both supervised and unsupervised problems. Unsupervised learning involves a certain amount of arbitrariness in the criteria for the “best” latent state. Therefore regularization is widely used as a secondary constraint on the primary objective [20, 2, 21, 26].

In a bit more detail, clustering algorithms like KMeans usually use the measure of model/centroid similarity, and may or may not use intra-model coherence measures or inter-cluster distance. Indeed, unsupervised learning models are often evaluated using a variety of criteria that are much more complex than the criteria used to derive the learning algorithms [22, 6]. The same holds true for topic models such as LDA, NMF and Word2Vec, and for collaborative filtering.

This is a paradox. Clearly one should get better scores for these criteria if they were directly optimized as part of training. Beyond these standard criteria, there are many others that are commonly used in the applications of machine learning. Historically it has probably been too difficult to optimize these criteria (the criteria may be expensive to evaluate, or non-locally computable).

On the other hand computing power is abundantly available now, especially in graphics processors. The bottleneck is often moving data rather than computing on it. Thus it is often practical to evaluate multiple, relatively complex criteria as part of optimization.

Combining these approaches, we can deal with a variety of secondary or “mixin” criteria as part of the learning process. In our present implementation, we use a linear combination of cost functions for primary and secondary criteria:

\[
\arg \min_x f(x, d) + \sum \lambda_i g_i(x)
\]

Where \(x\) is the model parameters, \(d\) is data, \(f\) is the primary cost function and \(g_i\) are the user-defined Mixin functions. The weights \(\lambda_i\) are “controls” that are dynamically adjusted by the analyst as part of training. For the primary criterion \(f\) and each secondary criterion \(g_i\) there should be at least one dynamic graphic that captures changes in that criterion in an intuitive way. The analyst watches these as each of the controls are adjusted to monitor the tradeoff between them.

4. INTERFACE DESIGN
In this section, we will describe the visual interface of our interactive machine learning system.

4.1 Visual dashboard
We use a dashboard approach where user can customize their own visualizations. As shown in Fig 3, the left side of the interface contains the menus and control sliders. From the menus, a user can select the metrics and controls for the modeling task. A corresponding control or metric visualization is then added to the dashboard, which can then be dragged, dropped and resized. There is at least one corresponding performance indicator for each control parameter, and more than one can be added to the dashboard. The details of each visualization component will be described next.

4.2 Visualizing the model
Directly visualizing the model provides a nice summarization for the dataset and users can gain a general understanding about the behavior of the algorithms. It can also help identify obvious errors and verify assumptions or intuitions. While there are different types of data and algorithm, the visualizations are necessarily model-specific, and should provide a natural interpretation of the model directly. For image data, the cluster centers can usually be visualized as in Fig 4a. For dictionary learning algorithms like NMF[18], the learned image dictionary can also be directly visualized as in Fig 4b, which is a much sparser representation. For more general matrix data, a simple direct visualization of element weights can work well. This was the approach taken
in the “termite” system [7] and we use it also for our topic model visualization.

Figure 3: Dashboard for KMeans using MNIST dataset

As we are displaying in real-time an evolving model, it is important to have a smooth and consistent word order after each update. We therefore only use the original topic weight to get the order of the words. In order to support a detailed zoom in for each topic, we also support ranking within a topic. This feature will be triggered when user mouse over the topic title, as shown in Fig 4d.

4.3 Continuous visualization of model quality

As described in 3.3, we are optimizing an additive function which consists of a main loss term as well as severalMixin terms. The value of the cost function can reflect how the model behaves under each criteria. As the engine keeps computing the update, we visualize those metrics as streaming data, as shown in Fig 5. Visualizing the main loss function is very important when we change the control parameters. It will reflect how algorithm responses to the user control and whether the tradeoff forMixin functions may affect the general model performance.

As discussed earlier, the main loss function is computed on each minibatch, and is a single scalar. We use a simple, dynamic curve plot to display its state. This style of plot is easy to read and understand. With the parameters fixed, one normally sees a rapid initial increase in likelihood, followed by a slow increase on a plateau as in fig 5.
### 4.4 Visualizing other performance indicators for Clustering

As mentioned above, the evaluation of unsupervised clustering algorithm is hard in general. Therefore showing multiple aspects of the clustering results at the same time would help users better interpret the model.

#### 4.4.1 Cluster size distribution

To examine the cluster size balance, we are interested in the distribution of cluster sizes. The natural visualization for this kind of data is a histogram or kernel density plot. Fig 6a shows the size distribution for the clusters of digits on the MNIST dataset.

#### 4.4.2 Silhouette graph

Another useful metric is the widely used silhouette graph (Fig 6b) for evaluating clustering results. The silhouette score is calculated for each data point \( x_j \) as:

\[
s_j = \frac{b_j - a_j}{\max(a_j, b_j)}
\]

Where \( a_j \) is the average distance between \( x_j \) and all other data points in the same cluster, \( b_j \) is the lowest average distance of \( x_j \) to any other cluster. It could be seen that \(-1 \leq s_j \leq 1\) and larger value indicates better clustering results.

#### 4.4.3 Recovered images from NMF

For the Non-negative Matrix Factorization algorithm[18], it is easy to recover the approximated matrix by multiplying the two factorized matrices. Although the quality of the recovered matrix can be measured using L2-distance between the original matrix and the approximated one, directly showing the recovered result for image data (Fig 6c) can provide more details about what kind of information is being lost or preserved.

### 4.5 Slider controls

Along with the visual interface, we also provide several kinds of control including weights for Mixin, learning rate, etc. We also provide temperature control by changing the sample variance of the Gibbs sampler using SAME sampling[27]. So far these are all continuous scalars, and are all implemented as slider widgets. When the user select one of the controls from the menu, a labeled slider widget is created on the dashboard. The user drags and resizes this widget in an appropriate area of their dashboard. The selection of controls is currently independent of the selection of related metrics, and they are placed separately as well. In future we will explore intuitive ways of linking them (e.g. highlighting related metrics when a control is selected and vice-versa, or moving them as a group).

### 5. USE CASES

In this section, we will demonstrate several representative use cases for our system.

#### 5.1 Non-negative Matrix Factorization

Our first demo is to monitor model update of the NMF algorithm in real-time. Although in this demo we don’t have interactive control to the algorithm, we will still demonstrate how we can gain insight by viewing the learning process.

The NMF algorithm is trying to find low rank representation of the data matrix \( V \) by factorizing it into a product of two matrices \( W, H \) with lower dimension. This is done by solving the optimization problem using multiplicative updates described in[18]:

\[
\arg\min_{W,H} \| V - WH \|_2
\]

This technique is very powerful in practice and can be used to find clusters of local patches for image data, or topics for text data. In our demo, we apply the NMF algorithm on the CIFAR-100 dataset[15]. The CIFAR dataset contains 50000 32*32 tiny RGB images which is stored in a 3072*50000 matrix. As we set the number of factors to be 256, the data matrix will be factorized into a 3072*256 matrix which is the dictionary for images and a 256*50000 matrix which contains the weights for each original image. Therefore each
original image can be approximated as a linear combination of the 256 template images.

We train the model using a NVIDIA GTX-690 GPU with 2GB graphics memory, which costs less than a second for one iteration on the entire dataset, achieving around 150 GFlops. Such speed allows us to monitor real-time update as the model converges. We visualize the first 100 template images as well as the first 100 recovered images to help interpret what information is being extracted from the dataset. We also visualize the L2-loss \( \|V - WH\|_2 \) as a quantitative measurement.

The four representative stages of the training procedure is shown in Fig 7. Initially, the weight matrix are set to all one. Therefore a reasonable local optimal is to find some averaging images as the dictionary, and the recovered results all look similar, as shown in Fig 7a. Later in the procedure, the algorithm begins to recover background information as shown in Fig 7b. The sparseness of NMF model can also be seen since most pixels in the dictionary are black. This is consistent with [17] that the NMF algorithm always learns parts of the images. More texture and edge information will then be learned in the third stage, shown in Fig 7c. Some of the recovered images gradually become recognizable. Also,

the template images clearly fall into two categories. One captures background information and the other represents tiny local patterns. Finally, in the last stage (Fig 7d), more details are being recovered and the L2-loss is about to converge. User now can trade off the model quality and training time based on their requirement.

Comparing to the L2-loss metric, directly visualizing the model provides much more information than a single scalar. Users can gain more insight about the algorithm behavior by viewing the whole training process, which could help them make better decisions in the future.

5.2 KMeans

Our second demo is KMeans. This time we will have some direct control to the model which can demonstrate the full power of interactive clustering.

5.2.1 Evaluation and implementation for KMeans

For KMeans, the primary loss function is inertia: the sum of squared distances from points to their centroid. And the algorithm is straightforward: iteratively assigns data points to clusters and updates the cluster centroids using the mean of data.
However, as mentioned previously, the evaluation and tuning of the KMeans algorithm turn out to be hard. We therefore use multiple criteria to examine different aspects of the clustering results. Aside from the main loss, we use silhouette graph to measure how tight it is for each cluster. A low silhouette score typically indicates small clusters at the periphery of larger ones.

Besides, we also use cluster size balance as a criteria, which could be naturally visualized with a histogram. This criteria can also be optimized within the KMeans algorithm by adding size as a secondary loss. The overall minimization problem then becomes:

$$id(x) = \arg \min_i \|cluster_i - x\|^2 + \lambda \times size_i$$

$id(x)$ is used to assign a cluster id for each data point $x$. Where $size_i$ represents how many data points have already been assigned to the $i$-th cluster, $cluster_i$ is the center of the $i$-th cluster.

The loss term $\lambda \times size_i$ penalizes clusters with a larger size. The algorithm would prefer to assign new data points to a smaller cluster, which will tend to balance cluster sizes over time. Size homogeneity matches most users’ intuition about clustering, and it may also be important for accurate estimation of cluster statistics. The $\lambda$ also becomes a parameter that we can interactively tune.

5.2.2 Incremental update

In order to take the advantage of mini-batch processing which can return early feedback during the training, we follow a similar approach to [28] for incremental KMeans updating. And we also need an averaging update for $size_i$ to maintain its scale and make the visualization consistent. For
Our last demo is applying our system to Latent Dirichlet Allocation (LDA) topic modeling [3], one of the most widely used topic models.

5.3.1 Implementation

LDA is a generative process to model the documents. For each document \( d \), it proceeds as follows (\( K \) is the number of latent factors):

- Draw a topic distribution for the document \( d \) as \( \theta_d \sim \text{Dirichlet}(\alpha) \), a \( K \)-dimensional Dirichlet.
- For each word position \( i \) (across all docs), draw a topic index \( z_{d,i} \in \{1, \ldots, K\} \) from \( z_{d,i} \sim \theta_d \).
- Draw the word \( w_{d,i} \) from the multinomial distribution \( w_{d,i} \sim \varphi_{z_{d,i}} \), which also has a prior: \( \varphi_{z_{d,i}} \sim \text{Dirichlet}(\beta) \).

\( \alpha, \beta \) are hyper-parameters specifying the Dirichlet prior. The other two parameters of the model are \( \theta \), which can be represented as a document-topic matrix, and \( \varphi \), the word-topic matrix. The algorithm therefore is to use Gibbs sampling to draw samples for hidden states \( z_i \):

\[
P(x_{z_{d,i}} = k | z_{-d,i}, \alpha, \beta, x_{d,i} = w) \sim \theta_{d,k} \cdot \varphi_{k,w}
\]

5.3.2 L1-regularized Topic Model

We first set topic number \( K \) to 1024 and the model converged in about two minutes. As shown in Fig 9, the resulting topic matrix is very sparse even without adding the L1-regularization. This is due to that we set a large \( K \) and many independent topics are generated.

We then set \( K \) to be 32. Without any regularization, the algorithm’s behavior is shown in Fig 10. The likelihood quickly converges to a local optimal, but the topic results are still very noisy, and the topics are overlapping. We then adjust the \( L_1 - \text{reg} \) slider away from zero. However, tuning \( L_1 - \text{reg} \) will not always give good results on complex likelihood functions due to local optima. Since SAME sampling is used in our LDA implementation, we can decrease \( nsamps \) to increase the variance of the random-walk, which makes it easier to jump between possible solutions.

After we increase the temperature, as shown in Fig 11, the likelihood drops significantly but we get a very sparse model.
Afterward, we set the $L_1$-reg back to a small value and use a large $nsamps$ (cold state) which prevents large changes in model state. This is equivalent to only allowing the model to make very small movement around that local optimal. From Fig 12, we can see that the likelihood returns to a normal value while the sparsity is maintained.

More Mixin functions like measuring independence of factors, or coherence of topics could be implemented in the future. Also, our framework is not limited to the unsupervised learning algorithms. Some concrete examples of competing goals in supervised learning include computational marketing where the primary goal is maximize revenue, but where secondary goals include user satisfaction, advertiser satisfaction, and budget constraints. Recommender systems seek to recommended the highest-rated items, but may also need to cover the available item inventory, favor more or less expensive items, or favor items which encourage future purchases. Those are all potential extensions that could be explored in the future.

6. CONCLUSION & FUTURE WORK
We have demonstrated how to perform interactive optimization on customized models using our system. The Mixin function is a convenient and useful way to capture user’s intuition and create customized model. The dashboard also enable users to easily monitor several performance indicators at the same time, which help users to evaluate the model from different perspectives. By summarizing this information, the trade-off decisions become straightforward. Also, the GPU accelerated toolkit makes it possible to get real-time feedback. This ensures that users can understand cause and effect of the algorithm behavior in an iterative refinement procedure.

7. REFERENCES


