FeatureHub: Towards collaborative data science

Micah J. Smith
MIT, LIDS
Cambridge, MA
micahs@mit.edu

Roy Wedge
MIT, LIDS
Cambridge, MA
rwedge@mit.edu

Kalyan Veeramachaneni
MIT, LIDS
Cambridge, MA
kalyan@csail.mit.edu

Abstract—**Feature engineering is a critical step in a successful data science pipeline. This step, in which raw variables are transformed into features ready for inclusion in a machine learning model, can be one of the most challenging aspects of a data science effort. We propose a new paradigm for feature engineering in a collaborative framework and instantiate this idea in a platform, FeatureHub. In our approach, independent data scientists collaborate on a feature engineering task, viewing and discussing each others' features in real-time. Feature engineering source code created by independent data scientists is then integrated into a single predictive machine learning model. Our platform includes an automated machine learning backend which abstracts model training, selection, and tuning, allowing users to focus on feature engineering while still receiving immediate feedback on the performance of their features. We use a tightly-integrated forum, native feature discovery APIs, and targeted compensation mechanisms to facilitate and incentivize collaboration among data scientists. This approach can reduce the redundancy from independent or competitive data scientists while decreasing time to task completion. In experimental results, automatically generated models using crowdsourced features show performance within 0.03 or 0.05 points of winning submissions, with minimal human oversight.**

I. Introduction

As organizations enjoy a growing ability to store, process and retrieve recorded data, many seek mechanisms that will help them derive value from these data stores. Such stores may include clickstreams from customer interactions with online or digital platforms, signals captured by wearables, or data from complex interconnected systems like on-demand car services or transit networks. Organizations are typically interested in using this data to predict outcomes in order to personalize customer experiences, streamline existing systems, and make day-to-day life better and smoother.

To build predictive models that enable these goals, data scientists engage in a rather lengthy process of ideating and writing scripts to extract explanatory features from the data, in a process called *feature engineering*. These features are measurable quantities that describe each entity. For example, in an online learning platform, quantities that describe a learner's interactions include the amount of time the learner spent on the platform during one particular week, the number of visits she made to the website, the number of problems she attempted, and so on. When attempting to solve a particular prediction problem, a small, easily-interpretable subset of features may prove to be highly predictive of the outcome. For example, the average pre-deadline submission time is highly predictive of a MOOC student's stopout [1]. To identify and extract these features requires human intuition, domain expertise, sound statistical understanding, and specialized programming ability. Successful practitioners contend that feature engineering is one of the most challenging aspects of a data science task, and that the features themselves often determine whether a learning algorithm performs well or poorly [2], [3].

Of the different steps involved in predictive modeling, we argue that feature engineering is most susceptible to a collaborative approach — whether within an in-house data science team or as a crowdsourced modeling effort. Ideation by multiple people with different perspectives, intuition, and experiences is likely to yield a more diverse and predictive set of features. Then, implementing these features in code can be done in parallel, as long as well-designed abstractions are in place to ensure that the features can be combined in a single framework. On the other hand, one expects that preprocessing or modeling be characterized by either diminishing returns to simultaneous work or redundant work from independent data scientists, as these tasks typically require a single end-to-end solution and cannot be trivially parallelized.

Despite the benefits of a collaborative model, no system currently exists to enable it. The advances and successes of collaborative models in software development, such as version control systems and pair programming, motivates us to create such a system for feature engineering. Such a system will be distinguished by the ability to parallelize feature engineering across people and features in real-time and integrate contributions into a single modeling solution.

To build such a system, we must:

- break down the data science process into well-defined steps,
- provide scaffolds such that data scientists can focus on the generation of features,
- provide automated methods for the steps that are *not* feature engineering, and
- provide the functionality to view, merge, and evaluate combined efforts.

In this paper, we present FeatureHub[1], our first effort towards a collaborative feature engineering platform. With this platform, multiple users are able to write scripts for feature extraction, and to request an evaluation of their proposed features. The platform then aggregates features from multiple

[1]FeatureHub is available at https://github.com/HDI-Project/FeatureHub.

users, and automatically builds a machine learning model for the problem at hand.

Our contributions through this paper are as follows:

1) Propose a new approach to collaborative data science efforts, in which a skilled crowd of data scientists focuses creative effort on writing code to perform feature engineering.
2) Architect and develop a cloud platform to instantiate this approach, in the process developing scaffolding that allows us to safely and robustly integrate heterogeneous source code into a single predictive machine learning model.
3) Present experimental results that show that crowd-generated features with an automatically trained predictive model can compete against expert data scientists.

The rest of the paper is organized as follows. Section II describes a typical data science process, and how FeatureHub fits into such a workflow. Sections III and IV provide details on the platform and software architecture. In Section V, we present our experimental setup for user testing, in which crowdsourced data scientist workers work with real-world datasets. We discuss our results in Section VI. Section VII describes related work in the context of our contributions. Finally, Section VIII concludes.

II. Overview

In this section, we motivate the use of FeatureHub for collaboration on a prediction problem. Consider an example of predicting to which country users of the home rental site Airbnb will travel [4]. The problem includes background information and a dataset, in relational format, containing information on users, their interactions with the Airbnb website, and their potential destinations. Data scientists are asked to make predictions such that their five most highly ranked destinations match the actual destination of each user as closely as possible. (This dataset is described in more detail in Section V.) Under the approach facilitated by FeatureHub, the coordinator first prepares the prediction problem for feature engineering by taking several steps. She deploys a FeatureHub instance or uses an already-running system. She uploads the dataset and problem metadata to the server, and then performs some minimal cleaning and preprocessing to prepare the dataset for use in feature engineering. This includes identifying the `users` table as containing the entities instances, moving the target `country_destination` column into a separate table, splitting the data into a training and validation set, and more.

The coordinator steps back and the data scientists log in. On the platform, they can read background information about the problem, load the dataset, and conduct exploratory data analysis and visualization. When they are familiar with the problem and the dataset, they begin writing features. One data scientist may use her intuition about what aspects of countries are most appealing to travelers, and write a set of features that encode whether the user speaks the language spoken in different destinations. Another data scientist may look for patterns hidden deep within users' clickstream interactions with the Airbnb site, and write a feature that encodes the number of actions taken by a user in their most recent session. Once these users have written their features, FeatureHub automatically builds a simple machine learning model on training data using each feature, and reports important metrics to the data scientists in real-time. If the predictive performance of a feature meets expectations, the feature can be submitted and "registered" to a feature database, at which point performance on the unseen test set is also assessed. Though these data scientists are looking for signal in different places, their work can be combined together easily within FeatureHub's scaffolding. They may be following their ideas in isolation, or using integrated collaboration and discussion tools to split up work and exchange ideas.

At this point in a typical data science workflow, data scientists working independently might have accumulated several features, and, having spent much time on preparing a working environment and cleaning data, would be anxious to test the performance of their model. They might specify several machine learning models in order to get a sense of baseline performance and compute cross-validated metrics. As they continue ideating and doing feature engineering, they might take successively longer pauses to focus on model training and selection. However, using FeatureHub, individual workers can focus their creative efforts entirely on writing features while the system takes care of evaluating performance.

Meanwhile, the coordinator is monitoring the progress of workers. Each time they register a new feature, a model is selected and trained completely automatically and the performance is reported to the coordinator. After three days of feature engineering, the coordinator finds that the model has crossed a suitable threshold of performance for her business purpose. She notifies the data scientists that the feature engineering has concluded, and they move on to a new task.

III. FeatureHub Workflow

In this section, we discuss a typical FeatureHub workflow in detail and describe how both coordinators and feature creators interact with the platform. The workflow is divided into three phases: Launch, Create and Combine. Coordinators execute Launch and Combine, and feature creators interact with the platform in the Create phase.

A. Launch

In the Launch phase, a problem coordinator initializes a FeatureHub problem. The coordinator starts with a prediction problem that they want to solve, along with a dataset in relational format. Next, they perform preprocessing on the dataset to extract important metadata, including the problem type, the target error metric used to evaluate solutions, and the URLs and names of data tables. The coordinator also has the option to pre-extract a set of basic features that can be used to initialize the machine learning models. Often, these are features that require the most minimal and obvious transformations, such as one-hot encodings of categorical variables

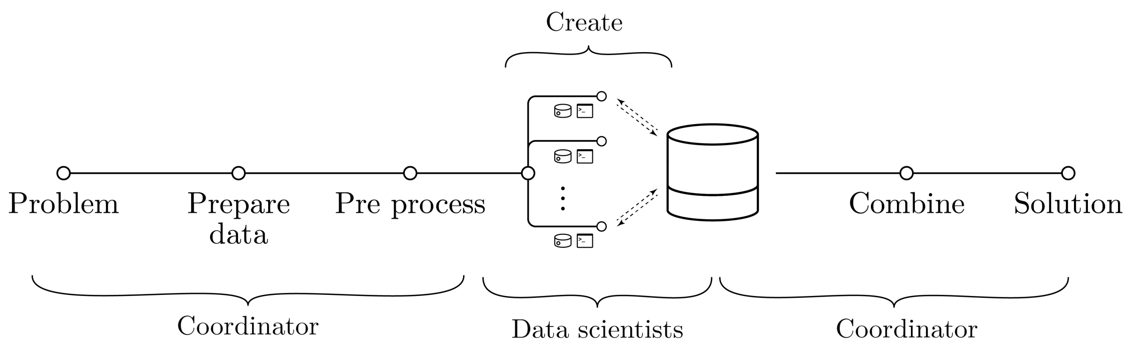


Fig. 1. Overview of FeatureHub workflow. A coordinator receives the problem to be solved, and associated data. She prepares the data and does the necessary preprocessing using different functions available. Then, the platform is launched with the given problem and dataset. Data scientists log in, interact with the data, and write features. The feature scripts are stored in a database along with meta information. Finally, the coordinator automatically combines multiple features and generates a machine learning model solution.

or conversions of string-encoded dates to timestamps. In fact, this entire step could be automated using existing frameworks. Finally, in order to onboard workers, the coordinator prepares a description of the prediction problem, the data tables and their primary key-foreign key relationships, and details of the pre-extracted features.

This requires that all relevant tables are available from the outset. One might argue that a key contribution of a talented feature engineer may be to ingest and merge previously-unknown, external data sources that are relevant to the problem at hand. For our purposes, we include this in the data preprocessing step.

B. Create

In the Create phase, data scientists log into a server using credentials provided by the coordinator. Their working environment is the Jupyter Notebook, a widely used interactive document that contains explanatory text, live code, and inline visualizations.

Load and explore: Notebooks for each problem contain detailed problem information provided by the coordinator, as in Section III-A. In this self-contained environment, users have all of the information they need to understand the problem and engage with the available data. The environment also comes pre-loaded with all of the packages users require for data science and feature engineering, including the FeatureHub client library. After reading through the problem setup, users import a FeatureHub client, which provides a minimal but powerful set of methods for interacting with the feature engineering process (see Table I).

Data scientists then load the data into their workspace by calling `get_dataset`. This populates an object that allows access to, and provides metadata for, each table in the relational dataset. Users can then explore the dataset using all of the familiar and powerful features of the Jupyter Notebook.

Write features:

We ask workers to observe a basic scaffolding of their source code when they write new features, to allow us to standardize the process and vet candidate features. This setup is crucial in allowing us to safely and robustly combine source code of varying quality.

```python
def hi_lo_age(dataset):
    """Whether users are older than 30 years"""
    from sklearn.preprocessing import binarize
    threshold = 30
    return binarize(dataset["users"]["age"]
        .values.reshape(-1,1), threshold)
```

Fig. 2. A simple feature. The input parameter (here, `dataset`) is an object with a mapping of table names to `DataFrame` objects. This function imports external packages within its body and returns a single column of values.

In this scaffold, a feature maps the problem dataset to a single column of numbers, where one value is associated with each entity instance. This format is suitable for many supervised learning algorithms, which expect a feature matrix of this form as input. At the outset, this informal definition seems to disallow categorical features or other "logical features" that consist of multiple columns. However, these can be represented simply as numerical encodings or encoded one column at a time, in the case of one-hot encodings of categorical features or lags of time series variables.

In this spirit, we require that a candidate feature be a Python[2] function that

- accepts a single input parameter, `dataset`, and
- returns a single column of numerical values that contains as many rows as there are entity instances, that is ordered in the same way as the entity table, and that can be represented as a tabular data structure such as a `DataFrame`.

In order for the feature values to be reproducible, we also require that features not use variables, functions, or modules that are defined outside of the function scope, nor external resources located on the file system or elsewhere. This ensures

[2]We could easily extend this framework to allow the use of other languages commonly used in data science, such as R, Julia, or Scala.

Phase	Method	Functionality
Launch (Coordinator)	`prepare_dataset`	Prepare the dataset and load it into the FeatureHub interface.
	`preextract_features`	Extract simple features.
	`setup`	Launch multiple machines on Amazon with JupyterHub installed and the dataset set up.
Create (Data Scientists)	`get_dataset`	Load the dataset into the workspace.
	`evaluate`	Validate and evaluate a candidate feature locally, on training data.
	`discover_features`	Discover and filter features in the database that have been added by the user and other workers.
	`submit`	Validate and evaluate* a candidate feature remotely, on test data, and, if successful, submit it to the feature database.
Combine (Coordinator)	`extract_features`	Execute the feature extraction scripts submitted by the data scientists.
	`learn_model`	Learn a machine learning model using AutoML.

* Though both of these last two methods "evaluate" the performance of the candidate feature, they are named from the perspective of the data scientist's workflow.

TABLE I
Set of methods for data scientists to interact with FeatureHub.

that the source code defining the feature is sufficient to reproduce feature values by itself, such that it can be re-executed on unseen test data. We verify that a feature meets these requirements during the feature submission process. A trivial feature that fits this scaffold is shown in Figure 2.

Evaluate and submit: After the user has written a candidate feature, they can evaluate its validity and performance locally on training data using the `evaluate` command. This procedure executes the candidate feature on the training dataset to extract feature values, and builds a feature matrix by concatenating these values with any pre-extracted features provided by the problem coordinator. A reference machine learning model is then fit on this feature matrix, and metrics of interest are computed via cross-validation and returned to the user. This procedure serves two purposes. First, it confirms to the user that the feature has no syntax errors and minimally satisfies the scaffolding. (As we will see, this by itself is not sufficient to ensure that the feature values are reproducible.) Second, it allows them to see how their feature has contributed to building a machine learning model. If the resulting metrics are not suitable, the data scientist can continue to develop and ideate.

Once the user has confirmed the validity of their feature and is satisfied by its performance, they submit it to the feature evaluation server using the `submit` command. In this step, they are also asked to provide a *description* of the feature in natural language. This description serves several valuable purposes. It allows the feature to be easily labeled and categorized for in-notebook or forum viewing. It also facilitates a redundant layer of validation, allowing other data scientists or problem coordinators to verify that the code as written matches the idea that the data scientist had in mind. It may even be used to provide a measure of interpretability of the final model, because descriptions of the features included in the final model can be included in output shown to domain experts or business analysts. At the server, the same steps are repeated, with slight exceptions. For example, the machine learning model is fit using the entire training dataset, and metrics are computed on the test set. The fact that the feature is extracted in a separate environment with an isolated namespace and a different filesystem ensures that the resulting feature values can be reproduced for future use.

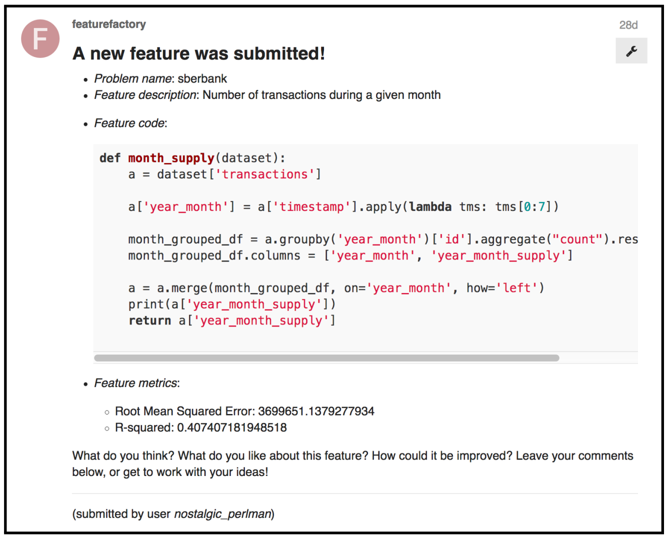
If the feature is confirmed to be valid, the feature is then both registered to the feature database and posted to a forum. The URL of the forum post is returned to the user, so that they can begin, view, or participate in a discussion around their feature.

Facilitating collaboration: Although the data scientist crowd workers are already collaborating implicitly, in the sense that their code contributions are combined, FeatureHub also aims to make this collaboration more explicit. We facilitate this through several approaches.

First, we provide an in-notebook API method, `discover_features`, that allows users to query the feature database for features that have been submitted by others, optionally filtering on search terms. This, for example, allows a user who is considering developing a feature for a particular attribute to see all features that mention this attribute. If there are close matches, the user could avoid repeating the work, and develop another feature instead. The user could also use the existing feature as a jumping-off point for a related or new idea.

Second, we tightly integrate a Discourse-based forum. Discourse[3] is an open-source discussion platform that is often used as a forum for discussing software projects. Users are provided with a forum account by the coordinator. They can then post to several categories, including *help*, where they can ask and answer questions about technical usage of the platform, and *features*, where they can see the formatted features, automatically posted, that have been successfully submitted to the feature database. This provides an opportunity

[3]https://www.discourse.org


featurefactory 28d

A new feature was submitted!

- *Problem name*: sberbank
- *Feature description*: Number of transactions during a given month
- *Feature code*:

```
def month_supply(dataset):
    a = dataset['transactions']

    a['year_month'] = a['timestamp'].apply(lambda tms: tms[0:7])

    month_grouped_df = a.groupby('year_month')['id'].aggregate("count").res
    month_grouped_df.columns = ['year_month', 'year_month_supply']

    a = a.merge(month_grouped_df, on='year_month', how='left')
    print(a['year_month_supply'])
    return a['year_month_supply']
```

- *Feature metrics*:
  - Root Mean Squared Error: 3699651.1379277934
  - R-squared: 0.407407181948518

What do you think? What do you like about this feature? How could it be improved? Leave your comments below, or get to work with your ideas!

(submitted by user *nostalgic_perlman*)

Fig. 3. An forum post for the *sberbank* prediction problem showing an actual feature submitted by a crowd worker. The feature description, source code, and computed metrics are shown, along with prompts for further discussion.

for users to discuss details of the feature engineering process, post ideas, get help from other users about how to set up a feature correctly, and use existing features as a jumping-off point for a related or new idea. An example forum post is shown in Figure 3.

While the specifics of this forum integration are important for our current instantiation of FeatureHub, the overarching idea is to define the unit of discussion at the *feature* level, facilitating pointed feedback, exploration, and ideation — one feature at a time. This contrasts with the software engineering approach, in which feedback is often presented at the *commit* or *pull request* level.

C. Combine

During or after the feature engineering process, the coordinator can use FeatureHub to build a single machine learning model, using the feature values from every feature submitted thus far. To do so, the coordinator uses the following methods:

- `extract_features`: Feature source code is queried from the database and compiled into functions, which are executed on the training and test datasets to extract corresponding feature matrices.
- *learn_model*: A sophisticated automated machine learning framework, `auto-sklearn` [5], is used to build a single predictive model. The coordinator can modify *AutoML* hyperparameters, but the goal is to build the model with little intervention. Alternately, the coordinator can override this behavior and build models directly for finer control.

These two tasks can be executed by the coordinator multiple times, at each point assessing the combined work of the data scientists to date.

IV. Platform architecture

The platform is architected around several priorities. First, it must support concurrent usage by dozens of data scientists, each of whom needs an isolated environment suitable for data science, along with a copy of the training data. Next, it must integrate heterogeneous source code contributions by different workers into a single machine learning model. Hand-in-hand with these is the requirement that the platform must be able to safely execute and validate untrusted source code without leaking information about the test sample. Finally, it must enable detailed logging of users' interactions with the platform, to be used in further analysis of data scientists' workflows.

A schematic of the FeatureHub platform is shown in Figure 4. We build on top of JupyterHub, a server that manages authentication and provisioning of Jupyter Notebook container environments. Each individual user environment is preloaded with the most common data science packages, as well as FeatureHub-specific abstractions for data acquisition and feature evaluation and submission, as discussed in Section III-B.

Careful consideration is given to ensuring that all code contributions are thoroughly validated, and that extracted feature values can be fully reproduced. To achieve this, a user first evaluates their feature on training data in the local environment. This ensures that the feature code has no syntax errors and minimally satisfies the scaffolding. However, it is not sufficient to ensure that the feature values are reproducible.

When the user attempts to submit a feature, FeatureHub both extracts the feature source code and serializes the Python function. Then, the function is deserialized by a remote evaluation service, which attempts to extract the feature values again, this time using the unseen test dataset as input. This reveals reproducibility errors such as the use of variables or libraries defined at the global scope, or auxiliary files or modules stored on the local file system. This service also has flexibile post-evaluation hooks, which we use to integrate with a separate Discourse forum, but can also enable other custom behaviors.

Logging the users' interactions with FeatureHub allows us to more carefully analyze the efficacy of the platform and user performance. Indeed, this is a key advantage of developing and deploying our own system. We log user session events and detailed information each time the user attempts to evaluate or submit a feature in the database backend.

Finally, we design the platform with an aim towards easy deployment, so that it can more easily be used in classroom or intra-organizational environments.

V. Experiments

We conducted a user test to validate the FeatureHub concept and to compare the performance of a crowdsourced feature engineering model to that of independent data scientists who work through the entire data science pipeline. We also assessed to what extent the collaborative functionality built into the platform affects collaboration and performance.

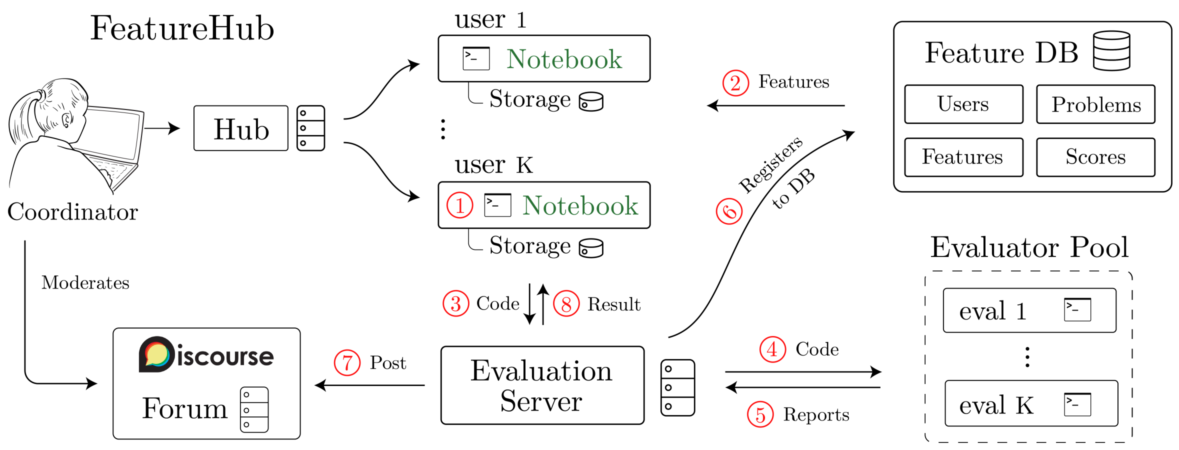


Fig. 4. Overview of FeatureHub platform architecture, comprising the FeatureHub computing platform and the Discourse-based discussion platform. A user writes a feature and evaluates it locally on training data. The user then submits their code to the Feature Evaluation server. The code is validated to satisfy some constraints and to ensure that it can be integrated without bugs into the predictive model. The corresponding feature is extracted and an automated machine learning module selects and trains a predictive model using the candidate features and other previously-registered features. The results are registered to the database, posted to the forum, and returned to the user.

A. *Freelance data scientists*

We recruited data scientists on Upwork[4], a popular freelancing platform. We advertised a position in feature engineering, filtering users that had at least basic experience in feature engineering using Python with *pandas*, a common module for manipulating tabular data. Unlike other crowdsourcing tasks, in which relatively unskilled workers execute "microtasks," the quality of feature engineering depends heavily on the freelancer's level of expertise. To allow workers with different experience levels to attempt the task, we hired each freelancer at their proposed rate in response to our job posting, up to a $45 per hour limit. Figure 5 shows the distribution of hourly rates for 41 hired crowd workers. This exposes FeatureHub to an extreme test in enabling collaboration; as opposed to a small, in-house, data science team or a group of academic collaborators, our experimental participants live in many countries around the world, work in different time zones, and possess greatly varying communication ability, skill levels, and experience.

Users were provided with a tutorial on platform usage and other documentation. They were then tasked with writing features that would be helpful in predicting values of the target variable.

B. *Prediction problems*

We presented data scientists with two prediction problems. In the first problem, data scientists were given data about users of the home rental site Airbnb and their activity on the site [4]. Data scientists were then tasked with predicting, for a given user, the country in which they would book their first rental. In the second problem, workers were given data provided by Sberbank, a Russian bank, on apartment sale transactions and economic conditions in Russia [6]. Data scientists were then tasked with predicting, for a given transaction, the apartment's final selling price. Details of these prediction problems are shown in Table II.

[4]https://www.upwork.com

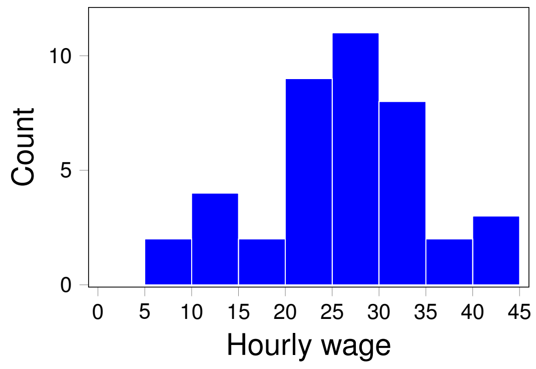


Fig. 5. Hourly wages of data scientist workers from Upwork.

Problem Name	Type	Tables	Examples	
			Train	Test
airbnb	Classification	4	213451	62096
sberbank	Regression	2	30472	7662

TABLE II
Prediction problems used in FeatureHub user testing.

C. *Experiment groups*

In order to assess the effects of different types of collaborative functionality on feature performance, we split workers into three groups. The first group was provided access to the FeatureHub platform, but was not able to use the in-notebook feature discovery method nor the forum for viewing and discussing submitted features. Thus, this group consisted of isolated data scientists who were implicitly collaborating in the sense that their features were combined into a single model, but otherwise had no coordination. A second group was provided access to and documentation for the `discover_features` client method as well as the feature forum. A third group was provided access to the same functionality as the second group, and was also provided monetary incentives to avail themselves of this functionality.

D. *Performance evaluation*

Although we hired workers at an hourly rate, we also wanted to create mechanisms that would motivate both high-quality features and collaborative behavior. We sought to align incentives that would motivate an entire group towards a particular goal: achieving the best-performing model using the features written by all workers in that group. To that extent, we offered participants bonus payments based on their performance in two broad areas.

- *Collaborative behavior*: Contributing to forum discussions, building off code written by other participants, writing well-documented and reusable code, and writing clear, accurate, informative, and concise natural language descriptions of features.
- *Evaluation of features*: Feature ranking of final model and incremental models, and qualitative evaluation of complexity and usefulness of features.

Bonus payments were offered in terms of percentage points over the worker's base hourly rate, allowing us to ignore differences in base salary. We then created a bonus pool consisting of 25 percentage points per user. Thus, a user that did no work might receive no bonus, an average user might receive a bonus of 25% of their base salary, and an exceptional worker could receive a bonus of as much as $25n$ % of their base salary, where n is the number of workers in their group. We then clearly defined, to each worker, what objective we were working toward, how we were measuring their performance with respect to this objective, and what weight we gave to different performance categories and subcategories. The weights shown to each group can be seen in Table III.

Group	1,2	3
Collaboration	28	50
. . . Feature descriptions	28	20
. . . Forum interactions	0	30
Evaluation of features	72	50
. . . Quantitative	57	40
. . . Qualitative	15	10

TABLE III
Bonus payment weights (in %) for control and experimental groups. For groups 1 and 2, the "Collaboration" category was listed as "Documentation" to avoid bias, and the "Forum interactions" subcategory was not shown.

VI. Results and discussion

Collectively, the data scientist crowd workers spent 171 hours on the platform. Of the 41 workers who logged into the platform, 32 successfully submitted at least one feature, comprising 150 of the hours worked. In total, we collected 1952 features. We also administered a detailed post-experiment survey to participants, asking about prior feature engineering tasks and their experience with the FeatureHub platform.

We limited data scientists to 5 hours on the platform, and those that submitted at least one feature used, on average, slightly less than that time. Before a data scientist could begin to ideate and write features, they needed to invest some time in learning the basics of the platform and understanding the specific prediction problem. Though 21% of users reported beginning to work on a specific prediction problem within 20 minutes of logging in, another 39% worked through tutorials and documentation for 40 minutes or more, restricting the time they spent on feature engineering directly. This constraint meant that a worker was allotted at most 2.5 hours to read the problem description, familiarize themselves with the data, and write, test, and submit features. In a real-world setting, data scientists may spend days or weeks becoming familiar with the data they are modeling. Even so, we find that useful features can be created within minutes or hours.

A. *Integrated model performance*

As described in Section III-C, during or after the feature engineering process, the project coordinator can combine source code contributions into a single predictive model. This should be accomplished with minimal intervention or manual modeling on the part of the coordinator, a key consideration reflected in the design. FeatureHub provides abstractions that allow the coordinator to automatically execute functions that extract feature values and use these as inputs to a machine learning model. Modeling can then be done either via an integrated wrapper around an automated machine learning library, or via manual model training, selection, and tuning.

Upon the conclusion of the experiment, we use this functionality to extract final feature matrices and model the feature matrix using our integrated *auto-sklearn* wrapper. Using these completely automatically generated models, we make predictions for the unseen test sets on Kaggle. The results of our models, as well as those of other competitors, are shown in Figures 6 and 7.

Overall, our models achieve performance that is competitive with the best models submitted by expert data scientists working for weeks or months at a time. The scores we achieve, though not close to winning such competitions, place our automatically generated models within several hundredths of a point of the best scores: 0.03 and 0.05 for *airbnb* and *sberbank*, respectively. In both cases, the scores achieved by our models put them at an inflection point, in which an important amount of predictive capability has been unlocked, but the last few hundredths or thousandths or a point of performance on the target metric have not been met. To be sure, to a business, the value proposition of this difference, though small in magnitude, can be significant. Regardless, this exercise demonstrates that the combination of crowd workers writing creative features and automated modeling can produce useful, "good-enough" results. If so desired, the coordinator can devote more resources to increasing the length of the CREATE phase, the automated machine learning modeling, or

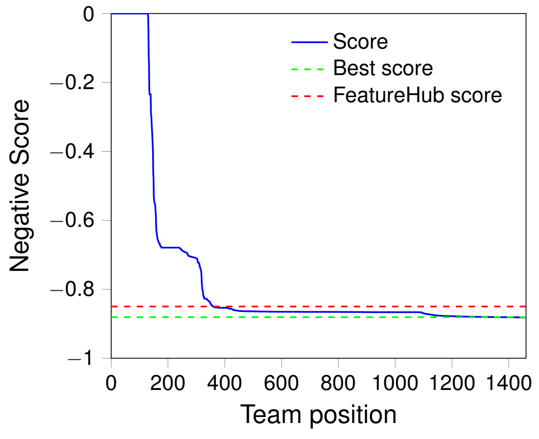


Fig. 6. Performance of the FeatureHub integrated model as compared to independent data scientists on *airbnb*, measured as normalized discounted cumulative gain at $k = 5$. The negative of the scoring metric is reported on the y-axis to facilitate a "smaller-is-better" comparison. The automatically generated FeatureHub model ranked 1089 out of 1461 valid submissions, but was within 0.03 points of the best submission.

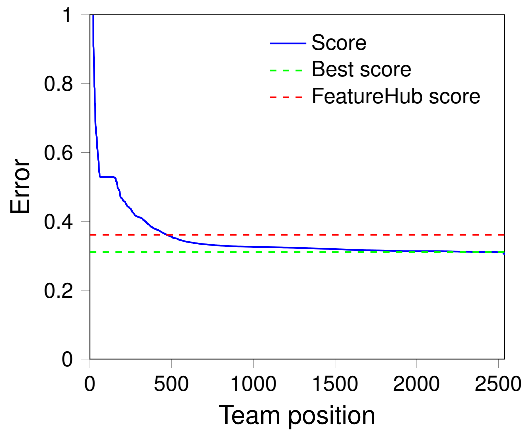


Fig. 7. Performance of FeatureHub integrated model as compared to independent data scientists on *sberbank*[6], measured as root mean squared logarithmic error. The automatically generated FeatureHub model ranked 2067 out of 2536 valid submissions, but was within 0.05 points of the best submission.

direct modeling, up to the point where they are satisfied with the model's performance.

B. *Time to solution*

If the comparison is between data scientists working for weeks or months and collaborators working for 2.5 hours or less per person, it may not be surprising that the former group outperforms the latter. However, for many organizations or employers of data science solutions, an operative concern is the end-to-end turnaround time for producing a solution. Under the FeatureHub approach, data scientists can be requisitioned immediately and work in parallel, meaning that the parameters controlling time to model solution are the amount of time for each data scientist to work on feature engineering and the amount of time to run the automated modeling. In our experiment, we allow participants to work for only 2.5 hours on each feature engineering task and run *auto-sklearn* for only six hours, for a potential turnaround time of less than one day.

On the other hand, in the independent or competitive approach, each data scientist has little incentive to begin work immediately, and rather considers the total amount of time they anticipate working on the problem and the final submission deadline, which may be two to three months ahead. In Figure 8, we show the time from first submission to first recording a submission that beats the score of the FeatureHub automated model output. Of the participants who succeed in surpassing FeatureHub at all, 29% take two days or longer to produce these submissions. This, too, is an extremely conservative analysis, in that many competitors work actively before recording a first submission. Finally, once a winning solution has been identified at the end of the entire competition period, the problem sponsor must wait for the winner's code and documentation to be submitted, up to two weeks later, and integrate the black box model into their own systems [4].

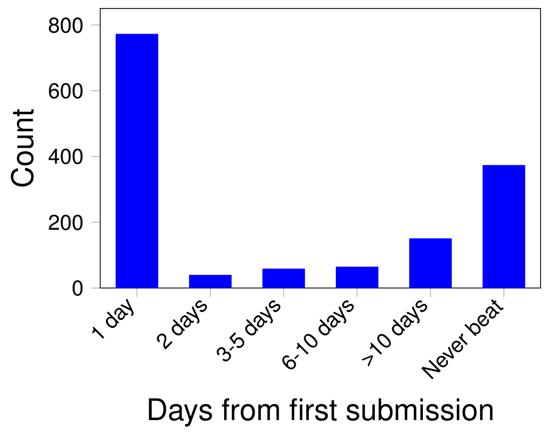


Fig. 8. The amount of days, by Kaggle competitor, from his/her first submission to the problem to their first submission that achieved a better score than FeatureHub on *airbnb*. This omits the effect of participants' preprocessing, feature engineering, and modeling work before initially submitting a solution. Considering that participants could begin the competition at any point over 2.5 month span, many top solutions were submitted weeks or months after the competition began.

C. *Diverse features*

Given that crowd workers have a variety of backgrounds and skill levels, as well as diverse intuition, it is unsurprising that we observed significant variation in the number and quality of features submitted. We visualize the collected features in Figures 9 and 10. Data scientist workers were able to draft features that covered a wide subset of the feature space. By comparison, we also show features automatically generated by the *deep feature synthesis* algorithm [7]. In a two-dimensional projection, features generated by FeatureHub data scientists take up more of the feature space than is filled by automatically generated features.

One challenge was that some participants thought they could maximize the reward for their features by registering very

[6]*Sberbank* competition leaderboard accessed on 2017-06-04; the competition end date is 2017-06-29.

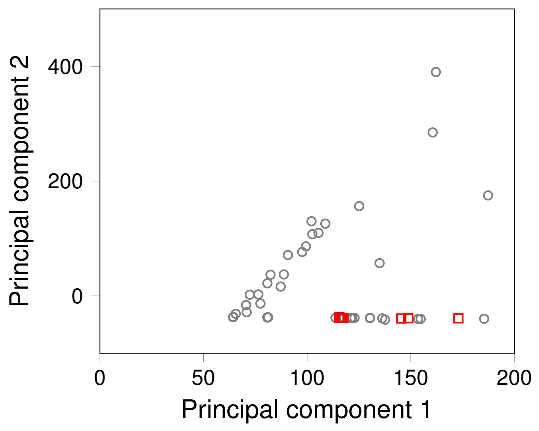


Fig. 9. Features for *airbnb* projected on first two PCA components by FeatureHub (grey circles) and Deep Feature Synthesis (red squares).

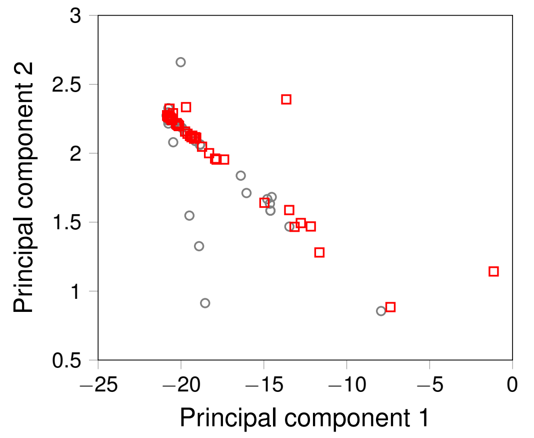


Fig. 10. Features for *sberbank* projected on first two PCA components by FeatureHub (grey circles) and Deep Feature Synthesis (red squares).

similar features in a loop. Though individual features scored low, based on criteria in Table III, we observed a single data scientist who still submitted an overwhelming $1,444$ features in our experiment. Automated modeling strategies that perform feature selection or use tree-based estimators are able to discard less predictive features. Nevertheless, such behavior should be restricted as best as possible.

D. *Facilitating collaboration*

Facilitating collaboration among data scientists is a challenging prospect. Data scientists, who may not have much software engineering experience, often do not take advantage of existing approaches such as version control systems.

FeatureHub facilitates implicit collaboration among data scientists through the integration of submissions from all workers. In this conception, the model of the group outperforms the model of any individual. We probe this suggestion by building models for each user separately based on the set of features that they submitted, and comparing this to a single model built on the entire group's feature matrix. We find that in the case of the *sberbank* prediction problem, the group's model improved upon the best user's model by 0.06 points. To be sure, it is not surprising that adding more features to a model can improve its performance. However, this still shows that the amount of work that can be produced by individuals working in isolation is limited, in practice, compared to the scalability of the crowd.

As discussed in Sections V-C and V-D, one approach to facilitating more explicit collaboration is through the implementation of native feature discovery methods, as well as through the integration of a discussion forum. Due to the substantial variation in individual worker quality and productivity, we were not able to robustly disentangle the effects of these mechanisms on feature performance. However, qualitative observations can be made. We found that including functionality for explicit collaboration reduced the mean time from first logging in to the platform to successfully submitting a first feature by about 45 minutes, as reported by participants. One factor contributing to this effect was that data scientists with access to the `discover_features` command used this functionality approximately 5 times each, on average, allowing them to see and build off existing successful submissions. Furthermore, this subset of data scientists reported that they found the integrated discussion most helpful for learning how to successfully submit features, through imitation or by asking others for help, and for avoiding duplicating work already submitted by others.

VII. Related Work

FeatureHub relates to work in several areas, including data science competitions, crowdsourced data analysis, and automated machine learning.

Programming and data science competitions: There are several existing platforms that present data science or machine learning problems as *competitions*. The most prominent of these, Kaggle, takes data science problems submitted by companies or organizations and presents them to the public as contests, with the winners receiving prizes provided by the problem sponsor. Competing data scientists can view problem statements and training datasets, and can develop and train models on their own machines or in cloud environments. They can also discuss the competition, comment on each others' code, and even modify existing notebooks. The idea of gamifying machine learning competitions has led to increased interest in data science, and the Kaggle platform has lowered the barrier to entry for new practitioners.

However, there are several aspects of Kaggle's approach that FeatureHub critiques and aims to improve upon. First, Kaggle's format may lead users to primarily tackle the machine learning algorithms themselves, instead of the features that go into them. Rather than focusing their collective brainpower on the difficult "bottleneck" step of feature engineering, users may engineer only a few features before attempting to tinker with different learning algorithms and their parameters, a potentially less efficient road to progress. In FeatureHub, we constrain users to work only on feature engineering while purposely abstracting away the other elements of the data science pipeline. Second, the pool of organizations that are willing to sponsor challenges is relatively small, because such sponsorship requires releasing proprietary data to the public. In FeatureHub, organizations with sensitive data can release a very small subset — just enough to allow data scientists to understand the schema and write feature definitions — yet still extract feature values in the backend on the entire, unseen test set. Third, because datasets are released in varying formats, users spend a significant amount of initial time wrangling and preprocessing the data, and these same tasks are redundantly performed by thousands of competitors over the course of the exercise.

In another vein, researchers have focused on factors that drive innovation in programming contests [8]. These factors include a competitive structure highlighted by leaderboards, real-time feedback, discussion, and open-source code. We seek to take advantage of these patterns, and more, in FeatureHub.

Crowdsourced data labeling: More and more researchers have used the crowd to label datasets, often of images, at scale, using crowdsourcing platforms like Amazon Mechanical Turk [9]–[11]. These platforms are best suited for relatively unskilled workers performing microtasks, such as labeling a single image. Crowd workers can also be leveraged for more complex tasks via hybrid systems that use machine learning models to prepare or combine data for human workers. For example, [12] use crowd workers to continuously evaluate machine-generated classifications in a large-scale multiclass classification setting. In another example, [13] use crowd workers to count the number of objects in an image by first segmenting the image into smaller frames using machine vision algorithms.

Crowdsourced feature engineering: Crowd workers have been involved in feature engineering in the past, but to a limited extent. Most systems have used crowd workers either to assess the value of a feature on an example-by-example basis (which can be thought of as data labeling, as above) or to compose features using natural language or other interface elements. For example, [14] incorporate crowd workers in the data labeling and feature engineering steps when constructing hybrid human-machine classifiers. In their system, users label data and give their opinions on different feature definitions via a structured voting system. Feature values must be manually extracted from these feature definitions; thus, scalability and automation remain issues.

Automated machine learning: Recent advances in hyperparameter optimization and open-source software packages have led to increased use of, and attention paid to, automated machine learning (*AutoML*) methods. [5] use Bayesian optimization techniques to automatically choose approaches to feature preprocessing and machine learning algorithms, and to select hyperparameters for these approaches. They improve on prior work [15] that formalizes *AutoML* problems as *combined algorithm selection and hyperparameter optimization*. In a similar approach, [16] use genetic programming to optimize machine learning pipelines. All of these state-of-the-art algorithms enable automated machine learning models to be deployed with relatively little difficulty, performing well in various machine learning tasks. However, one downside is that these systems expect input to be structured as a feature matrix. In many practical situations, input to the data science pipeline begins in a highly relational form, or in unstructured formats like logs, text, or images. Thus, manual feature engineering is still required to convert unstructured data into a feature matrix.

Automated feature engineering: In [7], the authors develop an algorithm for automatically extracting features from relational datasets. This algorithm, called *Deep Feature Synthesis*, enumerates a subset of the feature space through understanding of the dataset schema and the primary key-foreign key relationships. In the databases community, a body of work has been developed for *propositionalizing* relational databases into "attribute-value" form for data-mining applications [17], [18]. For other unstructured data, such as text and images, techniques like `tf-idf` vectors, Latent Dirichlet Allocation [19], convolutional neural networks, and autoencoders can extract feature representations.

VIII. CONCLUSION

We have presented a novel workflow for leveraging the teams of data scientists in feature engineering efforts, and instantiated this approach in FeatureHub, a platform for collaborative data science. This workflow addresses several shortcomings of existing efforts to crowdsource data science or machine learning. Whereas in traditional models, data scientists redundantly perform dataset cleaning and preprocessing as well as model selection and tuning, with FeatureHub, we focus workers' creative effort on feature engineering itself. We propose a set of abstractions and code scaffolding for feature implementations, allowing us to address another shortcoming of existing approaches: by processing feature definitions in real-time, as they are written and submitted by users, we can incrementally and automatically build models, allowing a problem coordinator great flexibility over the metrics they use to govern a task's stopping point. Moreover, by scaling the crowd up and down, the wall clock time to task completion can be managed directly, rather than being tied to the stated end date of a traditional data science competition.

We validated our platform in an experiment combining features submitted by freelance data scientists of different backgrounds working around the globe. Using automatically generated models, we achieved performance within 0.03 to 0.05 of the best models on Kaggle. Though our models are not close to winning such competitions, we achieve reasonable performance with limited resources and human oversight. By implementing, deploying, and testing FeatureHub, we have taken steps towards collaborative data science and feature engineering and opened a variety of opportunities for further exploration.

A. Future Work

Several avenues for future work present themselves. The platform could be designed to provide more accurate real-time feedback upon feature submission. That is, the models built to provide initial evaluation of candidate features do not take into account the set of features that have already been submitted to that point, nor the features that are likely to be submitted in the future. Efficient algorithms and metrics could be developed that would estimate a single feature's contribution to the final machine learning model in the absence of future features. This would also allow a problem coordinator to include real-time payments based on feature evaluation metrics in their incentive model.

FeatureHub, though a compelling demonstration of the power of the workflow we propose, could be made more robust and scalable, for use within organizations or classrooms. In the latter case, the platform could be used to teach feature engineering and tabular data manipulation without frustrating introductory students with problems of data acquisition and model development. A variety of additional functionality could also be built into the platform. For example, feature scaffolding could be modified to expose feature hyperparameters, such as threshold levels, thus allowing the system to include these in end-to-end hyperparameter optimization.

Finally, the power of the crowd could be harnessed for other aspects of data science that are currently handled, either implicitly or explicitly, by the problem coordinator. For example, additional sets of crowd workers could assume separate roles, as in [20]. *Validation* — ensuring that feature definitions do not have semantic bugs, and that feature descriptions match the code that is executed — and *curation* — selecting the most promising set of features for further refinement or specifc inclusion in a model, with a focus on interpretability or parsimony — would fit nicely within the FeatureHub workflow.

Acknowledgment

We thank the authors of [7] for kindly providing us with features produced by the *deep feature synthesis* algorithm. This research is partially funded by the National Science Foundation award number ACI-1443068.

References

[1] C. Taylor, K. Veeramachaneni, and U. O'Reilly, "Likely to stop? predicting stopout in massive open online courses," *CoRR*, vol. abs/1408.3382, 2014.

[2] P. Domingos, "A few useful things to know about machine learning," *Communications of the ACM*, vol. 55, no. 10, pp. 78–87, 2012.

[3] "Crowdsourcing feature discovery," http://radar.oreilly.com/2014/03/crowdsourcing-feature-discovery.html, accessed: 2015-01-30.

[4] "Kaggle airbnb new user bookings," https://www.kaggle.com/c/airbnb-recruiting-new-user-bookings, accessed: 2017-06-04.

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

[6] "Kaggle sberbank russian housing market," https://www.kaggle.com/c/sberbank-russian-housing-market, accessed: 2017-06-04.

[7] J. M. Kanter and K. Veeramachaneni, "Deep feature synthesis: Towards automating data science endeavors," in *Data Science and Advanced Analytics (DSAA), 2015. 36678 2015. IEEE International Conference on*. IEEE, 2015, pp. 1–10.

[8] N. Gulley, "Patterns of innovation: a web-based matlab programming contest," in *CHI'01 extended abstracts on Human factors in computing systems*. ACM, 2001, pp. 337–338.

[9] "Amazon mechanical turk," https://www.mturk.com/.

[10] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[11] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, "Microsoft coco: Common objects in context," in *European conference on computer vision*. Springer, 2014, pp. 740–755.

[12] C. Sun, N. Rampalli, F. Yang, and A. Doan, "Chimera: Large-scale classification using machine learning, rules, and crowdsourcing," *Proceedings of the VLDB Endowment*, vol. 7, no. 13, pp. 1529–1540, 2014.

[13] A. D. Sarma, A. Jain, A. Nandi, A. Parameswaran, and J. Widom, "Surpassing humans and computers with jellybean: Crowd-vision-hybrid counting algorithms," in *Third AAAI Conference on Human Computation and Crowdsourcing*, 2015.

[14] J. Cheng and M. S. Bernstein, "Flock: Hybrid crowd-machine learning classifiers," in *Proceedings of the 18th ACM Conference on Computer Supported Cooperative Work & Social Computing*. ACM, 2015, pp. 600–611.

[15] C. Thornton, F. Hutter, H. H. Hoos, and K. Leyton-Brown, "Auto-WEKA: Combined selection and hyperparameter optimization of classification algorithms," in *Proc. of KDD-2013*, 2013, pp. 847–855.

[16] R. S. Olson, N. Bartley, R. J. Urbanowicz, and J. H. Moore, "Evaluation of a tree-based pipeline optimization tool for automating data science," in *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, ser. GECCO '16, 2016, pp. 485–492.

[17] A. J. Knobbe, M. de Haas, and A. Siebes, *Propositionalisation and Aggregates*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 277–288.

[18] S. Kramer, N. Lavrač, and P. Flach, *Propositionalization Approaches to Relational Data Mining*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2001, pp. 262–291.

[19] D. M. Blei, A. Y. Ng, and M. I. Jordan, "Latent dirichlet allocation," *Journal of machine Learning research*, vol. 3, no. Jan, pp. 993–1022, 2003.

[20] M. S. Bernstein, G. Little, R. C. Miller, B. Hartmann, M. S. Ackerman, D. R. Karger, D. Crowell, and K. Panovich, "Soylent: A word processor with a crowd inside," in *Proceedings of the 23Nd Annual ACM Symposium on User Interface Software and Technology*, ser. UIST '10, 2010, pp. 313–322.