University of Washington Technical Report UW-CSE-12-11-02

Cascade: Crowdsourcing Taxonomy Creation

Lydia Chilton*, Greg Little**, Darren Edge***, Daniel S. Weld*, James A. Landay*

*University of Washington, **oDesk, ***Microsoft Research Asia

ABSTRACT
Taxonomies are a useful and ubiquitous way of organizing information. However, creating good organizational hierarchies is difficult because the process requires a global understanding of the objects to be categorized. Usually this is done by an individual or a small group of people working together. This approach does not work well for the quickly changing, large datasets found on the web. Cascade is a crowd workflow that allows crowd workers to spend as little at 20 seconds each towards collectively making a taxonomy. To achieve these results, we introduce a design pattern of generate-apply-edit and a novel technique called adaptive context filtering that allows the crowd to do robust categorization. We evaluate Cascade and show that on three datasets its quality is 80-90% of that of experts. Cascade has a competitive cost to expert information architects, but takes six times more human labor. However, that labor can be parallelized such that Cascade will run in as fast as four minutes instead of hours or days.

INTRODUCTION

Taxonomies are a useful and ubiquitous way of organizing information. However, creating good organizational hierarchies is difficult because the process requires a global understanding of the objects to be categorized. Currently, most taxonomies are created by a small group of experts that analyze a complete dataset before identifying the essential distinctions for classification. Unfortunately, this process is too expensive to apply to many of the user-contributed datasets forming on the Internet. Despite recent progress, completely automated methods, such as Latent Dirichlet Allocation (LDA) and related AI techniques, produce low-quality taxonomies. They lack the common sense, flexibility and language abilities that come naturally to people.

This paper presents Cascade, a novel method for creating taxonomies. Cascade applies automated techniques to combine the suggestions of many humans, none of whom have a global perspective of the data or the taxonomy under construction. Cascade uses crowdsourcing, a distributed method for solving a task by broadcasting an open call for solutions to its subtasks and composing the responses into an integrated answer. Crowdsourcing has become a popular way to solve problems that are too hard for today's AI techniques, such as translation, linguistic tagging, and visual interpretation. However, our method is novel because of the global nature of the taxonomy-creation problem. Most successful crowdsourcing systems operate on problems that naturally break into small units of labor, e.g., labeling millions of independent photographs. In contrast, Cascade develops a unique, iterative workflow which is non-trivial to break down and distribute amongst hundreds of people and demands no more than one minute of labor from any worker.

To create sophisticated crowdsourcing workflows, we need to employ design patterns that guide us to breaking down work. Just as iterative design is a guide to developing usable software, and map reduce is a design to help process distributed data, crowdsourcing needs to follow design patterns to achieve results. In Cascade, the crowd is led to follow a design pattern that we adapted from the creative process – generate lots of ideas for categories, and edit down the group of ideas to a cohesive whole by applying those categories to the raw data.

In this paper we first present the Cascade Algorithm and describe the three human intelligence task (HIT) primitives used to implement it. We then demonstrate the results of running Cascade on three representative data sets scraped from the Internet. We evaluate Cascade in three ways: we compare its time and cost to that of four expert information architects we paid to taxonomize the same data, we count the number of mistakes in Cascade's output and interpret it as an error rate, and we compare the coverage of the categories in Cascade to those of the expert-made taxonomies.

In summary, this paper makes the following contributions:

1. A novel *crowd algorithm,* Cascade, that produces a *global understanding of large datasets* from the actions of individual contributors, none of whom see more than a fraction of the data.
2. The *design pattern of generate-apply-edit* that provides the framework for the Cascade algorithm.
3. A technique called *adaptive context filtering* that allows the crowd to do robust categorization when the matches are sparse.
4. An evaluation of Cascade on three datasets showing that Cascade can perform close to expert level agreement (80-90% of expert performance) for competitive time and cost.

INITIAL APPROACHES

The Cascade algorithm evolved from a sequence of initial prototypes based on common crowdsourcing patterns. Our experience resulted in a several surprising observations that informed our ultimate design of a crowdsourced taxonomy workflow.

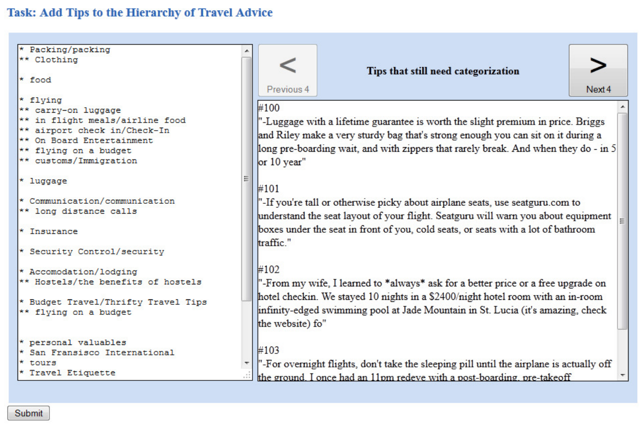
Iterative Improvement

Iterative improvement is a general crowdsourcing pattern first described in TurKit[10]. Iterative improvement has proven successful at using multiple workers to build on and improve each other's image descriptions, and to collectively decipher blurry text or bad handwriting. For example, given blurry text to transcribe, one worker will transcribe as much as he can, and another worker will iterate on the first worker's answer. The workflow can then ask a third worker to vote on whether the second worker has made an acceptable contribution. This can be repeated until a stopping condition is met – such as the entire text being transcribed or workers no longer being able to improve the transcription.

We applied iterative improvement to taxonomy creation by giving workers a list of tips and an editable hierarchy interface. Workers were asked to improve the taxonomy by adding, deleting, or moving categories or by placing tips in the taxonomy. We tested two iterative improvement interfaces: A text-based Wikipedia-style outline editing interface (see Figure 1.) and a drag and drop, folder-tree interface (Figure 2). We observed that both iteratively improvement interfaces suffered from the same two problems.

1. **The taxonomy grows quickly making the tasks more time consuming and overwhelming as time goes on.** The first tasks are very easy – creating the first category and placing a few tips in it is quick and easy. However, when there are 50 categories and you have to read all 50 categories to figure out whether or not it belongs in any of the existing categories, the work becomes so challenging that single workers have a difficult time making contributions in a short time frame. Additionally, taxonomy structure editing tasks such as merging two categories are proved difficult for workers because they require understanding a large fraction of tips to decide whether all the tips under "air travel" should be merged with the category "flying".
2. **The task had many options for how to contribute (add categories, place tips, merge categories, etc.) and workers had trouble selecting tasks.** Although giving workers options for how to contribute made the task very flexible, it also meant workers had to decide what was important to do next. One of the things that made it so hard to know what to do next is that you can't coordinate with future workers about what they are willing to do and how they will interpret your contributions. The uncertainty that workers faces led us to conclude that the "what to do next" problem was hard enough that we needed to embed structure in the workflow to identify and support the major decisions of how to create a taxonomy.

From our observations, we concluded that we needed to break down the taxonomization task into manageable units of work. Although iterative improvement may work for tasks where the task selection is obvious and the ultimate output is a manageable size for a single worker, it is less successful for large tasks that require significant coordination between workers in order to know what to do next.


Task: Add Tips to the Hierarchy of Travel Advice
<
Previous 4
Tips that still need categorization
>
Next 4
Submit

```
* Packing/packing
** Clothing

* food

* flying
** carry-on luggage
** in flight meals/airline food
** airport check in/Check-In
** On Board Entertainment
** flying on a budget
** customs/Immigration

* luggage

* Communication/communication
** long distance calls

* Insurance

* Security Control/security

* Accomodation/lodging
** Hostels/the benefits of hostels

* Budget Travel/Thrifty Travel Tips
** flying on a budget


* personal valuables
* San Fransisco International
* tours
* Travel Etiquette
```

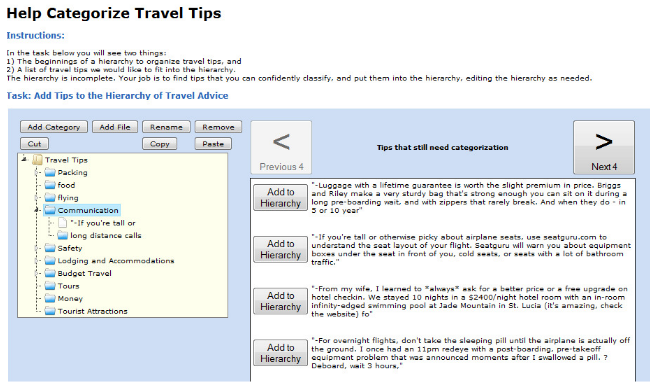
#100
"-Luggage with a lifetime guarantee is worth the slight premium in price. Briggs and Riley make a very sturdy bag that's strong enough you can sit on it during a long pre-boarding wait, and with zippers that rarely break. And when they do - in 5 or 10 year"

#101
"-If you're tall or otherwise picky about airplane seats, use seatguru.com to understand the seat layout of your flight. Seatguru will warn you about equipment boxes under the seat in front of you, cold seats, or seats with a lot of bathroom traffic."

#102
"-From my wife, I learned to *always* ask for a better price or a free upgrade on hotel checkin. We stayed 10 nights in a $2400/night hotel room with an in-room infinity-edged swimming pool at Jade Mountain in St. Lucia (it's amazing, check the website) fo"

#103
"-For overnight flights, don't take the sleeping pill until the airplane is actually off the ground. I once had an 11pm redeye with a post-boarding, pre-takeoff

Figure 1.Iterative Improvement Interface #1



In the task below you will see two things:
1) The beginnings of a hierarchy to organize travel tips, and
2) A list of travel tips we would like to fit into the hierarchy.
The hierarchy is incomplete. Your job is to find tips that you can confidently classify, and put them into the hierarchy, editing the hierarchy as needed.

"-Luggage with a lifetime guarantee is worth the slight premium in price. Briggs and Riley make a very sturdy bag that's strong enough you can sit on it during a long pre-boarding wait, and with zippers that rarely break. And when they do - in 5 or 10 year"

"-If you're tall or otherwise picky about airplane seats, use seatguru.com to understand the seat layout of your flight. Seatguru will warn you about equipment boxes under the seat in front of you, cold seats, or seats with a lot of bathroom traffic."

"-From my wife, I learned to *always* ask for a better price or a free upgrade on hotel checkin. We stayed 10 nights in a $2400/night hotel room with an in-room infinity-edged swimming pool at Jade Mountain in St. Lucia (it's amazing, check the website) fo"

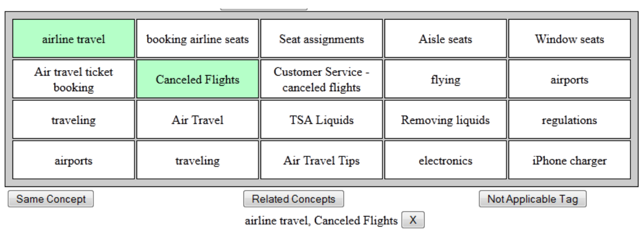
"-For overnight flights, don't take the sleeping pill until the airplane is actually off the ground. I once had an 11pm redeye with a post-boarding, pre-takeoff equipment problem that was announced moments after I swallowed a pill. ? Deboard, wait 3 hours,"

Figure 2 Iterative Improvement Interface #2

Category Comparison

Many crowdsourced workflows start with a step for generating data (suggesting image labels [ESPGame], finding places where writing can be improved [Soylent], drawing bounding boxes on food [PlateMate,Gajos]), and follow up with a step that uses those initial generated data. For example, the ESP game uses two people to generate image labels, and accepts only the shared labels. We tried a similar approach for Cascade. We first asked workers to read tips and generate categories for them. Next, we wanted to organize the categories into a cohesive taxonomy by removing duplicate categories and nesting subset categories under superset categories (for example, nesting "boarding" under the "air travel" category). Although the category generation step worked well (and is in the final version of Cascade), we unfortunately did not get promising results for any of the designs we implemented for identifying the relationships between categories.

When identifying the relationships between categories, A and B, we want to know if A equals B, if A is a subset of B, if B is a subset of A, or if A and B have no relationship. To find this out, we pick two categories and ask workers which relationship they have. In one design, we explicitly asked whether categories were equal, supersets, subsets, or unrelated (Figure 3,4). In another design we showed the workers 20 categories and let them select the pairs they wanted to compare and then say whether they were the same concept, or related concepts (we relaxed the notion of equals, or superset/subset to eliminate any problems from workers not understanding the notion of a superset and a subset.) Although this approach seems natural and straightforward, regardless of the design, our workers did not agree with each other and didn't agree with our "gold standard" judgments.

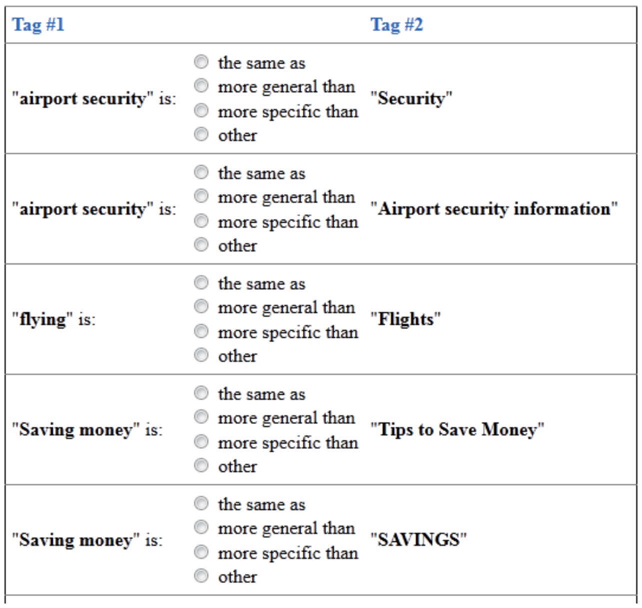


airline travel	booking airline seats	Seat assignments	Aisle seats	Window seats
Air travel ticket booking	Canceled Flights	Customer Service - canceled flights	flying	airports
traveling	Air Travel	TSA Liquids	Removing liquids	regulations
airports	traveling	Air Travel Tips	electronics	iPhone charger

Figure 3 Category Comparison interface #1

The underlying problem in category comparison is the inherent difficulty of comparing two abstractions. In order to compare two abstract category names, workers must make assumptions about their meaning – some people may interpret categories broadly and some narrowly, some people may see connections that others may not. There are many assumptions that go into comparing two abstractions and without a way for workers to write down their assumptions, their judgments of the relationship between two categories weren't comparable. For example, in the dataset of travel tips, our "gold standard" considered categories "air travel" and "flying" to be the same concept. Some workers agreed with this but many workers considered these to be different. They thought that "air travel" was a superset of "flying." In fact, both interpretations are correct. The category "flying" can mean both the entire set of activities around flying (similar to "air travel") and can mean just the part of air travel where you are on an airplane and in the air. Difficult comparisons also included "TSA liquids" and "removing liquids", "packing" and "what to bring", and "advice" and "general advice." In order to effective compare these two abstractions workers need more context to ground the meaning of the abstract category labels. In this case, the context is the tips.

From these observations, we concluded that it was a mistake asking workers to compare abstractions to abstractions – the differing assumptions people make about abstractions are too hard to write down and render individual worker's judgments incomparable. Instead, judgments should be made relating actual abstractions to data, in this case, relating categories to tips.



Tag #1		Tag #2
"airport security" is:	○ the same as ○ more general than ○ more specific than ○ other	"Security"
"airport security" is:	○ the same as ○ more general than ○ more specific than ○ other	"Airport security information"
"flying" is:	○ the same as ○ more general than ○ more specific than ○ other	"Flights"
"Saving money" is:	○ the same as ○ more general than ○ more specific than ○ other	"Tips to Save Money"
"Saving money" is:	○ the same as ○ more general than ○ more specific than ○ other	"SAVINGS"

Figure 4 Category Comparison interface #2

Tip Clustering (without Categories)

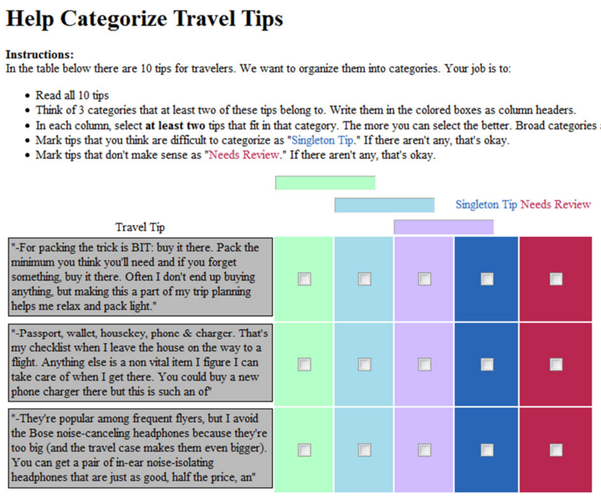
In effort to cluster tips, we tried putting tips in small, randomly generated groups and asking workers which tips were the most similar. Workers were eager to do this task, but unfortunately, their judgments were not consistent. The problem is that there are many potential similarities among data that is very open-ended in its coverage. In order to compare workers' judgments of tip similarity, we not only need to know which tips they think are most similar but also the dimension along which their similarity is being judged. For example, if we have three tips, A, B and C. We could have that A and B are both about "air travel", B and C are both about "saving money" and C and A are both about "websites." In this case, all there is no clear pair of similar tips in this set. The pair of tips that you pick as similar depends on the dimension on which they are being compared.

If this problem were rare, we could perhaps work around it, but in an open world, there are many (dozens, hundreds, thousands, possibly more?) dimensions on which to measure things. We concluded that to combat the incomparability of judgments based on unstated assumptions it is necessary to extract the dimension of similarity as well as clusters.

Tip Clustering (with Categories)

In order to elicit both clusters and cluster names from workers, we prototyped an interface which presented workers with a small number of tips (8-10) and asked them to suggest categories that fit at least two tips (figure 5). Although workers found the task easy and intuitive, the quality of the categories was not as good as when we generated category suggestions for single tips. Restricting workers to naming categories that satisfied multiple tips seemed to encourage people to name overly broad categories that fit all the 8 tips such as "good tips" or "advice" and prevented them from naming the category that would fit one tip perfectly, and would perhaps fit other tips which didn't happen to be in the random group of 8. For example, workers can use their common sense and intuition about travel advice and infer that "TSA Security" might fit multiple tips even though it only fits one tip in the 8-tip subset.

We decided that the clusters workers found in small subset of the data were often unnatural and forced and that **it was better to allow workers to suggest categories that fit one tip very well rather than fit multiple tips more loosely.**



Help Categorize Travel Tips

Instructions:
In the table below there are 10 tips for travelers. We want to organize them into categories. Your job is to:

- Read all 10 tips
- Think of 3 categories that at least two of these tips belong to. Write them in the colored boxes as column headers.
- In each column, select **at least two** tips that fit in that category. The more you can select the better. Broad categories are good.
- Mark tips that you think are difficult to categorize as "Singleton Tip." If there aren't any, that's okay.
- Mark tips that don't make sense as "Needs Review." If there aren't any, that's okay.

Travel Tip				Singleton Tip	Needs Review
"-For packing the trick is BIT: buy it there. Pack the minimum you think you'll need and if you forget something, buy it there. Often I don't end up buying anything, but making this a part of my trip planning helps me relax and pack light."	☐	☐	☐	☐	☐
"-Passport, wallet, housekey, phone & charger. That's my checklist when I leave the house on the way to a flight. Anything else is a non vital item I figure I can take care of when I get there. You could buy a new phone charger there but this is such an of"	☐	☐	☐	☐	☐
"-They're popular among frequent flyers, but I avoid the Bose noise-canceling headphones because they're too big (and the travel case makes them even bigger). You can get a pair of in-ear noise-isolating headphones that are just as good, half the price, an"	☐	☐	☐	☐	☐

Figure 5 Tip Clustering with categories interface

Divide and Conquer

One approach to taxonomy creation is to partition data into several the large categories, then recursively divide large categories into subcategories. For example, in the travel data, you might find that "air travel" is a large category. By recursing into the "air travel tips" you could then sub categories of "air travel" such as "in-flight meals", "layover", and "TSA security." We found that this approach has one critical problem which is related to the difficulty of comparing abstractions in the Category Comparison prototype.

In our experience, when soliciting subcategories for "air travel" workers do suggest strict subcategories of air travel such as "in-flight meals" but workers also suggest categories which may or may not be strict subcategories. The categories they suggest might in fact be cross-cutting facets that apply outside of "air travel." For example, a commonly suggested subcategory of "air travel" is "saving money" because many tips are about saving money on air fare. However, "saving money" doesn't strictly apply to "air travel" it could apply to nearly anything.

We learned from Category Comparison prototype that comparing abstraction to abstraction is hard – given two categories labels it is often difficult to say whether one is a proper subset of the other. If the suggested subcategories are in fact not subcategories, then we aren't partitioning the data effectively, which was the crux of the efficiency gain from a divide and conquer approach.

One approach to fixing the problem of ambiguous hierarchical relationships of categories is to narrow down "saving money" to "saving money on air travel." Thus, by construction, "saving money on air travel" is a subcategory of "air travel." The danger of doing this is that "saving money" might be a subset of many large categories: "saving money in international travel" "saving money on hotels", "saving money on train travel" and possibly many more. It's possible that "saving money" is a big category, and we don't want it appearing multiple times in the taxonomy split across multiple categories. If we allowed this, then the taxonomy would have a very large branching factor and explode quickly.

We concluded that the divide and conquer approach to creating taxonomies did not work because of the difficulty deciding whether a category was truly a strict subcategory of another or a cross-cutting facet without looking at all the data. Without lines along which we could effectively partition the data and then only work on that partition, divide and conquer is not an effective strategy.

Summary

From initial approaches to crowdsourcing taxonomy creation, we learned several things. We first learned that the task needed to be deliberately broken down into subtasks. Next, we learned that comparing abstractions to abstractions was noisy and thus it was better to compare abstraction to data (in this case, comparing categories to data). We learned that to get better tip similarity judgments, we needed to extract the dimension of similarity along with tips were being compared, and make sure that each dimension of similar fit at least one tip very well as opposed to fitting multiple tips more loosely. Lastly, we learned that it is better to solicit multiple categories that overlap ("air travel", "saving money", "luggage") and figure out the hierarchical relationships between them later rather than trying to identify the biggest categories and then recurse into them.

These approaches lead us to our final algorithm design wherein we first generate many category names, we next vote on the best category names to remove spam, chose the best of multiple alternatives, and to eliminate vague categories, and categories that otherwise are sub-par. Next we categorize all the data into those categories, and using the statistical overlap between the members of two categories, decide which ones should appear in the final taxonomy and what their hierarchical relationships should be.

THE CASCADE ALGORITHM

In this section we discuss the inputs and output of Cascade, the parameters that govern its execution, the steps of the algorithm, the three HIT primitives used to implement it, and the conditions for running Cascade iteratively on a dynamic data set. We include a running-time and cost analysis of Cascade.

Inputs

Cascade takes two inputs: a set of items (called the *tips*) to be categorized and a descriptive phrase (the *topic*) identifying this set. Although "items" is a more general term, and has the correct level of generality, we find that in talking about Cascade, the concreteness of the term "tip" is less confusing than "items." An example of inputs to Cascade are a tip-set of 100 responses to the question "What is your best travel advice?" with the topic "Travel Advice."

This paper demonstrates Cascade only on textual items (tips). However, we believe Cascade is general enough to work for a wide range of entities (e.g., images, audio clips, video, charts) as long as each item can be judged independently by a human. Cascade is not designed to run ordered sequences of items (such as essay paragraphs or sequential audio clips). Cascade is designed to run on tips that take less than one minute for a human to process and judge.

In theory, Cascade can handle tip-sets of arbitrary cardinality. The smallest set we've run is 22 items,[1] and the largest is close to 200 tips. The algorithm's expense grows super-linearly in the number of tips, so although there is no theoretical limit to the size of the tips-set, it does become more expensive. Future work can be spent on optimizing the cost for large tip-sets.

Output

The output of Cascade is a *taxonomy* consisting of labeled *categories* and associated tips. More precisely, Cascade generates a tree whose nodes are labeled with a textual string, called a category; the tree's root is labeled with the topic input and an '*other*' node is added as a child of the root if necessary. Cascade appends categories to the tree root and appends categories to non-root categories. The depth of tree Cascade produces is often greater than one. It is important to note that Cascade allows tips to appear in the taxonomy in multiple categories. Tips generated in the wild often span categories. For example, it is easy for a travel tip to be both about "air travel" and "saving money."

[1] With fewer than 22 items it seems unlikely that one would want a taxonomy or would need a crowd to do it.

Parameters

Cascade's behavior is guided by a set of parameters, which we now summarize and name our default values where applicable.

- Let n be the number of tips input.
- Let $m \leq n$ denote the number of tips considered in Cascade's initial, generative pass, default = 32
- Let k be the *replication factor* (the number of workers who may be asked to repeat a step), default = 5.
- Let t be the maximum number of tips shown to a worker at once, default = 8.
- Let c denote the maximum number of categories a user is asked to consider when selecting the best tag, default = 5.
- Let s be the maximum number of categories a user is asked to judge the relevance of, default = 7.

By changing the values of these parameters, the designer can trade off cost and running time against taxonomy quality. Decreasing k or increasing t, c, and s will lower the cost of execution.

Primitive Worker Tasks

Before describing Cascade's overall control flow, we first present the three types of HITs that are presented to workers. The order in which these tasks are generated is dependent on the characteristics of the input tips and can be complex, but the primitives are individually quite simple. Here we present them abstractly; Figure 6 displays concrete instances of these tasks.

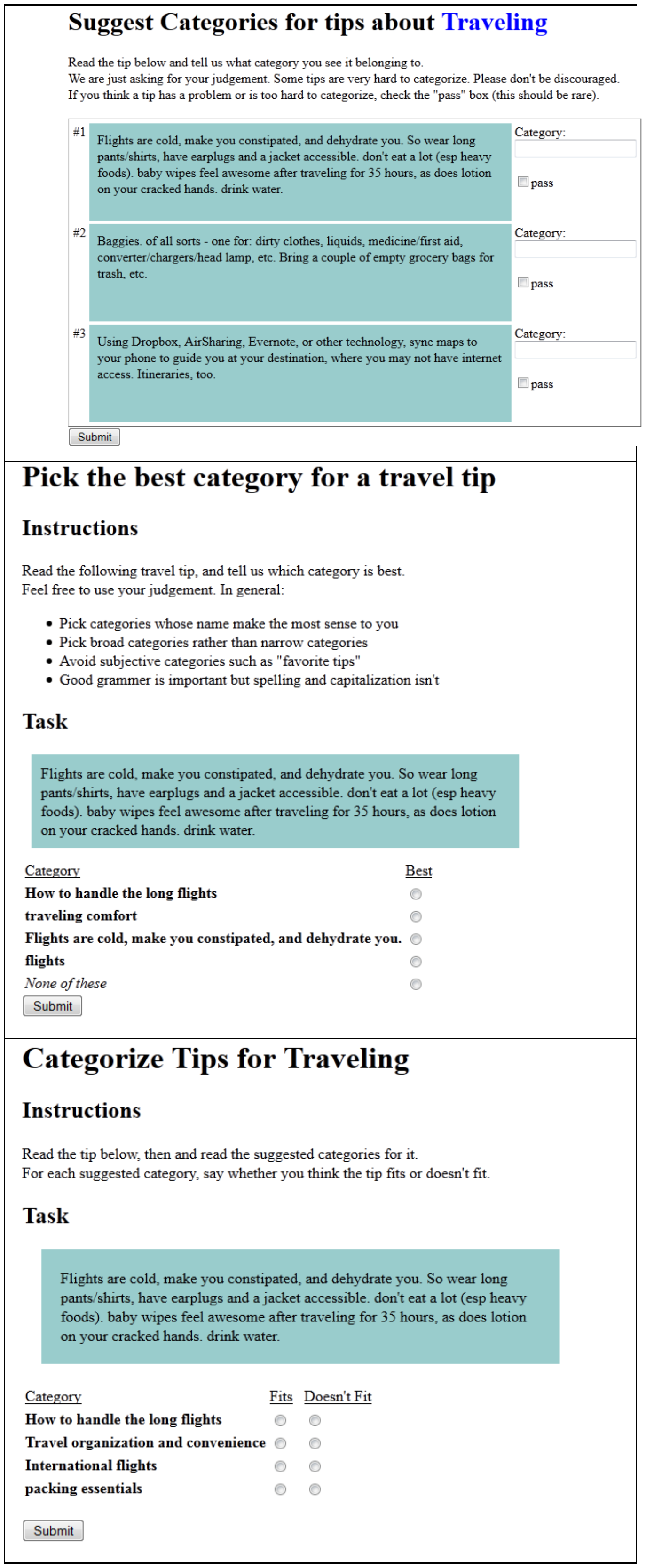
- *Generate*(t tips) -> t categories
  The Generate HIT presents a worker with t different tips and asks her to generate one suggested category tag for each tip; the categories don't need to be distinct, but they often are.

- *SelectBest*(1 tip, c categories) -> 1 category
  The SelectBest HIT shows a worker a single tip and c distinct category tags and asks her to pick the single best tag.

- *Categorize*(1 tip, s categories) -> bit vector of cardinality s
  The Categorize HIT is similar to SelectBest, but is used at a different point in the Cascade process. It presents a worker with a single tip and s possible category tags and asks the worker to select *all* of the categories which are relevant to the tip.

Algorithm Steps

Described at the highest level, the Cascade algorithm takes every tip and solicits multiple suggested categories for it from different workers. A new set of workers then votes on the best suggested category for each tip. Cascade then asks workers to categorize every tip into every best suggested category. With that data, it computes a taxonomy where duplicate categories are removed, empty categories are removed and related categories are nested appropriately.



Suggest Categories for tips about Traveling

Read the tip below and tell us what category you see it belonging to.
We are just asking for your judgement. Some tips are very hard to categorize. Please don't be discouraged.
If you think a tip has a problem or is too hard to categorize, check the "pass" box (this should be rare).

#1	Flights are cold, make you constipated, and dehydrate you. So wear long pants/shirts, have earplugs and a jacket accessible. don't eat a lot (esp heavy foods). baby wipes feel awesome after traveling for 35 hours, as does lotion on your cracked hands. drink water.	Category: ☐ pass
#2	Baggies. of all sorts - one for: dirty clothes, liquids, medicine/first aid, converter/chargers/head lamp, etc. Bring a couple of empty grocery bags for trash, etc.	Category: ☐ pass
#3	Using Dropbox, AirSharing, Evernote, or other technology, sync maps to your phone to guide you at your destination, where you may not have internet access. Itineraries, too.	Category: ☐ pass

Submit

Pick the best category for a travel tip

Instructions

Read the following travel tip, and tell us which category is best.
Feel free to use your judgement. In general:

- Pick categories whose name make the most sense to you
- Pick broad categories rather than narrow categories
- Avoid subjective categories such as "favorite tips"
- Good grammer is important but spelling and capitalization isn't

Task

Flights are cold, make you constipated, and dehydrate you. So wear long pants/shirts, have earplugs and a jacket accessible. don't eat a lot (esp heavy foods). baby wipes feel awesome after traveling for 35 hours, as does lotion on your cracked hands. drink water.

Category	Best
How to handle the long flights	○
traveling comfort	○
Flights are cold, make you constipated, and dehydrate you.	○
flights	○
None of these	○

Submit

Categorize Tips for Traveling

Instructions

Read the tip below, then and read the suggested categories for it.
For each suggested category, say whether you think the tip fits or doesn't fit.

Task

Flights are cold, make you constipated, and dehydrate you. So wear long pants/shirts, have earplugs and a jacket accessible. don't eat a lot (esp heavy foods). baby wipes feel awesome after traveling for 35 hours, as does lotion on your cracked hands. drink water.

Category	Fits	Doesn't Fit
How to handle the long flights	○	○
Travel organization and convenience	○	○
International flights	○	○
packing essentials	○	○

Submit

Figure 6.Primitive HIT Interfaces –top to bottom: Generate, SelectBest and Categorize

A slightly more formal description of the Cascade workflow is as follows:

1) use a *subset* of m tips to generate a candidate set of category labels,

2) filter this candidate set to produce a smaller set of the very best labels,

3) iterate over the initial subset of tips to tag them with every category – appropriate label using the best labels generated in the previous step,

4) run a fully-automated process to determine which categories are best considered to be children (subcategories) of another category,

5) iterate over all of the n tips (not just the initial m used previously) to assign relevant category tags to *every* tip,

clean up newly-processed tips that have no applicable categories. Cascade uses steps 1-4 to generate new category labels for these tips.

Cascade concludes by iterating over the complete set of n tips to label them with the new categories where appropriate.

The rest of this section explains how these steps are implemented using a combination of automated algorithms and the Generate, SelectBest and Categorize HITs. Step 3, which we term *adaptive context filtering*, is the most expensive and also the most novel, since it ensures globally meaningful results from individual contributors with a local perspective.

Step 1. Intentional Category Over-Generation

HIT Primitives used: *Generate* is called (ceil *m*/t) **k* times.
Output: *k* suggested categories for each of the *m* tips.

The first step of Cascade is to show each tip to *k*=5 different people, to tell them the topic the tip belongs to, and to have them suggest a category it might belong in. Although tips could be presented individually, we present tips in groups of *t*=8 using the HIT primitive Generate. The instructions explain that although multiple tips are displayed together, category suggestions should be independent, meaning that workers do not have to try to come up with categories that apply to multiple tips in that HIT.

By showing a group of tips in each HIT, workers get more context about the tip-set as a whole. This is useful particularly for a worker's first HIT in the tip-set, by providing the worker with an idea of the spread in the tips before she suggests a category. Another benefit of showing multiple tips in a group is that workers can easily "pass" on one or two tips in the HIT that are difficult to categorize (or have errors that make them not worthwhile to categorize – being off topic or spam.)

Step 2. Best Category Suggestion Vote

HIT primitives used: *SelectBest* is called *m* * *k* times.
Output: a set of best suggested tags (of cardinality ~1.5*m*)

In Step 1, k=5 individual workers attempted to categorize each tip, resulting in up to five *suggested categories* for each tip. There will be fewer than k=5 suggested categories if one or more workers "passed" on the tip or categories were removed for being exact-string duplicates. In Step 2, we show each worker one tip and all of its suggested categories and ask workers to pick the one they think is best (or select "None" meaning that none of them are good). Any suggested category that gets two or more votes will get passed on to the next step. We call these the *best suggested categories.* The suggested categories that do not meet the threshold will be filtered out by the algorithm.

Step 3. Adaptive Context Filtering

HIT primitives used: *Categorize* is called m*ceil(|tags|/t)*k + m*k times

Output: a list of *best suggested categories* and the tips that fit in them

Phase 1

In Step 1 we generated more categories than we needed for each tip. In Step 2, we filtered out categories that were not necessary for each tip. In Step 3, we filter out categories that are not necessary for the tip set as a whole by asking workers to vote whether each tip fits each of the best suggested categories. To do this, we present workers with one tip and a group of s=7 best suggested categories and for each best suggested category they have to choose whether they think it "fits" or "doesn't fit." If two or more out of five workers agree that a tip fits one of the best suggested categories, then it passes the filter and goes to the next phase of Adaptive Context Filtering.

Phase 2

The results of Phase 1 give us a list of potentially applicable categories for each tip. These categorization decisions are deceptively difficult – tips range in quality and in clarity, and *best suggested categories* range in how vague they are. It is easier to make difficult categorization decisions when the group of categories presented together are all potentially applicable, as opposed to the applicable categories being spread very sparsely across hundreds of HITs. Phase 1 of this step aggregates the potentially applicable categories for each tip, and now in Phase 2, we categorize again on those potentially applicable categories. Although we are using the same HIT primitive (*Categorize*), the context we are asking it in (i.e., the group of other best suggested categories displayed around it) has improved, which allows workers to be more discriminating. If four or five out of five workers agree that a tip fits one of the applicable categories, then it passes the filter and goes to the next step of Cascade.

Step 4. Edit-Matrix Operations

No HITs

Output: a taxonomy

After Step 3, we compile a matrix that says for all tips, for all *best suggested categories*, which tips belong in which *best suggested categories*. We use this matrix to edit down the list of best suggested categories in the following ways:

1. *Remove duplicate categories*. For any two categories that share more than 75% of their tips, we remove the category with fewer tips (or at random in the case of a tie)
2. *Create nested categories*. For any category, *c_small* that shares more than 75% of its tips with another category, *c_large*, make *c_small* a subcategory of *c_large*.
3. *Remove categories that are too small.* Remove any category that has fewer than two tips.

This results in a taxonomy where all categories have at least two items, sibling categories are distinct, and subset categories are properly nested under their super category.

Iteratively Running Cascade

When you apply Cascade to a tip-set there are several reasons you might not want to run it on all the tips at once. One reason is that the tips may be generated dynamically and as new tips come in, you want Cascade to update its taxonomy. The second reason is that you may try to save time and money by applying Cascade to a subset of the data, then deciding later whether to run the rest of the tips depending on how "done" you think the taxonomy is. Whatever the reason, Cascade can be run on a hold-out set of tips to grow the taxonomy as follows:

	Running time	# HITs	Cost
Step 1 : Intentional Category Over-Generation	ceil (m/t) *k	20	$3.20
Step 2: Best Category Suggestion Vote	m * k	160	$3.20
Step 3. Adaptive Context Filtering – Phase 1	m*ceil(|tags|/s)*k	1100	$22
Step 3. Adaptive Context Filtering – Phase 2	m*k	160	$3.20
Step 5. Categorize Hold-outs	(n-m)* ceil(|cats.|/t)*k	320	$6.40
Iteration 1 total		1760	$38.00
Iteration 2 total		1760	$38.00
Total:		3520	$76.00

Table 1 Running Time of Cascade with n=64, m=32 and other values at their stated typical value.

Step 5. Categorize Hold-outs
HIT primitives used: Categorize is called ceil((n-m)/t)*k times
Output: an updated Edit-Matrix

Categorize the hold-out set of tips on the existing categories.

Step 6. Update Taxonomy
No HITs
Output: an updated Taxonomy

Rerun Step 4 on the new Edit-Matrix to produce a taxonomy that includes all the hold-out tips.

Step 7. Generate Iterative tip-Set
No HITs
Output: a tip-set

From the taxonomy, determine which of the hold-out tips are not in any category and which of the hold-out tips are only "loosely categorized." A tip is loosely categorized if it is not in any category that has fewer than 20 tips. 20 is the minimum number of tips we use in Cascade in order to recurse. The idea behind this step is that if a category that is exceedingly large ends up in the final taxonomy, we don't want to treat tips that are in it as being categorized to satisfaction. For example, the category "all tips" could be suggested. All the tips would be in this category, but it wouldn't contribution to our understanding of the data. This is actually a big problem. Workers often produce vague categories that have 60-70% of the data in them. In our "travel advice" tip-set the category "travel organization and convenience" had 68% of the tips, but clearly does not make a useful contribution to our understanding of how the data breaks down.

EXPERIMENTS

To test the performance of Cascade we run the algorithm on three datasets and present the taxonomies it produces.

Data
We ran Cascade on three datasets scraped from Quora.com, an online Question and Answer site reputed to be of high quality. Many of the questions on Quora are fact-based, such as "How much did it cost AOL to distribute all those CDs back in the 1990s?" which was answered by Steve Case[2]. But many questions have no single best answer and all the responses are valid answers to the question, such as "What are your best travel hacks?" These types of questions get many responses and it is time consuming to get a sense of what has been said. This is the type of domain where a taxonomy would help users get a global picture of the data and navigate the responses.

Quora has a wiki-like section at the top of its interface for users to summarize the answers, but from a cursory look, it does not seem to be used often or effectively. This is probably because it is hard to update as new responses appear and because the time and effort required to compile a fair, global picture of free-text responses is non-trivial.

The three datasets were picked are summarized in the table below.

Abbreviation	Topic	Number of tips
editWriting	"What are some tips for editing your own writing?"	22
sideProjects	"How can I increase my productivity on my side projects at the end of the day when I'm tired from work?"	67
travel	"What are your best travel hacks?"	100

Table 2. Topics and size of tip-set

Often, a single response will contain multiple tips in bulleted lists, numbered lists or separated by paragraphs. We manually broke these responses into their separate tips. We changed the text minimally to make individual tips readable by means of capitalization, removal of leading bullet points, and reiteration of pronouns ("I think…"). Previously, we have had the crowd do this breakdown. It is a trivial process but not a part of the Cascade Algorithm.

We randomized the order of the tips to avoid any effects of our workers seeing tips in the order they were generated.

Implementation
We implemented the primitive HITs in HTML and JavaScript to be used as externalQuestions on Mechanical Turk. To dispatch HITs, we used TurKit[10]. Python scripts were used to process data in between steps.

RESULTS

We ran Cascade on three tip-sets. The smallest, editWriting, required only one iteration of Cascade, starting with all the tips. The mid-sized tip-set, sideProjects, was first run with 32 tips (with a hold-out tip-set of 35 tips). The taxonomy produced in the first iterations was then applied to the hold out tip-set and there was an insufficient number of uncategorized or loosely categorized tips to run a second iteration of Cascade. The largest tip-set, travel, was first run with 32 tips, (with a hold-out tip-set of 68 tips). When categorizing the hold-out tips to the first iteration taxonomy, 51 tips were not adequately categorized, so a second iteration of Cascade was run.

Here are details of how Cascade ran on each tip-set, which will be discussed below.

editWriting
Cascade is meant to run on tip-sets with at least 20 items. With 22 tips, *editWriting* is the smallest tip set we ran

[2] http://www.quora.com/AOL-History/How-much-did-it-cost-AOL-to-distribute-all-those-CDs-back-in-the-1990s

Cascade on. This tip-set required only one round of Cascade, wherein we created a taxonomy for all 22 tips. None of the tips were uncategorized at the end and we ended up with 15 categories, 8 of which were top-level categories (i.e., children of the root node). To get to the final 15 categories, we started with 83 suggested categories, filtered that down to the 33 best suggested categories, then did exactly-string matching to filter that down to 27 unique best suggested categories. After applying the best suggested categories to all the tips, we removed 4 categories for having too much overlap (Table 3) and 8 categories because they had 0 or 1 tips in them (Table 4). The resulting taxonomy can be seen in Figure 7

Larger Category	Smaller Category	Overlap %
Self-Editing	Editing	82%
Read out loud	Read aloud	100%
getting help	asking for help editing	100%
working off an outline	Outlining	100%

Table 3. Overlapping categories for editWriting

Category	**# tips**
'Continuity and consistency'	1
'Edit when you are finished writing'	1
'Eliminate repetition'	1
'Know Your Limits'	1
'Put yourself in someone elses shoes'	1
'Reformat	1
'Write, Delete, Rewrite'	1
'story detail editing'	1

Table 4. Categories with fewer than 2 items for editWriting

sideProjects

sideProjects is a mid-sized tip-set with 67 items, of which 32 were used in the first round to generate a taxonomy with 22 categories. After generating the first-round taxonomy, we applied the remaining 35 tips to it and found that it explained all but 2 of the tips and had no loosely categorized tips and thus we did not need to start a second round of Cascade because no new categories were required to categorize the tips.

To get the final 22 categories in the first round of Cascade, we generated 120 suggested categories and filtered that down to the 37 best suggested categories. After doing exact-string matching we were left with 34 unique best suggested categories to apply the data to. After applying the 32 first-round tips to all 24 unique best suggested categories, we removed 2 categories for having too many associated tips, and nine categories which had too few (0 or 1) tips in them.

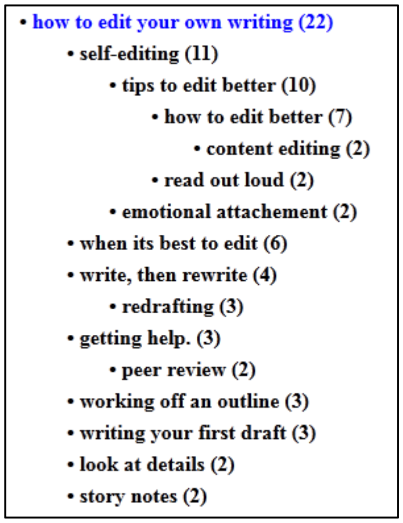
The resulting taxonomy can be seen in Figure 8.

	editWriting	sideProjects	Travel
# of tips total	22	67	100
# of tips used in round 1	22	32	32
Step 1 – Intentional Category Over-generation			
# of suggested categories	83	120	149
Step 2 – Best Category Suggestion Vote			
# best suggested categories	33	37	45
# unique best suggested categories	27	34	43
Step 4 – Edit Matrix Operations			
# of overlapping categories	4	2	7
# of empty/singleton categories	8	9	29
# of applicable categories	15	23	7
# of uncategorized tips	0	4	15
Step 5 – Categorize Hold-Out Tips			
# of tips applied	n/a	35	68
# of uncategorized tips	n/a	2	15
# of loosely categorized tips	n/a	0	51

Table 5 Cascade Iteration 1 details

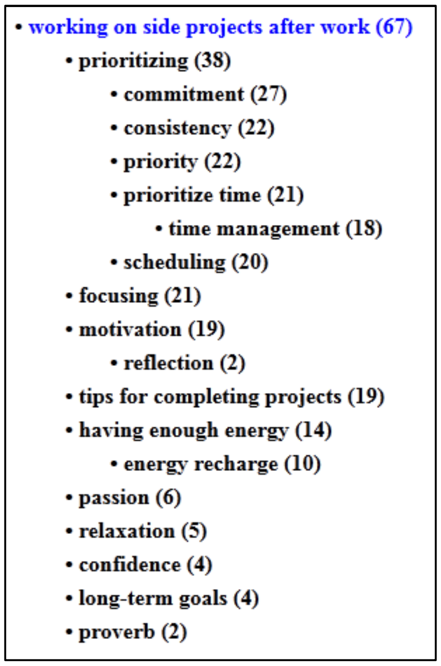
travel

Travel is a large tip set with 100 items, of which 32 were used in the first round to generate a taxonomy with 7 categories. After generating the first-round taxonomy, we applied the remaining 68 tips to it and found that it there were 15 tips which it did not categorize and 51 loosely categorized tips. The loose tips were all in the category "travel organization and convenience."



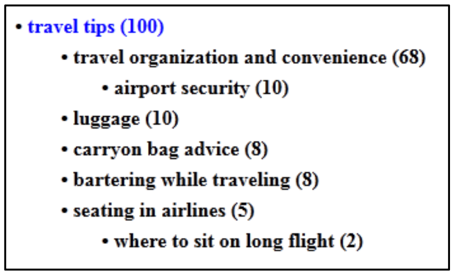
- how to edit your own writing (22)
  - self-editing (11)
    - tips to edit better (10)
      - how to edit better (7)
        - content editing (2)
      - read out loud (2)
    - emotional attachement (2)
  - when its best to edit (6)
  - write, then rewrite (4)
    - redrafting (3)
  - getting help. (3)
    - peer review (2)
  - working off an outline (3)
  - writing your first draft (3)
  - look at details (2)
  - story notes (2)

Figure 7. Taxonomy for editWriting



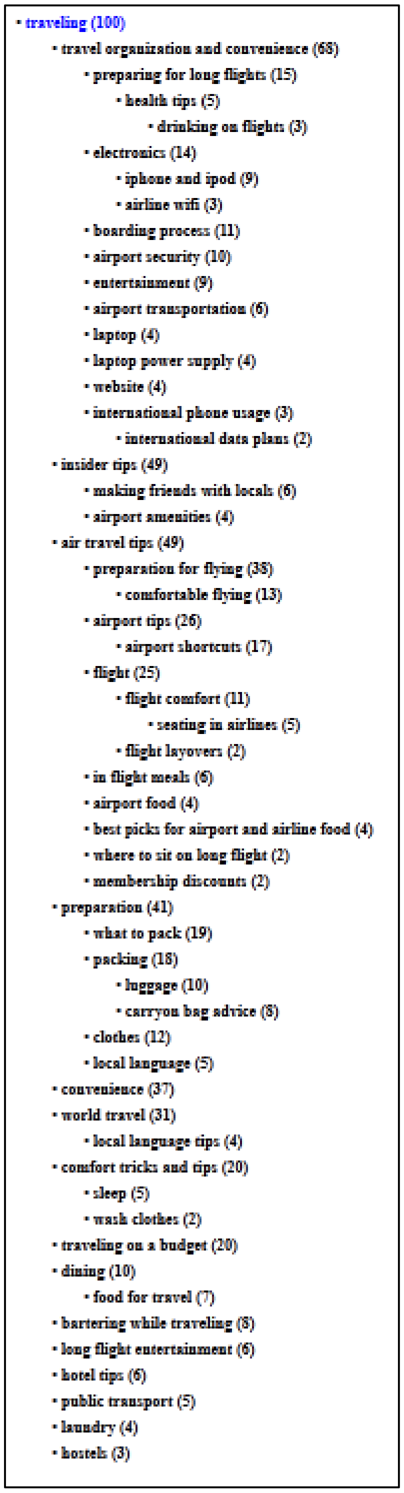
- working on side projects after work (67)
  - prioritizing (38)
    - commitment (27)
    - consistency (22)
    - priority (22)
    - prioritize time (21)
      - time management (18)
    - scheduling (20)
  - focusing (21)
  - motivation (19)
    - reflection (2)
  - tips for completing projects (19)
  - having enough energy (14)
    - energy recharge (10)
  - passion (6)
  - relaxation (5)
  - confidence (4)
  - long-term goals (4)
  - proverb (2)

Figure 8. taxonomy for sideProjects



- travel tips (100)
  - travel organization and convenience (68)
    - airport security (10)
  - luggage (10)
  - carryon bag advice (8)
  - bartering while traveling (8)
  - seating in airlines (5)
    - where to sit on long flight (2)

Figure 9. taxonomy for travel, iteration 1



- traveling (100)
  - travel organization and convenience (68)
    - preparing for long flights (15)
      - health tips (5)
        - drinking on flights (3)
    - electronics (14)
      - iphone and ipod (9)
      - airline wifi (3)
    - boarding process (11)
    - airport security (10)
    - entertainment (9)
    - airport transportation (6)
    - laptop (4)
    - laptop power supply (4)
    - website (4)
    - international phone usage (3)
      - international data plans (2)
  - insider tips (49)
    - making friends with locals (6)
    - airport amenities (4)
  - air travel tips (49)
    - preparation for flying (38)
      - comfortable flying (13)
    - airport tips (26)
      - airport shortcuts (17)
    - flight (25)
      - flight comfort (11)
        - seating in airlines (5)
      - flight layovers (2)
    - in flight meals (6)
    - airport food (4)
    - best picks for airport and airline food (4)
    - where to sit on long flight (2)
    - membership discounts (2)
  - preparation (41)
    - what to pack (19)
    - packing (18)
      - luggage (10)
      - carryon bag advice (8)
    - clothes (12)
    - local language (5)
  - convenience (37)
  - world travel (31)
    - local language tips (4)
  - comfort tricks and tips (20)
    - sleep (5)
    - wash clothes (2)
  - traveling on a budget (20)
  - dining (10)
    - food for travel (7)
  - bartering while traveling (8)
  - long flight entertainment (6)
  - hotel tips (6)
  - public transport (5)
  - laundry (4)
  - hostels (3)

Figure 10 taxonomy for travel, iteration 2

To get the final 7 categories in the first round of Cascade, we generated 149 suggested categories and filtered that down to 45 best suggested categories. After doing exact-string matching we were left with 40 unique best suggested categories to apply the data to. After applying the 32 first-round tips to all 40 unique best suggested categories, we removed 4 categories for having too much overlap, and 29 categories which had 0 or 1 tips in them.

The fact that one round of Cascade left 66 items unclassified gave us the opportunity to run a second round of Cascade. We reran Cascade on 45 tips – the 15 uncategorized tips and 30 of the 51 loosely categorized tips). This resulted in a taxonomy with 51 items. (Figure 10)

Observations:
The most important things to notice about the performance of Cascade is that in all three datasets, we started with many more tips than we intended to include in the final taxonomy, and effectively edited it down to a better and more cohesive set of categories.

Most of the categories that were eliminated for having fewer than 2 tips had 1 tip (only 1 of the 46 categories that was eliminated had 0 tips). This makes sense because one would expect each category to contain at least the one tip that originally generated it.

EVALUATION
The goal of Cascade is to produce a taxonomy that provides a global understanding of independent tips. There are three questions we want to answer to determine how well Cascade performs:

1. Are the category labels in the taxonomy as good as labels created by experts?
2. Do we create an appropriate hierarchical structure in the taxonomy?
3. Is the cost and running time of Cascade competitive with that of hiring experts?

Good Category Labels
Taxonomies are inherently subjective; there is no right answer. One would not necessarily expect two experts to produce the same hierarchy. However, given a small pool of experts independently categorizing a dataset, one would expect some of the same categories to appear in multiple experts' taxonomies. In order to compare Cascade's categories to those of experts, we paid four information architects to produce taxonomies independently for our three datasets.

We performed the following comparison on the taxonomies. For each data set, we took the Cascade-produced taxonomy, *taxC*, and the four expert taxonomies: *tax1-tax4*. We wanted to know two things:

1. What fraction of *taxC* categories are also named in *tax1-tax4*?
2. What fraction of *tax1-tax4*categories are named in another taxonomy in *tax1-tax4*?

We want to compare the fraction of *taxC* categories used by experts to the fraction of categories used by at least two experts for *tax1-tax4*. The comparison may seem slightly unfair in favor of *taxC* because *taxC* gets compared against 4 other taxonomies and *tax1-tax4*can only compare against 3 other taxonomies. However, Table 6 contains the results. For all three datasets, about 50% of Cascades categories were also named by an expert. For example, in the editWriting dataset, four out of four experts named a category closely matching Cascade's category "working off an outline." When comparing experts to themselves, the average expert matching fraction was 32%, 70%, and 64% for the three datasets. This averages to 55% of tips matching another expert's tips across these three hierarchies. Therefore, Cascade had 91% of the category agreement the experts did among themselves.

	edit-Writing	side-Projects	travel	Avg
# Cascade categories taxonomy	15	18	51	
% of Cascade categories shared by expert	47%	50%	53%	50%
Number of categories shared by 2+ experts	2	11	6	
Avg # expert categories	14	22	30	
Avg % of tips shared by 2+ experts	32%	70%	64%	55%

Table 6 Category name quality comparison – Cascade vs. Experts

Mistakes in Hierarchical Structure
Cascade infers a global understanding of the data from the tip membership of categories. Cascade removes categories that do not have enough tips in them, removes categories that have a high tip overlap, and creates a parent-child relationship for categories where one category has high tip overlap with the other. These inferences are based on many small judgments by potentially hundreds of different people. We want to know if all those judgments come together to infer a sensible hierarchy. In particular, we are looking for three types of mistakes in the Cascade hierarchies:

1. Duplicate categories
2. Missing Parent-Child Relationships
3. Incorrect Parent-Child Relationships

To find the error rate in the hierarchical structure, we divide the number of errors by the number of categories in the taxonomy. editWriting has the smallest error rate of 13% (Table 7), with only 2 errors in 15 categories. Both were duplicate categories errors. The categories "tips to edit better" and "how to edit better" should have been the same, but Cascade did not remove one of them.

	Edit-Writing	Side Projects	Travel: iteration 1	Travel: iteration 2
# categories	15	18	7	51
Duplicate Categories	2	2	0	2
Missing Nesting	0	0	0	5
incorrect Nesting	0	3	1	3
Correct Nesting	5	3	1	23
total errors	2	5	1	10
Error rate	13%	27%	14%	20%

Table 7 Error rate for structural mistakes in the hierarchy

sideProjects had the highest error rate of 27%. This came from 3 incorrect parent-child relationships: 'prioritizing' was the parent of 'commitment,' 'prioritizing' was the also parent of 'consistency,' and 'motivation' was the parent of 'relaxation.' In our judgment, there is no clear reason that prioritizing should be a parent of commitment or consistency, or that motivation should be the parent of relaxation, and thus it is a mistake in the hierarchical structure of the taxonomy. These are errors produced by the machine step – the Edit Matrix Operations - which created a parent-child relationship any time more than 75% of the tips of a smaller category were also in a larger category. Concretely, the Edit Matrix operations nested commitment under prioritizing because more than 75% of the tips about commitment were also about prioritizing. However, although these categories share many tips in common, they aren't semantically related: this is a danger of machine steps. Perhaps a solution would be to have humans check the resulting taxonomy for obvious errors.

Across the three datasets, the average error rate was 18.5%.

There was an impressive number of correct parent-child relationships, especially in the travel dataset. (23 correct parent child relationship and 3 incorrect ones). Many air-travel and flight related categories with complicated nesting are expressed with coherent hierarchical structure. For example, "Air Travel Tips" is a parent of "flights" which is a parent of "flight layovers."

	editWriting	sideProjects	travel
Cascade Time	7 h 56 m	16h 13 m	16h 32m
Cascade Cost	$35.40	$109.45	$224.45
Avg Expert Time	1h 23 m	2h 36m	2h 5 m
Average Expert Cost	$34.87	$65.13	$71.38

Table 8 Time and Cost Comparison - Cascade vs. Experts

Time and Money

It is non-trivial to compare the costs associated with creating a taxonomy with Cascade versus experts. There is a cost-quality-time trade-off. For example, on MTurk, if you under-price a HIT, it will eventually get done, but it will take a long time. The most basic comparison we provide is the actual costs and times in our run of Cascade and that of our recruited experts (Table 8). Cascade took ~6.5 times longer to complete the HITs, and was 1-3 times as expensive. However, the prices were set fairly arbitrarily. We paid our experts $25/hour as a set wage. We paid MTurk workers $0.05 per HIT. The average time to complete a HIT was 21.46 seconds. This equates to $8.39/hour which is high for MTurk. $3-$4 an hour would be more expected. That would reduce the cost of Cascade by a factor of 2, making Cascade's cost competitive with the wage we offered experts.

Comparing time is also difficult. The total time spent on all three datasets by the average expert was 6 hours and 50 minutes. And the total time spent by MTurk workers was 43 hours and 3 minutes. This is a factor of 6.3 more time spent by MTurk workers. Seeing as the work done by workers is basically replicated *k*=5 times over, the time it would take one person to run Cascade on themselves would be competitive with the expert's time.

More important than comparing total time spent on the algorithm is to think about the amount of time that it would take to run the algorithm if infinitely many people work in parallel, as is supported by Cascade. Each worker spends on average 21.3 seconds per HIT, and all the HITs in any step can be run completely in parallel. Thus, assuming Cascade is run in two iterations of 5 steps each, the entire time it would take to run Cascade would be 3 minutes and 33 seconds

DISCUSSION

Cascade is driven not only by human judgments, but by human judgments based on other human judgments. Since humans are difficult to predict, it is impossible to guarantee how Cascade will perform with different worker populations. In the worst case, the workers could be unfamiliar with the domain and not generate any useful category suggestions in the first step. If that happened,

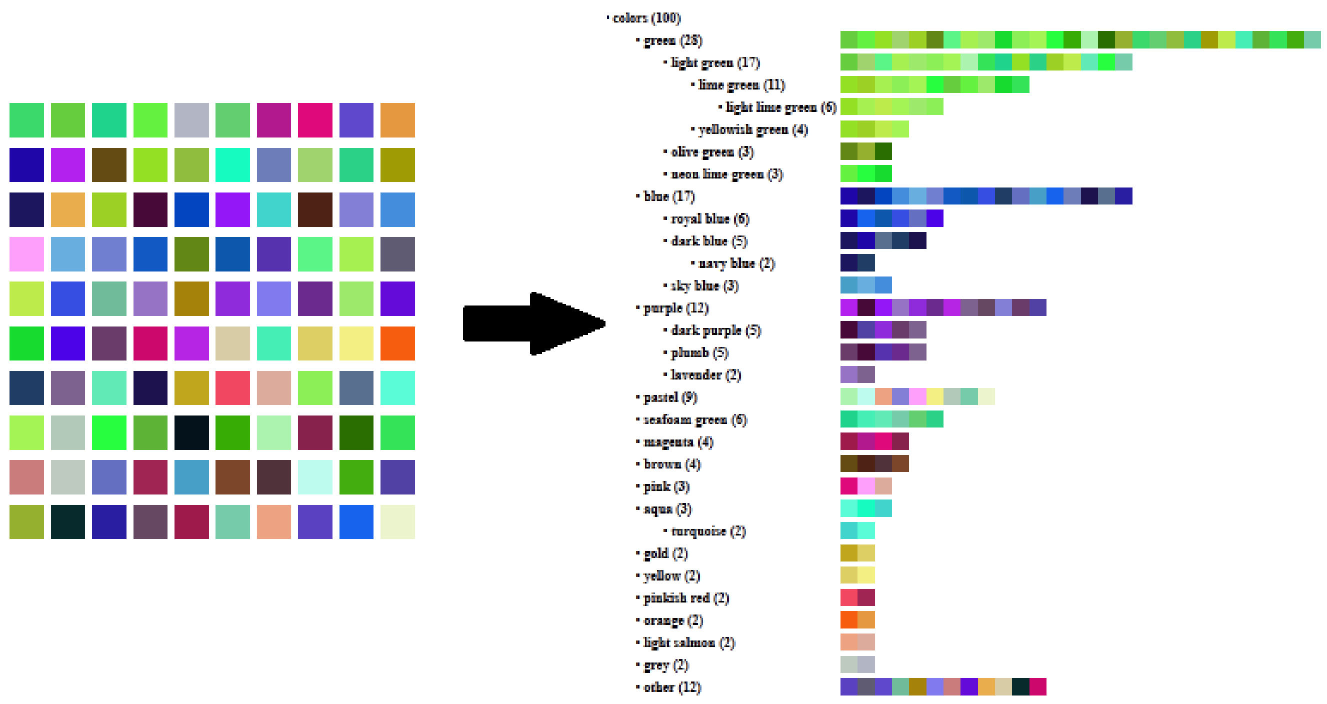


Figure 11 Cascade applied to 100 randomly generated colors. On the left is the input to Cascade - 100 colors . On the right is the output of Cascade – a taxonomy organizing the colors. Colors can appear in multiple categories, and 12 of the 100 colors are in the "other" category – meaning they were not categorized into a category with at least two items.

Cascade would not produce any taxonomy at all beyond the root node. If the categories were good, but workers voted erratically, the resulting taxonomy would have essentially random structure – repeated categories, parents and children that do not relate semantically, and missing parent-child relationships.

The worst case will always be bad for crowdsourcing problems, but the performance of Mechanical Turk workers represents a reasonable lower bound for the quality of the resulting taxonomy. Given a community that cared about the data and had domain specific knowledge, they would probably do it very well. Moreover, we believe that this task is as fun for some people as the ESP game and other game-based crowdsourcing and thus people will probably be willing to do it for free. Every step involves reading tips and the tips are interesting. It is hard to read one without wanting to read more. There is probably potential to game-ify the interface to encourage contributions.

Cascade has five steps, including two slightly unusual steps – over-generation and adaptive context filtering. In this paper we did not seek to prove that these steps are required. It seems plausible that we could just generate n categories, categorize all the tips and then do the edit-matrix step. These two steps are the result of design decisions we made based on running the algorithm many different ways early in its development. Things we noticed that encouraged us to keep these two steps. First, the quality of the taxonomy is most strongly correlated with the quality of the categories. If we did not over-generate tags, then ask people to pick the best, we would have ended up with lower quality categories. In particular, we often end up with vague categories which are problematic for Cascade. Vague categories contain a lot of members. For example, in the travel dataset, vague categories might be something like "comfort and convenience" or "organization and advice." These categories encompass most, but not all of the tips. And are indistinguishable from large categories which are not vague, such as "air travel." A combination of trying to filter out vague categories in the Best Suggested Categories phase and in Phase 2 of adaptive context filtering is the design that we settled on.

Although the evaluation in this paper only deals with text data, we have applied Cascade to visual data as well. Figure 11 is an example of creating a taxonomy for 100 randomly generated colors. Part of the future work for Cascade is to push the boundaries of what types of data humans can taxonomize For example, can we create taxonomies for images, audio clips, videos, and xixed media such as websites? Nothing about the Cascade algorithm is particular to text. We believe that any data type that humans can process will be applicable to use Cascade on.

RELATED WORK

Crowdsourcing complex tasks

In the past three years, there have been several crowd workflows that produce outputs more complex than the results of worker's local contributions.

TurKit[10] is a programming environment that allows you to easily compose the results of tasks and issue new tasks built upon the previous tasks. Turkit has been used to iteratively improve image descriptions, to pick the best photo from an album, and to decipher nearly-unintelligible handwriting. These iterative tasks are a step beyond simple image labeling [1]. Much work was able to build upon this simple iterative framework [2][3][5].

Considering workflows that go beyond TurKit, CrowdForge[7] uses a MapReduce-like framework for writing articles by mapping separate workers to different aspects of the article (e.g., outline, the facts, the quotes, etc.) and then composing the results in a reduce-step. Mobi[12] solves problems like travel planning that have global constraints which are met by workers creating to-do items for other workers to do. Turk-o-matic[9] asks workers to break down the task and then creates subtasks for more workers to do. Real-Time Audio Capture [11] uses a combination of novel interface and sequence alignment to combine work. Complex tasks can be tracked and managed [8].

Card Sorting

Card Sorting[6] is a technique for members of a group to contribute to an organization of their data. Today, card sorting is often used for employees to influence the knowledge architecture of their intranet, or for supermarkets to organize their produce in ways that shoppers think about things (for example, putting peanut butter next to jelly). Card sorting is an investigative technique. It is not designed to give output a usable categorization. It is designed to help knowledge architects understand their target users' mental models, and it requires a moderator to digest all the work participants do.

Automated Approaches

Automated text-clustering such as LDA[4] could be employed recursively to create hierarchical taxonomies. One drawback of these automated approaches is that they tend to work best on very large datasets - 50 to 500 short responses are insufficient. In practice, AI clustering algorithms require substantial tuning, e.g., manually removing stop words and choosing the number of categories. Additionally, there are categories that LDA would not be able to produce because they are not based strictly on the text. For example, LDA would not be able to create clusters that distinguish jokes with observational humor from jokes with puns because similarities within the groups are not present in the words, but are properties of the meaning as a whole.

CONCLUSION

In this paper we present a crowd-algorithm that produces a taxonomy for a set of independent data items, such as travel tips or ideas for how to edit your own writing or strategies for working on personal projects after work. We show that using three HIT primitives – Generate, SelectBest, and Categorize, we can create an algorithm where each worker can do as little as 20 seconds of work and produce a taxonomy competitive in price and quality with expert information architects, but which will require more total time put in by people, mainly due to the replicative factor we use to ensure the crowd agrees on judgments.

REFERENCES

1. von Ahn, L., Dabbish, L. Labeling Images with a Computer Game. *CHI 2004*
2. Bernstein, M., et al.. Soylent: A Word Processor with a Crowd Inside.UIST 2010
3. Bigham, Jeffrey P., et al.. VizWiz: Nearly Real-time Answers to Visual Questions. UIST 2010
4. Blei, D. M., Ng, A. Y., Jordan, M. I. Latent dirichlet allocation. *The Journal of Machine Learning Research. Volume 3*, 3/1/2003. Pages 993-1022.
5. Dai, P., Mausam, Weld, D.S. Decision-Theoretic Control of Crowd-Sourced Workflows. *AAAI 2010*.
6. Hudson, William (2012): Card Sorting. In: *Soegaard, Mads and Dam, Rikke Friis (eds.). "Encyclopedia of Human-Computer Interaction". Aarhus, Denmark: The Interaction-Design.org Foundation.*
7. Kittur, A., Smus, B., Khamkar, S., Kraut, R. E. CrowdForge: crowdsourcing complex work. *UIST 2011*.
8. Kittur, A., Khamkar, S., Kraut, R.E. (in press). CrowdWeaver: Visually managing complex crowd work. CSCW 2012
9. Kulkarni, A., Can, M., Hartmann, B. Collaboratively crowdsourcing workflows with turkomatic. *CSCW 2012*.
10. Little, G., Chilton, L. B., Goldman, M., Miller, R. C. TurKit: human computation algorithms on mechanical turk. *UIST 2010*.
11. Walter S. Lasecki, etal . Online Sequence Alignment for Real-Time Audio Transcription by Non-Experts (UIST 2012).
12. Zhang, H., Law, E., Miller, R., Gajos, K., Parkes, D., Horvitz, E. Human computation tasks with global constraints. *CHI 2012*.