NEXT: A system to easily connect crowdsourcing and adaptive data collection

Scott Sievert[‡†*], Daniel Ross[‡†], Lalit Jain[¶†], Kevin Jamieson[§], Rob Nowak[‡], Robert Mankoff[‖]

https://www.youtube.com/watch?v=blPjDYCvppY

Abstract—Obtaining useful crowdsourcing results often requires more responses than can be easily collected. Reducing the number of responses required can be done by *adapting* to previous responses with "adaptive" sampling algorithms, but these algorithms present a fundamental challenge when paired with crowdsourcing. At UW–Madison, we have built a powerful crowdsourcing data collection tool called NEXT (http://nextml.org) that can be used with arbitrary adaptive algorithms. Each week, our system is used by The New Yorker to run their Cartoon Caption contest (http://www.newyorker.com/cartoons/vote). In this paper, we will explain what NEXT is and it's applications, architecture and experimentalist use.

Index Terms—crowdsourcing, adaptive sampling, system

Introduction

The ubiquitousness of the Internet has enabled crowdsourcing, which gives fast access to unprecedented amounts of human judgment data. For example, millions of crowdsourcing participants have been asked to determine the locations in an image that contain a certain object (e.g., "select all image locations that contain buildings") on many different images [DDS+09].

The cost of collecting crowdsourcing responses can be significant – especially in problem domains where expert input is required. Minimizing the number of queries required has large practical benefits: higher accuracy with fewer responses, and ultimately a shorter time to the result. To obtain these benefits, a fundamental change in the method of data collection is required.

At UW–Madison, we have developed a crowdsourcing data collection tool that efficiently collects crowdsourced data via "adaptive" sampling algorithms [JJF+15]. In this paper, we will focus on the use of NEXT rather than the applications of NEXT and their results. We will mention the fundamental problem NEXT addresses, its applications, and the interfaces NEXT presents to the experimentalist and algorithm designer.

Problem statement

Supervised machine learning relies humans to label examples in order to build a model to predict the response a human would give [KZP07]. One example of this workflow is with the popular ImageNet dataset [DDS+09]: humans have provided millions of image labels, and there have been dozens of models to predict labels for unseen images [SLJ+15], [HZRS15], [SZ14].

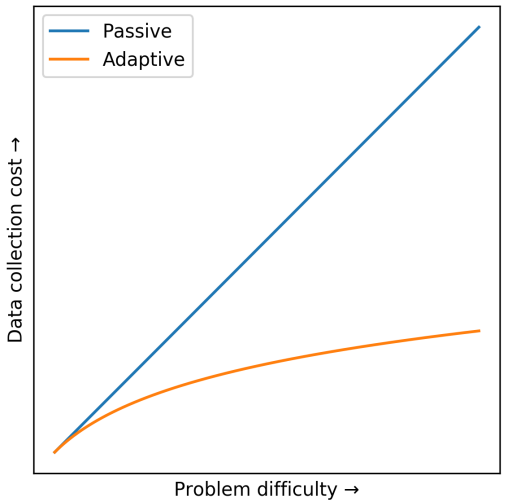

Fig. 1: *As problem difficulty increases, fewer samples (e.g., labeled images) are needed with adaptive sampling to reach a particular quality (e.g., classification accuracy).*

The collection of these data is *passive* and does not *adapt* to previous responses: previous responses do not effect which queries are presented. Adaptive data collection is a process which selects the most useful data as quickly as possible to help achieve some goal (e.g., classification accuracy) [Hol92]. Adaptive data collection is done by an adaptive sampling algorithm that chooses the next query to be labeled.

Adaptive data collection naturally requires fewer responses to produce the same model as passive data collection: it's adapting to previous responses by choosing which query to present next. This is most useful when many labels are needed unlabeled examples. Adaptive algorithms do not require more responses than passive algorithms [CWN05]. A representative depiction of gains obtained by adaptive data collection is shown in Figure 1 [DHM08].

Applying adaptive data collection to crowdsourcing has the potential to reduce the number of samples required. An simple example that requires many human judgments is sorting n items with pairwise comparisons (e.g., $x < y$). In the ideal case, an

† *These authors contributed equally.*
∗ *Corresponding author: stsievert@wisc.edu*
‡ *University of Wisconsin–Madison*
¶ *University of Michigan, Ann Arbor*
§ *University of California, Berkeley*
‖ *The New Yorker*

adaptive algorithm requires $O(n \log n)$ comparisons on average while passive algorithms requires $O(n^2)$ comparisons [Hoa62].

Adaptively collecting large-scale datasets is challenging and time-consuming, as mentioned below. As such, the evaluation of novel adaptive sampling algorithms resort to simulations that use large passively collected datasets. These simulations do not address the practical issues faced in crowdsourcing: adaptive algorithm response time, human fatigue and differing label quality among humans.

The problem that needs to be solved is to allow arbitrary adaptive algorithms to collect crowdsourced data in real time by experimentalists. Arguably, some of the deepest insights and greatest innovations have come through experimentation. This is only possible if adaptive data collection is easily accessible by both

1) Machine learning researchers, to test and deploy adaptive algorithms
2) Experimentalists, to use and test adaptive algorithms in real-world applications

Easy use by both groups will enable feedback between experimentalists and machine learning researchers to improve adaptive data collection through crowdsourcing.

Challenges

Adaptive data collection is not possible without access to previous responses, a fundamental change to data collection. This introduces human feedback: the most useful queries are selected using previously recorded human labels by some adaptive algorithm. If a particular query has shown to be of little use, it doesn't make much sense to label the same query again.

Adaptive algorithms use previous responses to ask questions, which means that they require

- receiving, storing and accessing responses
- delivering and selecting queries to be labeled
- updating some internal model which selects queries to be presented.
- scaling to tens or hundreds of simultaneous users in an online environment when applied to crowdsourcing

General crowdsourcing systems (e.g., Mechanical Turk, PsiTurk, Crowd Flower) were not designed with these requirements in mind. Adaptive data collection requires a fundamentally different interaction flow as show in Figure 2, which requires the data flow in Figure 3 when applied to crowdsourcing.

Crowdsourcing adaptive data collection presents a variety of challenges in mathematics, systems and software development. These challenges stem from the storage and connection of responses to the adaptive sampling algorithm. Any such system needs to process, store and receive crowdsourcing responses and work crowdsourcing scale, meaning the development and maintenance of such a system is involved. This has served as a barrier to developing such a system for mathematicians, and lack of knowledge on adaptive methods have hindered experimentalists.

One other system that addresses this challenge is the Microsoft Decision Service [ABC+16], which can effectively evaluate the collection of crowdsourced data with different adaptive algorithms. However, design of this system involved different goals, including working with exactly one problem formulation and working well at very large scales.

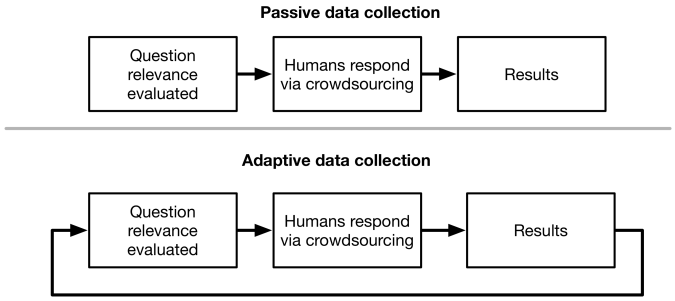

Fig. 2: *The data flows required to collect crowdsourcing data both passively and adaptively. The primary difference is adaptive data collection requires using previous responses in some way.*

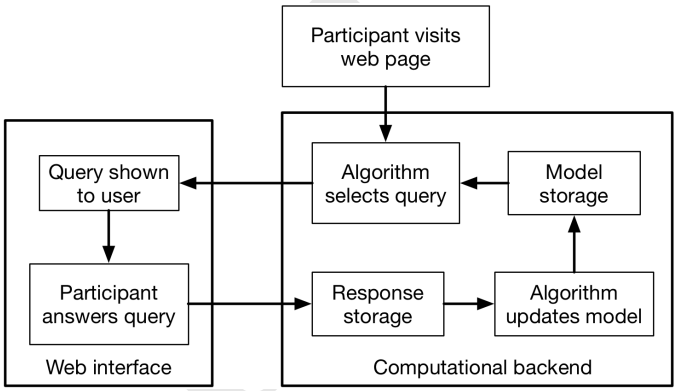

Fig. 3: *The system required to use adaptive algorithm with crowdsourcing. The results are stored in the model, which may contain additional information.*

Our system

The system we have developed at the UW–Madison is called NEXT[1][2]. It provides adaptive, crowdsourced data collection by selecting which query to present *next*. NEXT provides

- easy implementation, selection, and evaluation of different adaptive algorithms
- a web interface for crowdsourced experiment participation
- an HTTP-based API for experiment access (and for use in other contexts)
- live experiment monitoring dashboards that update as responses are received
- easy use and configuration by experimentalists in a wide variety of fields and disciplines

Our design goals necessitate that NEXT be an end-to-end system that is easily accessible. It is a web interface that can be accessed by both experimentalists and crowdsourcing participants, and a Python interface for the algorithm developer. We explain use by experimentalists and algorithm developers in the following sections. A block diagram representation of our system is in Figure 4.

In use of NEXT, mathematicians have implemented new algorithms [Jun16] and UW–Madison psychologists have independently used our system[3]. NEXT has been used by the New Yorker and in the insurance industry. In at least one case, two

1. Homepage at http://nextml.org
2. Source available at https://github.com/nextml/NEXT

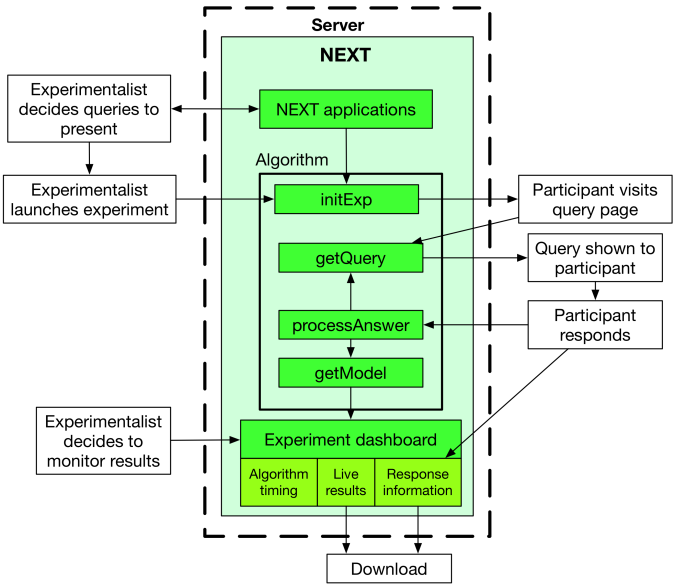

Fig. 4: *When and how different users interact with NEXT. Arrows represent some form of communication between different system components.*

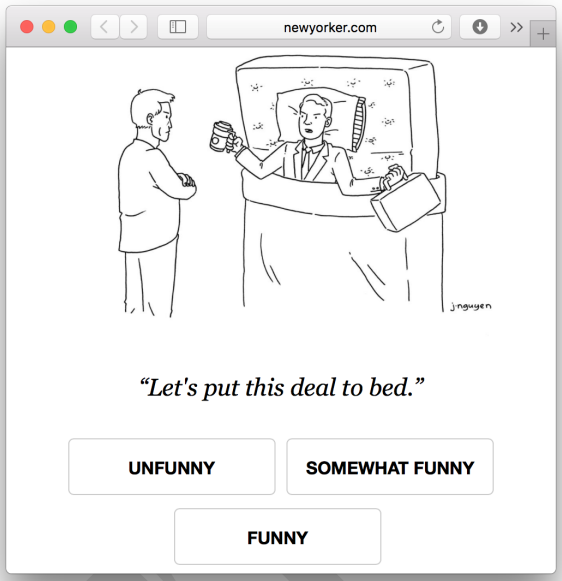

Fig. 5: *An example query shown in The New Yorker Caption Contest (cartoon drawn by P. C. Vey)*

adaptive algorithms have been evaluated in the real world and one required fewer samples as expected[4].

In our usage, the system remains responsive to participants even after receiving millions of responses from thousands of participants. This is illustrated by the problem below, though it also illustrates other features.

Applications of NEXT

NEXT *applications* control the presentation of queries for users to consider.

There are three "built-in" applications shipped with NEXT, geared to three different types of judgments a user can make. These applications are

- Cardinal bandits, which asks participants to rate one object [GGL12] as shown in Figure 5.
- Dueling bandits, which asks participants to select one of two objects [YBKJ12] as shown in Figure 6.
- Triplets, which displays three objects and asks for *triplet responses* of the form "object i is more similar to object j than object k." [JJN16], as shown in Figure 7.

We will now describe each application in more detail.

Cardinal bandits

Each week, The New Yorker draws a cartoon and asks readers for funny captions. They receive about 5,000 captions, of which they have to find the funniest. NEXT runs this contest each week. The interface NEXT provides is visible at http://www.newyorker.com/cartoons/vote and in Figure 5.

The interface is presented every time a query is generated. One caption is presented below the comic with buttons to rate the caption as "unfunny", "somewhat funny" or "funny". Every time one of these buttons is pressed, the adaptive algorithm processes the response and generates a new query.

Each week, we collect and record up to a million ratings from over 10,000 users. All told, this dataset[5] includes over 20 million ratings on over 363,000 different captions. This dataset has been of practical use in improving adaptive sampling algorithms [Jun16].

The New Yorker's goal is to find the funniest caption from this set of 5,000 captions[6]. To achieve this, the algorithms of choice only sample captions that can possibly be the funniest. If a caption has received only "unfunny" ratings, it is probably not the funniest caption and should not be further sampled.

This system has enabled evaluation and improvement in algorithm implementation. In initial contests, we verified that one adaptive algorithm [JMNB14] saw gains over a random algorithm. Later, we implemented an improved adaptive algorithm (KL-UCB at [KK13]) and saw adaptive gains as expected.

This was one of the motivations for NEXT: enabling easy evaluation of adaptive algorithms.

Dueling bandits

We also support asking the crowdsourcing participants to chose the "best" of two items. We tried this method during the first several caption contests we launched for The New Yorker. This interface asks participants to select the funnier of two captions, and is shown in Figure 6. This problem formulation has theoretic guarantees on finding the best item in a set [AB10], but can also be applied to ranking different objects [CBCTH13].

The early evaluation of dueling bandits in the Caption Contest is again part of why we developed NEXT. After trying dueling bandits for several contests, we decided using cardinal bandits is preferable. Cardinal bandits works better at scale, and requires less work by The New Yorker.

3. See http://concepts.psych.wisc.edu/index.php/next-tutorial/
4. With contest 559 of The New Yorker Cartoon Caption contest
5. https://github.com/nextml/caption-contest-data
6. The top caption for the comic in Figure 5 was "Like you've never taken anything from a hotel room"

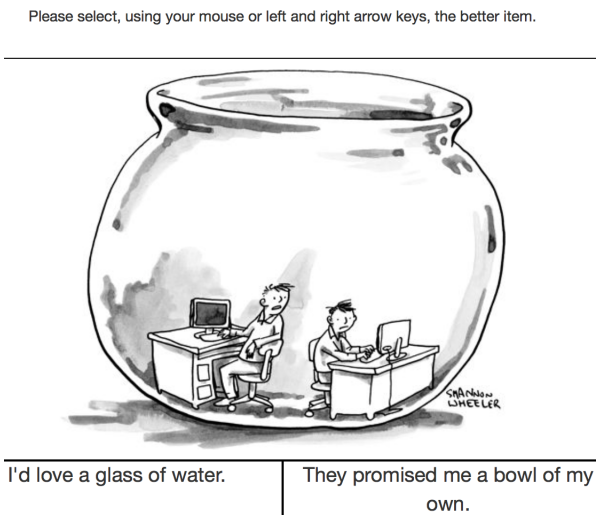

Fig. 6: *The dueling bandits interface, where two items are compared and the "better" item is selected (cartoon drawn for The New Yorker Caption Contest by Shannon Wheeler)*

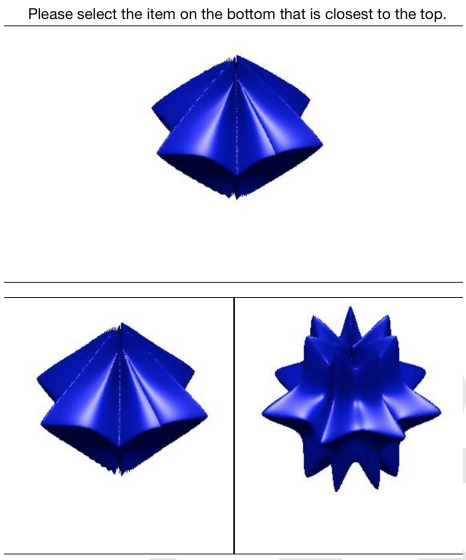

Fig. 7: *An interface that asks the user to select the most similar bottom object in relation to the top object.*

Triplets

Finding a similarity measure between different objects is the goal of this problem formulation. For example, it may be desired to find the similarity between different facial expressions. Happy and excited faces may be similar but are probably different from sad faces.

Human attention span cannot handle the naive number of comparisons (which is proportional to n^2 with n items). Instead, we ask the crowdsourcing participant to make a pairwise similarity judgement, or a triplet response as shown in Figure 7. There are theoretic guarantees on finding some similarity measure given these responses [JJN16] and have been used in practice with NEXT to compare visual representations of different molecules [RMN16].

NEXT Architecture

The design goals of NEXT are to provide

- convenient default *applications* (which handle different problem formulations by serving different types of queries; e.g., one application involves the rating of exactly one object)
- straightforward and modular algorithm implementation
- live experiment monitoring tools via a dashboard, which must update as responses are received and provide some sort of offline access
- easy experimentalist use, both in system launch and in experiment launch

These different system components and their data flow is shown in Figure 4. Complete system documentation is available and addresses use cases seen by both algorithm developers and experimentalists[7].

Algorithm implementation

Required functions: To implement Figure 4, we must implement four functions for each algorithm:

1) `initExp`, which initializes the algorithm when the experiment is launched
2) `getQuery`, which generates a query to show one participant
3) `processAnswer`, which processes the human's answer
4) `getModel`, which gets the results and is shown on the dashboard

These function handle various objects to displayed in each query (e.g., the New Yorker displays one text object in every query for a rating). By default, these objects or *target* are abstracted to an integer index (though the other information is still accessible). This means that a particular target is referred to only by index (e.g., the user is seeing target i, not `foo.png`).

All these functions are implemented in Python, and we provide easy access other tasks needed for adaptive algorithms (database access, background jobs).

Arguments and returns: We treat each algorithm as a black box – NEXT only needs each algorithm function to accept and return specific values. These arguments and return values for all algorithm functions are specified exactly in a YAML-based schema. Every algorithm has to create a mapping from the specified inputs to the specified outputs.

NEXT verifies the inputs and output to/from algorithms and can also include a description of each parameter. This means that YAML schema is always up to date and is self-documenting. Changing this schema means different arguments are passed to every algorithm, and we offer flexibility by allowing arguments of any type to be passed.

This schema depends on `Algs.yaml` (e.g., in `apps/[application]/algs/Algs.yaml`) and contains four root level keys for each of `initExp`, `getQuery`, `processAnswer`, and `getModel`. Each one of these sections describes the input arguments and returns values by `args` and `rets` respectively. These sections are filled with type specifications that describe the name and type of the various keyword arguments.

For example, a particular `Algs.yaml` may include

```
getQuery:
  args:
    participant_uid:
```

7. Documentation can be found at https://github.com/nextml/NEXT/wiki

```
      type: string
      description: ID of the participant answering the query
  rets:
    description: The index of the target to ask about
    type: num
```

The keyword argument `participant_uid` is specified in the `args` key, and the return value must be a number. The corresponding `getQuery` implementation would be

```
def getQuery(butler, participant_uid):
    return 0  # for example
```

More complete documentation on these parameter specifications, which can be found at the API endpoint `assistant/doc/[application-name]/pretty`.

Database access: We provide a simple database wrapper, as algorithms need to store different values (e.g., the number of targets, a list of target scores). We provide a variety of atomic database operations through a thin wrappers to PyMongo[8] and Redis[9], though we can support arbitrary databases[10]. Each "collection" in this wrapper mirrors a Python dictionary and has several other atomic database operations. We provide

- `get`, `set` and `{get, set}_many` which provide atomic operations to store values in the database
- `append` and `pop`, which atomically modify list values, and return the result
- `increment`, which atomically increments a stored value by a given amount

All these operations are atomic, and can be accessed through an interface called `butler` which contains multiple collections. The primary collection used by algorithms (`butler.algorithms`) is specific to each algorithm and allows for independent evaluation of different algorithms (though other collections are available). The arguments to an algorithm function are `butler` followed by the values in the schema.

Example: This example illustrates the interface we have created for the algorithm developer and provides an example of algorithm implementation. After implementation, this algorithm can receive crowdsourcing responses through the web interface.

```
import numpy as np

def choose_target(butler):
    # Adaptive sampling hidden for brevity
    n = butler.algorithms.get(key='n')
    return np.random.choice(n)

class MyAlg:
    def initExp(self, butler, n):
        butler.algorithms.set(key='n', value=n)
        scores = {'score'+ str(i): 0 for i in range(n)}
        pulls = {'pulls' + str(i): 0 for i in range(n)}
        butler.algorithms.set_many(
            key_value_dict=scores
        )
        butler.algorithms.set_many(
            key_value_dict=pulls
        )

    def getQuery(self, butler):
        return choose_target(butler)

    def processAnswer(self, butler,
                      target_id, reward):
        butler.algorithms.increment(
            key='score' + str(target_id),
            value=reward
        )
        butler.algorithms.increment(
            key='pulls' + str(target_id),
        )

    def getModel(self, butler):
        n = butler.algorithms.get(key='n')
        scores = [butler.algorithms.get(
                    'score' + str(i))
                  for i in range(n)]
        pulls = [butler.algorithms.get(
                    'pulls' + str(i))
                 for i in range(n)]
        mean_scores = [s/p if p != 0 else float('nan')
                       for s, p in zip(scores, pulls)]
        return mean_scores
```

The `Algs.yaml` file for this algorithm would be

```
initExp:
  args:
    n:
      description: Number of targets
      type: num
getQuery:
  rets:
    type: num
    description: The target to show
                 the user
processAnswer:
  args:
    target_id:
      description: The target_id that was shown
                   to the user
      type: num
    reward:
      description: The reward the user gave
                   the target
      values: [1, 2, 3]
      type: num
getModel:
  rets:
    type: list
    description: The scores for each target ordered
                 by target_id.
    values:
      description: The mean score for a particular target
      type: num
```

Experiment dashboards

NEXT can be monitored in real-time via dashboards for each experiment, which include:

- experiment logs
- basic information (launch date, number of received responses, etc)
- the results, with current responses received (example in Figure 8)
- client- and server-side timing information
- download links to the responses and the live results (which allows processing of these data offline).

The dashboards include histograms for both human response time and network delay (time taken for NEXT to respond to request), a measure of system responsiveness. An example is shown in Figure 9. These dashboards also include timing information for algorithm functions, a useful debugging tool for the algorithm developer.

From the dashboard, we support the download of both experiment results and participant response information.

8. http://api.mongodb.com/python/current

9. https://redis.io/

10. Which requires implementation of the Collection API found in `next.apps.Butler`

Rankings

BR_LilUCB BR_Random

Rank	Target	Score	Precision
0	I'm drowning in work.	0.91667	0.20412
1	The women's office is the same, except it has glass on the top, too.	0.86957	0.20851
2	If you work hard they give you a rock and plastic seaweed.	0.78571	0.26726
3	Your seat cushion can be used as a flotation device.	0.75000	0.28868

Fig. 8: *The dashboard display of results from different algorithms for the example in Figure 6.*

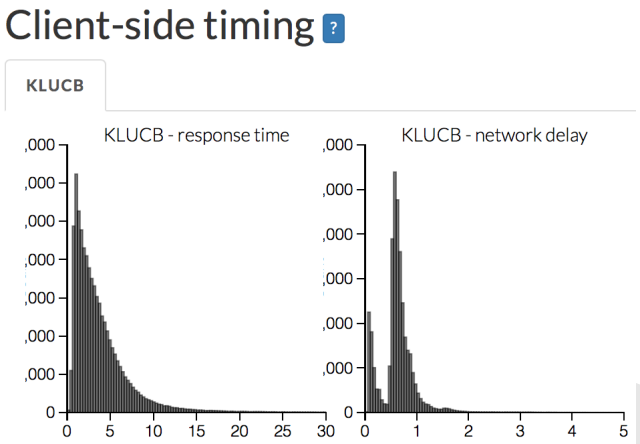

Fig. 9: *Timing histograms measured client-side in seconds for cartoon caption contest 573. Network delay represents the total time NEXT took to respond and response time measures human resposne time.*

Experimentalist use

Below, we will refer to different NEXT features which are available through different API endpoints. After NEXT has launched, these are available via HTTP on port `8000` on the hosting machine. In practice, this means the API endpoint `/home` (for example) is available at `[next-url]:8000/home` when `[next-url]` is one of `ec2-...-amazonaws.com` or `localhost`.

Launching NEXT: The easiest way to launch NEXT is through Amazon EC2 (which can provide the interface required for crowdsourcing) and their AMI service. After launch, the main NEXT interface is available at the API endpoint `/home` which provides links to the list of dashboards, an experiment launching interface and the associated documentation.

Launching can be done by selecting the "Launch instance" button on Amazon EC2 and choosing the AMI "NEXT_AMI", `ami-36a00c56` which is available in the Oregon region. We recommend that production experiments be run on the EC2 instance-type `c4.8xlarge`, a server large enough to provide the necessary memory and compute power. A complete guide can be found in the documentation at https://github.com/nextml/NEXT/wiki.

Experiment launch: Experiments are launched by providing two files to NEXT, either via a web interface or an API endpoint. An experiment description file is required. The other (optional) file enumerate the objects under consideration ("target"). These two files can be uploaded through the interface available at `/assistant/init`.

The experiment description contains the information required to launch and configure the experiment. The following experiment description was used to generate the image in Figure 6:

```
app_id: CardinalBanditsPureExploration
args:
  alg_list:
  - {alg_id: KLUCB, alg_label: KLUCB}
  algorithm_management_settings:
    mode: fixed_proportions
    params:
    - {alg_label: KLUCB, proportion: 1.0}
  context: # image URL, trimmed for brevity
  context_type: image
  failure_probability: 0.05
  participant_to_algorithm_management: one_to_many
  rating_scale:
    labels:
    - {label: unfunny, reward: 1}
    - {label: somewhat funny, reward: 2}
    - {label: funny, reward: 3}
```

These parameters are defined in schemes, and are documented at the API endpoint `/assistant/doc/[application-id]/pretty` in the "initExp" section.

The other file necessary for experiment launch is a ZIP file of targets (e.g., the images involved in each query). We support several different formats for this ZIP file so images, text and arbitrary URLs can be supported. If images are included in this ZIP file, we upload all images to Amazon S3.

Experimentalist use with crowdsourcing: After experiment launch, a link to the experiment dashboard and query page is presented. We recommend distributing this query page link to crowdsourcing participants, which typically happens via Mechanical Turk or email.

Experiment persistence: We support saving and restoring experiments on the experiment list at `/dashboard/experiment_list`. This allows experiment persistence even when Amazon EC2 machines are terminated.

Conclusion

At UW–Madison, we have created a system that is connecting useful adaptive algorithms with crowdsourced data collection. This system has been successfully used by experimentalists in a wide variety of disciplines from the social sciences to engineering to efficiently collect crowdsourced data; in effect, accelerating research by decreasing the time to obtain results.

The development of this system is modular: sampling algorithms are treated as black boxes, and this system is accessible with other interfaces. NEXT provides useful experiment monitoring tools that update as responses are received. This system has shown to be cost effective in bringing decision making tools to new applications in both the private and public sectors.

References

[AB10] Jean-Yves Audibert and Sébastien Bubeck. Best arm identification in multi-armed bandits. In *COLT-23th Conference on Learning Theory-2010*, pages 13–p, 2010.

[ABC+16] Alekh Agarwal, Sarah Bird, Markus Cozowicz, Luong Hoang, John Langford, Stephen Lee, Jiaji Li, Dan Melamed, Gal Oshri, Oswaldo Ribas, et al. A multiworld testing decision service. *arXiv preprint arXiv:1606.03966*, 2016.

[CBCTH13] Xi Chen, Paul N Bennett, Kevyn Collins-Thompson, and Eric Horvitz. Pairwise ranking aggregation in a crowdsourced setting. In *Proceedings of the sixth ACM international conference on Web search and data mining*, pages 193–202. ACM, 2013.

[CWN05] Rui Castro, Rebecca Willett, and Robert Nowak. Faster rates in regression via active learning. In *NIPS*, volume 18, pages 179–186, 2005.

[DDS+09] Jia Deng, Wei Dong, Richard Socher, Li-Jia Li, Kai Li, and Li Fei-Fei. Imagenet: A large-scale hierarchical image database. In *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*, pages 248–255. IEEE, 2009.

[DHM08] Sanjoy Dasgupta, Daniel J Hsu, and Claire Monteleoni. A general agnostic active learning algorithm. In *Advances in neural information processing systems*, pages 353–360, 2008.

[GGL12] Victor Gabillon, Mohammad Ghavamzadeh, and Alessandro Lazaric. Best arm identification: A unified approach to fixed budget and fixed confidence. In *Advances in Neural Information Processing Systems*, pages 3212–3220, 2012.

[Hoa62] Charles AR Hoare. Quicksort. *The Computer Journal*, 5(1):10–16, 1962.

[Hol92] John H Holland. *Adaptation in natural and artificial systems: an introductory analysis with applications to biology, control, and artificial intelligence*. MIT press, 1992.

[HZRS15] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. In *Proceedings of the IEEE international conference on computer vision*, pages 1026–1034, 2015.

[JJF+15] Kevin G Jamieson, Lalit Jain, Chris Fernandez, Nicholas J Glattard, and Rob Nowak. Next: A system for real-world development, evaluation, and application of active learning. In *Advances in Neural Information Processing Systems*, pages 2656–2664, 2015.

[JJN16] Lalit Jain, Kevin G Jamieson, and Rob Nowak. Finite sample prediction and recovery bounds for ordinal embedding. In *Advances in Neural Information Processing Systems*, pages 2711–2719, 2016.

[JMNB14] Kevin Jamieson, Matthew Malloy, Robert Nowak, and Sébastien Bubeck. lil'ucb: An optimal exploration algorithm for multi-armed bandits. In *Conference on Learning Theory*, pages 423–439, 2014.

[Jun16] Kwang-Sung Jun. Anytime exploration for multi-armed bandits using confidence information. In *Proceedings of The 33rd International Conference on Machine Learning*, pages 974–982, 2016.

[KK13] Emilie Kaufmann and Shivaram Kalyanakrishnan. Information complexity in bandit subset selection. In *COLT*, pages 228–251, 2013.

[KZP07] Sotiris B Kotsiantis, I Zaharakis, and P Pintelas. Supervised machine learning: A review of classification techniques, 2007.

[RMN16] Martina A Rau, Blake Mason, and Robert Nowak. How to model implicit knowledge? similarity learning methods to assess perceptions of visual representations. In *Proceedings of the 9th International Conference on Educational Data Mining*, 2016.

[SLJ+15] Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet, Scott Reed, Dragomir Anguelov, Dumitru Erhan, Vincent Vanhoucke, and Andrew Rabinovich. Going deeper with convolutions. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 1–9, 2015.

[SZ14] Karen Simonyan and Andrew Zisserman. Very deep convolutional networks for large-scale image recognition. *arXiv preprint arXiv:1409.1556*, 2014.

[YBKJ12] Yisong Yue, Josef Broder, Robert Kleinberg, and Thorsten Joachims. The k-armed dueling bandits problem. *Journal of Computer and System Sciences*, 78(5):1538–1556, 2012.