Feature Factory: A Collaborative, Crowd-Sourced Machine Learning System

by

Alex Christopher Wang

Submitted to the Department of Electrical Engineering and Computer Science
in partial fulfillment of the requirements for the degree of

Masters of Science in Computer Science and Engineering

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

Author .....
Department of Electrical Engineering and Computer Science
May 22, 2015

Certified by .....
Kalyan Veeramachaneni
Research Scientist
Thesis Supervisor

Accepted by .....
Albert Meyer
Chairman, Masters of Engineering Thesis Committee

Feature Factory: A Collaborative, Crowd-Sourced Machine Learning System

by

Alex Christopher Wang

Submitted to the Department of Electrical Engineering and Computer Science
on May 22, 2015, in partial fulfillment of the
requirements for the degree of
Masters of Science in Computer Science and Engineering

Abstract

In this thesis, I designed, implemented, and tested a machine learning learning system designed to crowd-source feature discovery called Feature Factory. Feature Factory provides a complete web-based platform for users to define, extract, and test features on any given machine learning problem. This project involved designing, implementing, and testing a proof-of-concept version of this platform. Creating the platform involved developing user-side infrastructure and system-side infrastructure.

The user-side infrastructure required careful design decisions to provide users with a clear and concise interface and workflow. The system-side infrastructure involved constructing an automated feature aggregation, extraction, and testing pipeline that can be executed with a few simple commands. Testing was performed by presenting three different machine learning problems to test users via the user-side infrastructure of Feature Factory. Users were asked to write features for the three different machine learning problems as well as comment on the usability of the system. The system-side infrastructure was utilized to analyze the effectiveness and performance of the features written by the users.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Research Scientist

Acknowledgments

I would like acknowledge Kalyan Veeramachaneni for all of his support, guidance, and help throughout the course of this thesis. I would also like to thank my parents for always loving me, supporting me, and being the reason I got to where I am today.

Contents

1 Introduction **17**

1.1 Objectives 22

2 Design of Feature Factory **27**

2.1 User-side Infrastructure 28

2.1.1 Interactive IPython notebook 28

2.1.2 Collaborative framework 30

2.1.3 Machine Learning Service 30

2.2 System-side Infrastructure 31

2.2.1 MySQL Database 31

2.2.2 Automatic Feature Extraction Infrastructure 32

2.2.3 Leaderboard 33

3 Uploading New Machine Problems to Feature Factory **35**

3.1 Data 35

3.2 Problem-Specific Code 36

3.3 Template IPython Notebook 36

3.3.1 Setup and Registration 37

3.3.2 Sample Dataset and Example Feature 37

3.3.3 Training and Testing 39

4 Feature Factory: First trials **41**

4.1 Experiment methodology 41

4.2 KDDCup 2014 42

4.2.1 Dataset 43

4.2.2 Integration with Feature Factory 48

4.2.3 Test-Train-Submission 49

4.2.4 Results 49

4.3 Route Prediction 49

4.3.1 Data Preprocessing 50

4.3.2 Problem 51

4.3.3 Dataset 51

4.3.4 Integration with Feature Factory 52

4.3.5 Results 52

4.4 IJCAI 53

4.4.1 Dataset 54

4.4.2 Integration with Feature Factory 55

4.4.3 Results 55

4.5 Lessons Learned from Trials 56

4.5.1 Feature Quality 56

4.5.2 Feedback on User Experience 57

5 Challenges Faced **59**

5.1 Example problem 59

5.2 Generating a coherent data sample 60
5.3 Debugging user submitted features 60
5.4 Noisy Estimates of Feature Accuracy 64
5.5 Compute challenges with feature extraction 64
5.6 Takeaways 68

6 Conclusion and Future Work **69**

6.1 Future Work 69

List of Figures

1-1 This is an example Kaggle competition (photo credit: Kaggle). At the top, one can see the duration of the contest. The main body of the page is taken up with a full description of the contest, its rules, and the dataset. On the bottom left, one can see the current leaders in the competition. 21

1-2 Feature Factory, at a high level, can be viewed as a pipeline with specific stages. The pipeline begins with Users writing features using Feature Factory API accessed via a web interface. In this stage, users are able to extract and test their features using a prepackaged machine learning approach on the sample dataset, but not the full dataset. After this, another module assembles all the notebooks, aggregating the features that users have written and extracting them from the full dataset. These extracted features and labels for the full dataset are then passed to a machine learning module, where a pre-selected machine learning algorithm is trained and the trained model is tested on unseen data. Finally, the resulting AUCs or accuracies are returned to the users via a leaderboard. 23

2-1 Excerpt from an IPython notebook. It can be accessed via a web interface. This particular notebook is showing the IJCAI machine learning problem which we later describe in the experiments section. The notebook can have detailed text in between two code cells. We write detailed description about each step to get user started on the machine leanring problem at hand. 29

2-2 In this figure, we visualize the feature aggregation infrastructure. The process begins by reading from the MySQL database which users have written features for a given machine learning problem and their associated IPython notebooks. The IPython notebook files are parsed, and the feature functions the users wrote within the notebooks are compiled into individual executable python files. 33

2-3 In this figure, we visualize the feature extraction infrastructure. After the feature aggregation module executes, the system is left with a series of executable python files, with each file representing a feature function waiting to be applied to the full dataset. These files are placed into a queue. Feature Factory then leverages multiple cores on an Amazon EC2 machine to pop off a job from the queue and execute it. In the case that the number of jobs overwhelms the number of cores available on the Amazon EC2 machine, Feature Factory simply spins up another EC2 machine and applies the same feature extraction process. 34

5-1 This figures shows an example of the alignment procedure in post-processing. The top left table depicts the output of a user's feature function applied to the full dataset. The bottom left table depicts the labels. Notice that the top left table does not have an entry with a unique identifier of value 3, but the bottom left table does. This implies that a feature value is missing. In this situation, te alignment procedure performs a left join on the two tables, keeping all rows from the table containing the labels, and filling in a default value of 0 as a feature value for the row with a unique identifier of 3. This ensures that the feature matrix and labels matrix align when performing training and testing. 62

List of Tables

4.1 Outcomes table within KDDCup dataset 44

4.2 Donations table within KDDCup dataset 45

4.3 Resources table within KDDCup dataset 46

4.4 Projects table within KDDCup dataset 47

4.5 Essays table within KDDCup dataset 48

4.6 Features for KDDCup and associated AUCs 50

4.7 Route prediction dataset 51

4.8 Features for route prediction and associated prediction accuracies. Note that due to instructions that were not perfectly clear, the time_traveled_start_location and total_day_driving features are incorrect, since they use data from the trip whose destination we are trying to predict. 53

4.9 Training / Test data for IJCAI repeat buyer problem 54

4.10 User profile data for IJCAI repeat buyer problem 54

4.11 User behavior log data for IJCAI repeat buyer problem 54

4.12 Features for IJCAI repeat buyer problem and associated AUCs 56

Chapter 1

Introduction

The field of machine learning has evolved significantly in the last decades. With increasing in interest in big data, researchers have made many breakthroughs in different aspects of the field.

In the academic realm, many researchers have focused on scaling existing machine learning algorithms via parallelization, approximation, and scientific computing. Specifically, researchers have realized that most machine learning algorithms are bottlenecked by a few common operations [1], and much research has been devoted to speeding these operations up through a variety of techniques.

In business environment, many companies have invested money and resources in making machine learning available to the general public through various cloud based platforms. We briefly describe, some of the most popular cloud based machine learning platforms today. [2]

- Amazon Machine Learning - generic machine learning pipeline designed for users of all skill levels.

- Algorithmia - online repository where users can share algorithms that they have written.

[1] http://www.umiacs.umd.edu/labs/cvl/pirl/vikas/publications/raykar_research_statement.pdf
[2] http://butleranalytics.com/10-cloud-machine-learning-platforms/

- Microsoft Machine Learning Studio - online cloud featuring a library of machine learning algorithms from different Microsoft businesses.

- BigML - cloud based machine learning system with a graphical user interface for ease of use.

- wise.io - provides fast implementation of the random forest algorithm designed to handle high dimensional data.

All of these new tools make machine learning algorithms more accessible to the general public, but all of them still do not focus their attention towards the biggest bottleneck in machine learning: feature discovery.

What exactly is a *feature*, and why is feature discovery hard?: [3] A feature by definition is a measurable quantity of an observable phenomenon. Specifically, a feature is a mathematical quantity extracted from a dataset by applying a specific function. For example, in spam detection, where the goal is to identify spam emails from legitimate emails, example features could be the number of misspelled words in an email or a boolean representing whether or not an email has a proper header. These features are extracted from the dataset (which in this case would be a large amount of emails) and run through a classification algorithm. So how does one discover good features? This question is the crux of the problem of feature discovery.

Features are generally discovered by data scientists who hope that their knowledge of the problem's domain gives them some insights on what features would be useful. However, recently, data scientists have admitted that feature engineering/discovery is one of the toughest aspects of machine learning [4]. Because feature engineering relies on the creativity and insights of humans, it is a problem that lends itself well to a crowd-sourced solution. If many people are able to quickly propose and effectively extract and try different features, a good set of features should be discovered in a

[3] http://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/

[4] http://radar.oreilly.com/2014/03/crowdsourcing-feature-discovery.html

short amount of time with high probability. Currently, however, no system seems to exist exclusively for this purpose.

Crowd-sourced data labeling: The idea of crowdsourcing has been applied to machine learning in the past, but in a different manner than the one we intend to create. Existing forms of crowdsourcing relegate users to performing manual tasks that computers can't do, rather than asking users to actually think about the machine learning problem at hand in order to create features. For example, many classification algorithms assume that a sample of data can be taken, labelled, and the used to train classifiers. However, what if the data can't be automatically labeled? What if the data requires a human to determine a label e.g. a picture of a particular item? If the dataset had millions, this would be an incredibly difficult task. Hence, in the past, people have turned to crowd-sourcing to create datasets to train their classifiers.[5]

Seeking human evaluations for videos and text: Recently, researchers at Stanford have progressed to create systems that not only take advantage of users to label data, but also asks users for their opinions on different features in a structured voting system [5].[6] In their paper, authors seek features for *videos* and *text* that are based on human interpretations.

For example, in the study, researchers ask test subjects to watch a video of a person making a verbal statement, and to come up with features to determine whether or not the person is lying. Some of the crowd-nominated features are whether or not the person shifts their eyes, whether or not the person uses gestures while speaking, and the brevity of the person's sentences. However, some of the features that authors get from the participants are not automatically computable. In other words, to compute such features, when attempting to deploy a machine learning system based on those features, one would need humans again or to compute such features. While, this is a step further than simply asking users to label data, it falls short in solving the actual feature engineering/discovery problem. Additionally, we argue that with the advent of deep learning based methods many automatic feature discovery methods already

[5]http://pages.cs.wisc.edu/ anhai/papers/chimera-vldb14.pdf
[6]available at: http://hci.stanford.edu/publications/2015/Flock/flock_paper.pdf

exist for *text* and *images*. On the other hand, for behavioral data no automatic feature engineering methods exist.

Kaggle: a crowd source solution for behavioral data Another approach that has been taken in terms of crowd-based machine learning is Kaggle (www.kaggle.com). Kaggle is a platform that allows companies to submit machine learning problems for the general public to work on. Kaggle takes the problems and creates competitions, with winners receiving prizes provided by the company. Each Kaggle competition has a specified duration, dataset, and a well defined problem. During a competition, users are free to download the dataset and create predictive models however they want. When users have a model they are happy with, they use the model on the test dataset that Kaggle provides and generate predictions. Users then submit these predictions to Kaggle and receive their AUC or accuracy score. Each user's score is made public through a leaderboard on the website. At any time during a Kaggle competition, users can see how they rank against others and also ask each other questions via a Kaggle-hosted forum. An example of a typical Kaggle competition is shown in Figure 1-1.

Though the idea of turning machine learning problems into actual competitions where people can win money is immensely popular, Kaggle's method of presenting machine learning challenges is rather suboptimal in multiple aspects, specified below.

Lack of focus on feature discovery: First, as mentioned before, the bottleneck in machine learning problems is in feature discovery. Rather than directing the public's focus in that direction, Kaggle's format of presenting problems directs focus to machine learning algorithms. Users often engineer a few features and attempt to tinker with different learning algorithms and their parameters in order to generate better results. This is clearly a suboptimal away of utilizing collaborative brainpower to solve machine learning problems.

Hardware limitations of the end user:Secondly, Kaggle relies on the fact that users have computers with enough memory and computing power to handle datasets. This restricts the population that is able to benefit from Kaggle's competitions. For

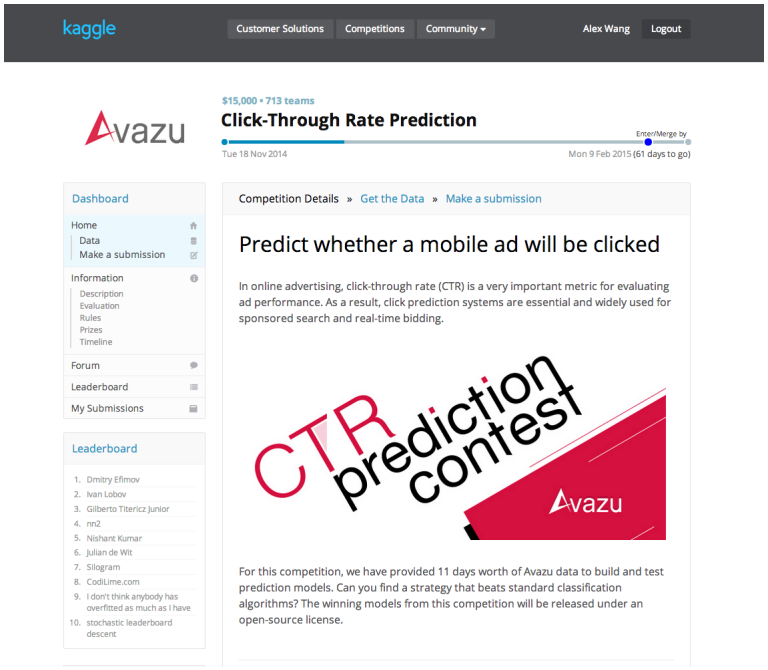

Figure 1-1: This is an example Kaggle competition (photo credit: Kaggle). At the top, one can see the duration of the contest. The main body of the page is taken up with a full description of the contest, its rules, and the dataset. On the bottom left, one can see the current leaders in the competition.

example, what if a person cannot afford anything beyond a entry-level laptop? The person is either at a disadvantage in Kaggle competitions (slower development cycle) or simply cannot compete at all.

Data sharing concerns:Thirdly, Kaggle struggles to get the attention of companies in certain fields, specifically fields where datasets are and should remain private. Since Kaggle relies on handing out entire datasets, any company with a privacy concern simply cannot use Kaggle.

Lack of Collaboration:Finally, in every Kaggle competition, users will have to spend a good portion of their time wrangling with the data. Since companies are allowed to give data in any format that they want, users will generally have a significant ramp-up period to get everything in place before even beginning to get results. To make matters worse, due to competitiveness, users may not share their data wrangling software with others, causing contestants to perform redundant work. With these problems, Kaggle has a limited effectiveness.

MIT crowdsourcing experiment: In summer of 2013, MIT researchers conducted a crowd sourcing experiment for gathering features for predicting student dropout from an online education platform. The authors sought ideas via a google form and then wrote scripts themselves [10, 11]. They report in their papers that the features that they acquired from the crowd were more complex and these features performed better than what they themselves had come up with. While they were able to gather a number of ideas they were only able to extract a few of them. This was because of the time it took to operationalize and write features for some of the scripts. Hence following this work we posit a platform that can enable users to write and submit scripts in addition to ideas.

This is where our new evolved system, Feature Factory comes in. In the following sections, we discuss a high-level overview of Feature Factory and describe its main objectives.

1.1 Objectives

Our new cloud-based feature discovery system, called Feature Factory, aims to be a framework designed to allow users to easily design, extract, and test features on any dataset. Specifically, Feature Factory wants its users to focus solely on feature discovery, delegating as much of the other work required in machine learning, such as data preprocessing, feature extraction, training, and testing, to a backend system. The ultimate goal is to provide users with the tools they need to rapidly prototype features without worrying about anything else. An overview of how this system would work is shown in Figure 1-2.

To achieve its goal, Feature Factory seeks to provide the following specific features

Interactive Interface Feature Factory aspires to provide users with an interactive interface that allows them to easily create and test features within the comfort of their web browser. This interactive interface not only allows users to rapidly

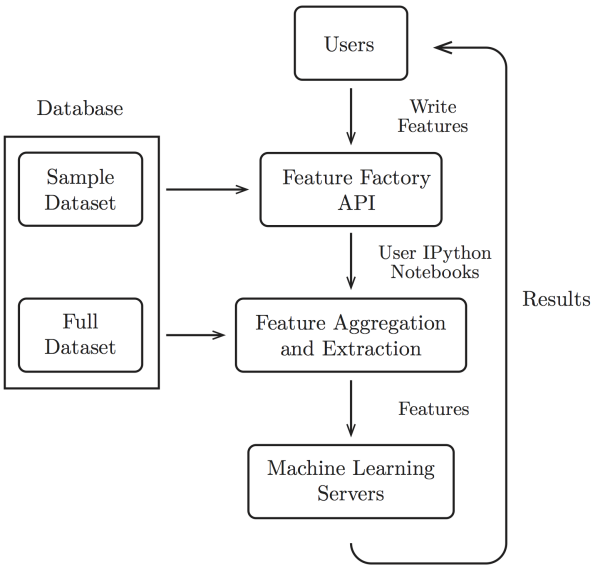

Figure 1-2: Feature Factory, at a high level, can be viewed as a pipeline with specific stages. The pipeline begins with Users writing features using Feature Factory API accessed via a web interface. In this stage, users are able to extract and test their features using a prepackaged machine learning approach on the sample dataset, but not the full dataset. After this, another module assembles all the notebooks, aggregating the features that users have written and extracting them from the full dataset. These extracted features and labels for the full dataset are then passed to a machine learning module, where a pre-selected machine learning algorithm is trained and the trained model is tested on unseen data. Finally, the resulting AUCs or accuracies are returned to the users via a leaderboard.

prototype and test features, but it also eliminates hardware limitations since users no longer need powerful machines to load and process the data.

Collaborative Interface Our system seeks to promote collaboration and interaction among its users. Not only will Feature Factory provide an open repository of features that users have written, but it will also maintain a leaderboard for the best performing features for each machine learning problem based on a variety of metrics. This leader-board provides an incentive for users to write better features, and can also be leveraged to serve as way to run competitions within Feature Factory.

In-memory Evaluation Feature Factory aims to maintain interactivity with the user. To do this, the system must ensure that the users spend as little time as possible waiting on code to execute and results to be returned. Hence, to ensure that operations waste as little time as possible accessing data, datasets are pre-loaded into memory by the Feature Factory infrastructure. This ensures that all operations are performed in-memory, thus minimizing computation time.

Private Scoring A private scoring mechanism will be provided for the features that users write. Instead of allowing users to score their features on the full dataset through an API, Feature Factory grabs the features that users write and scores them "privately" in the back-end. By doing this, Feature Factory can still provide users with relatively quick feedback on their feature performance without having to release or share a complete dataset. This provides Feature Factory the flexibility to handle data science problems that deal with sensitive information.

Feature Factory aims to popularize and crowd-source feature discovery in a revolutionary manner that does not suffer from the same problems that Kaggle currently does. Providing a seamless, easy-to-use yet powerful feature discovery platform is difficult, but Feature Factory shows promise that it can be done. In this thesis, we describe our journey in designing, creating and testing Feature Factory via chapters outlined below.

Chapter two describes the development and design of a remote machine platform, called Feature Factory. and the motivation for the design decisions made.

Chapter three describes step by step how a machine learning problem can be integrated into Feature Factory. Chapter four describes the three different machine learning problems that integrated into Feature Factory and solved using Feature Factory.

Chapter five documents the breakthroughs that Feature Factory has achieved along with future applications of Feature Factory.

Chapter six describes the conclusions we took from our journey with Feature Factory

along with ideas for future work on the platform.

Chapter 2

Design of Feature Factory

In this chapter, we describe the technical design of Feature Factory, and the motivations behind our decisions. Given that it is critical that Feature Factory be able to handle any machine learning problem, a significant amount of careful thought was placed into creating a system that is flexible and agile.

Feature factory is split into user-side infrastructure and system-side infrastructure. The components of each infrastructure are detailed below.

- User-side Infrastructure

    1. Interactive IPython Notebook

    2. Collaborative framework consisting of a shared repository of IPython

    3. Machine learning algorithms as a service

- System-side infrastructure

    1. Automatic infrastructure to extract features

    2. Leaderboard

2.1 User-side Infrastructure

2.1.1 Interactive IPython notebook

One of the biggest advantages of Feature Factory is providing users with the ability to code in the browser using IPython Notebook. IPython Notebook is a web-based framework that was designed to perform similarly to a Mathematica notebook. It allows users to write Python code in individual "cells", and execute each of these cells at the click of a button. To do this, each IPython Notebook is backed by a dedicated python kernel living on the server. The python kernel is not only responsible for executing the user's Python code, but it also provides Python object persistence. This means that if a user executes code within a cell that generates certain objects, these objects are accessible by any other cell. This is extremely useful in data science since computationally expensive operations to generate or transform datasets can be performed once and then accessed by newly written code. An example IPython notebook is shown in Figure 2-1.

Notebook creation After the user reaches the main webpage and selects a machine learning problem to work on, they are directed to an IPython Notebook. This notebook contains detailed instructions and information on how to get started with the machine learning problem at hand. Users are first asked to clone the notebook and rename it. The users then execute a Feature Factory API command to register the newly named notebook with the Feature Factory system. Once this is complete, the new notebook becomes the user's workplace for the machine learning problem.

Feature creation The user's feature is effectively a function that takes in the dataset as an object, and iterates through the rows of the dataset, giving one feature value per row as output. Feature Factory can provide the dataset as either a Numpy matrix or a Pandas data frame, catering to users of either Python library. Within the IPython Notebook environment, the user is provided a sample dataset to test his or her feature before submitting it to the Feature Factory System. Once the

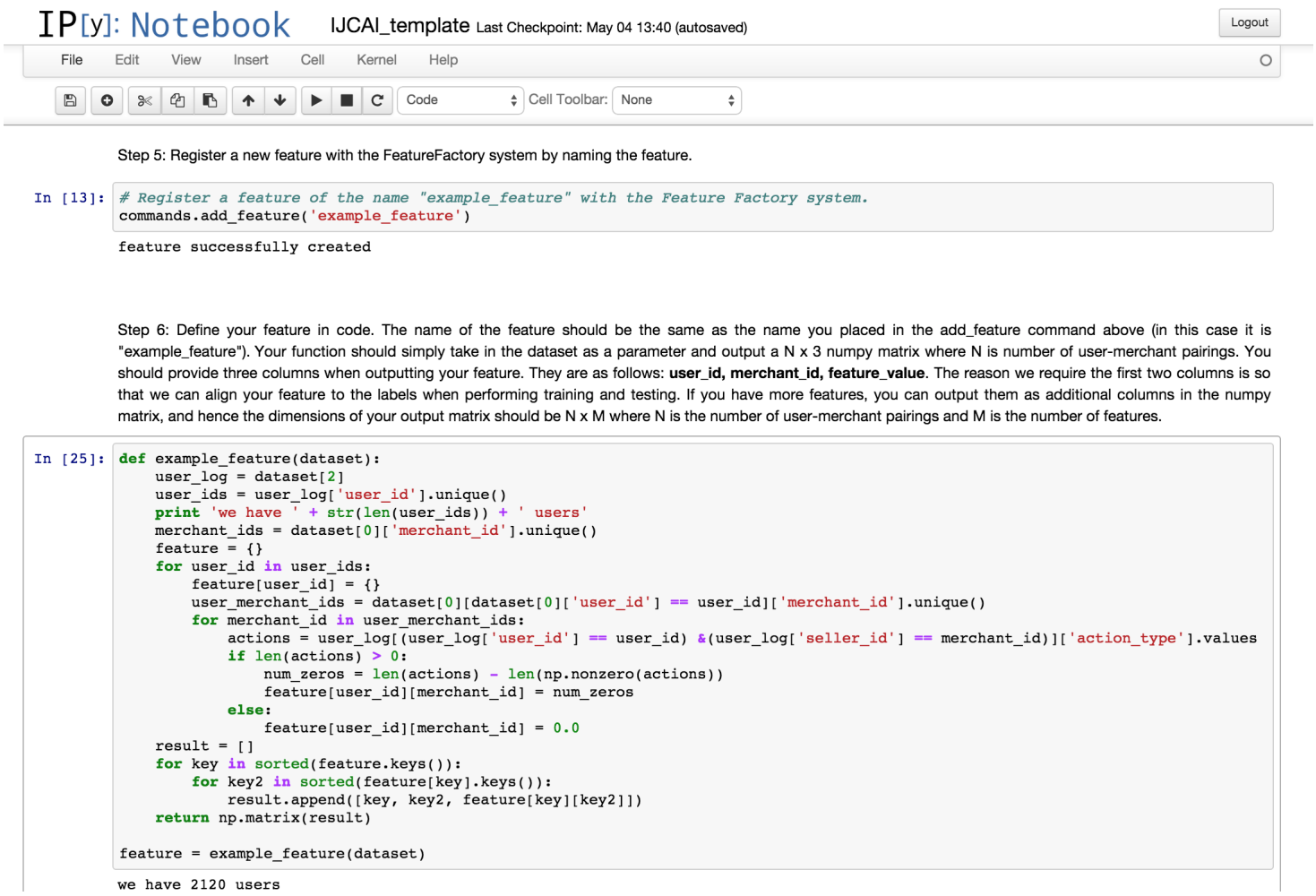

Figure 2-1: Excerpt from an IPython notebook. It can be accessed via a web interface. This particular notebook is showing the IJCAI machine learning problem which we later describe in the experiments section. The notebook can have detailed text in between two code cells. We write detailed description about each step to get user started on the machine leanring problem at hand.

user is ready, he or she can submit the feature by calling a specific Feature Factory API command. This command effectively registers the feature in Feature Factory's database, notifying Feature Factory that the user desires to test the feature on the full dataset.

2.1.2 Collaborative framework

Feature Factory not only provides users with an efficient, simple way to write features from the convenience of their web-browser via iPython Notebook, but it also promotes collaboration among its users. Although each user in Feature Factory has their own individual notebook, all users can view each other's code but not edit each other's notebooks. This allows each user to take a look at the directions in which other users are going with their features, promoting creativity among the entire user base.

2.1.3 Machine Learning Service

One of the distinguishing parts of Feature Factory is the fact that it does not allow users to perform any actual machine learning. For each machine learning problem, the machine learning algorithm and cross-validation methodology is always chosen beforehand. The concept behind providing machine learning simply as a black-box service is to force users to create better features to increase their score as opposed to trying to simply tinker with the machine learning algorithm.

Training and testing Once features have been extracted for each user, the features are fed into a variety of different machine learning algorithms, depending on the machine learning problem at hand. For general problems, Python's SciKit-Learn library is utilized for these algorithms, in addition to using k-fold cross validation to partition the dataset into training and validation sets. If a particular problem has an extremely complex or large dataset, other libraries are utilized to maintain performance and scalability. Methods to measure the accuracy and performance of features are pre-specified by the institution when submitting a machine learning problem to Feature Factory. Once a feature has been evaluated using the pre-selected classifier, the results are pushed to the database. For scalability, this procedure is performed using the same producer-consumer multi-processed infrastructure used in feature extraction.

Results The final stage of the Feature Factory workflow is returning results to the user. The training and testing module outputs a score for each feature that users submit to the system. The score can be in any form, but the most popular value returned is an AUC. The scores of all the features are saved to the database by the training and testing module, which are then read by the results module and displayed on the Feature Factory website. Users can easily login and see the scores of their features, as well as see a leaderboard of which user's features are currently doing the best for a particular machine learning problem.

2.2 System-side Infrastructure

2.2.1 MySQL Database

Feature Factory utilizes a MySQL database not for the storage of machine learning datasets, but for keeping track of users, features, machine learning problem descriptions, and their respective relationships with each other. Specifically, when a new machine learning problem is integrated into Feature Factory, its name and description is stored into a table within the database. Similarly, when users register with Feature Factory, their information (username, password, email) is stored in the MySQL database. The registration of features is also done via the database – when a user executes the Feature Factory API command to register a feature, the system saves the feature's name along with the id of the user who created it and the id of the machine learning problem it is related to into a table. All tables within the MySQL database have relationships with each other, thus making it easy to make queries for various things. For example, to figure out which machine learning problems a user, with an example id of 10, has written features for, one can simply execute the following MySQL command: `SELECT DISTINCT(problem_id) from features where user_id == 10`. This command functions properly since features, a table, stores relationships with users as well as problems, and thus these kinds of queries can be made.

2.2.2 Automatic Feature Extraction Infrastructure

Since the objective of Feature Factory is to gather, train, and test features written by different people, it is critical to automate the extraction process. Going through each feature that users have written and manually extracting them is infeasible. Hence, to do this, Feature Factory has an automatic feature extraction infrastructure. This infrastructure automatically finds all of the features that users have written for a particular machine learning and extracts them from the dataset in parallel. After the features have been extracted, they are saved in CSV files.

Feature Function Aggregation At a set time each day, the Feature Factory infrastructure runs an automated software suite to aggregate all of the python feature functions that users have written in their own IPython Notebooks. To do this, Feature Factory leverages the format of IPython Notebooks. An IPython Notebook, denoted with the extension of `.ipynb`, is effectively JSON-like files storing the contents of the notebook. Hence, Feature Factory can simply read from the MySQL database, find the users that have written functions for the given machine learning problem, retrieve the names of the feature functions for each user, and copies the feature functions from each user's IPython Notebook into their own separate python files. This complex process is expressed Figure 2-2.

Feature Extraction Now that the feature functions have been aggregated into executable python files, these functions need to be applied to the full dataset. As one can imagine, running each of these extraction jobs sequentially isn't feasible, especially with extremely large datasets. Hence, Feature Factory uses a producer-consumer model to multiprocess the feature extraction process. Feature Factory spawns anywhere between 2 to 12 processes, and each process pops a job off of the queue until the queue is empty. When a process finishes extracting a feature, the results are saved in CSV format. A graphical representation of this infrastructure is shown in Figure 2-3.

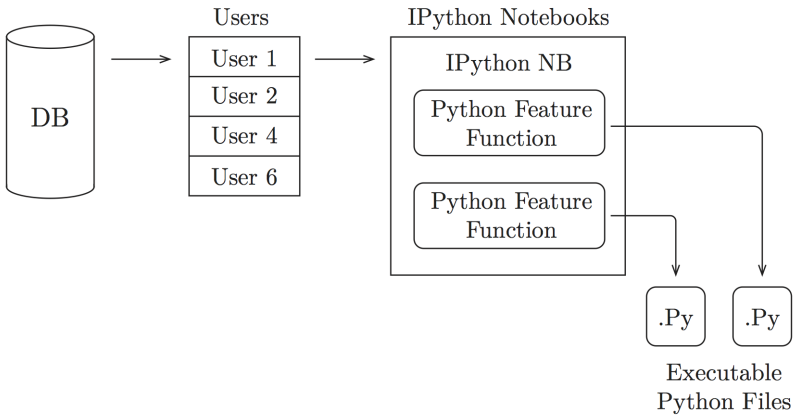

Figure 2-2: In this figure, we visualize the feature aggregation infrastructure. The process begins by reading from the MySQL database which users have written features for a given machine learning problem and their associated IPython notebooks. The IPython notebook files are parsed, and the feature functions the users wrote within the notebooks are compiled into individual executable python files.

2.2.3 Leaderboard

To add additional incentive for users to write good features, Feature Factory maintains a leaderboard for each machine learning problem. The features that users have written are ranked by a specific metric (accuracy, AUC, etc) depending on the type of machine learning problem. The rankings are published online by the Feature Factory website. The motivation behind having a leaderboard with Feature Factory is to promote a sense of friendly competition within the user-base. Given that all parts of Feature Factory work properly together, users will be able to rapidly create features, extract and test them, and see how they stack up against everyone else.

Summary In this chapter we presented the different components we built as part of the feature factory. We paid careful attention to the requirements we placed as a result of noticing the limitations of Kaggle. This is a first step towards developing a system of this kind. We certainly expect to make improvements and develop it further.

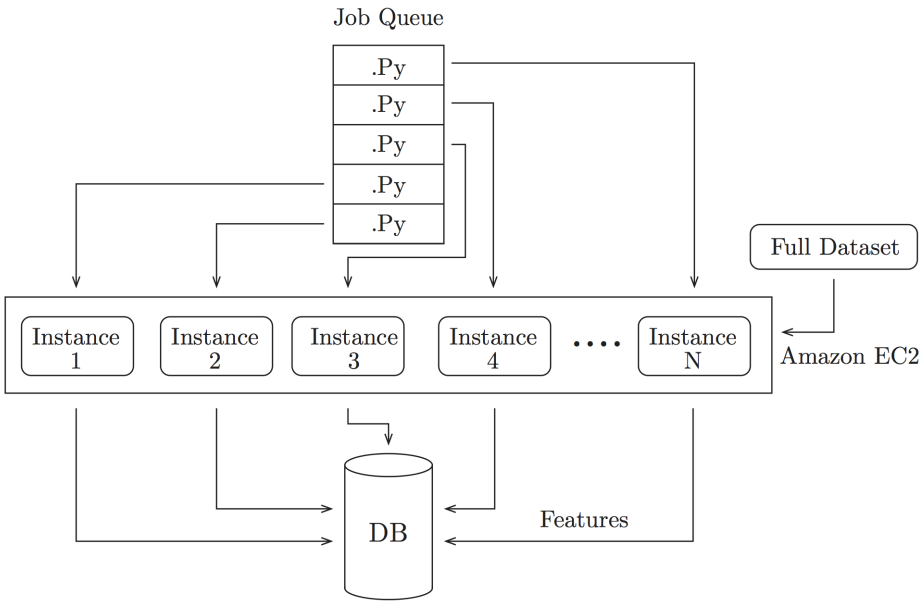

Figure 2-3: In this figure, we visualize the feature extraction infrastructure. After the feature aggregation module executes, the system is left with a series of executable python files, with each file representing a feature function waiting to be applied to the full dataset. These files are placed into a queue. Feature Factory then leverages multiple cores on an Amazon EC2 machine to pop off a job from the queue and execute it. In the case that the number of jobs overwhelms the number of cores available on the Amazon EC2 machine, Feature Factory simply spins up another EC2 machine and applies the same feature extraction process.

Chapter 3

Uploading New Machine Problems to Feature Factory

Feature Factory is designed to be able to integrate any machine learning problem into its infrastructure. To do this however, very specific steps need to be followed in order to ensure that no part of the system breaks. Specifically, to integrate a problem, three different parts are required: pre-processing the dataset, writing problem-specific code within the Feature Factory system-side infrastructure, and creating a template IPython notebook. Each of these parts are documented in this chapter.

3.1 Data

Once a machine learning problem has been provided either by a company or an institution, the first step to integrating it into the Feature Factory platform is to organize and configure the dataset. Feature Factory accepts data files in CSV format, so any dataset that isn't in CSV format would first need to be transformed to fit the requirements of Feature Factory. After this is accomplished, a sample dataset needs to be created. This is done by subsampling the original dataset. The sample datasets contain between $1\% to 10\%$ of the original dataset, depending on the size of the original dataset. The general rule of thumb is to provide users as much data as possible to

enhance their ability to create features but also maintain a reasonable size such that users do not have to wait long periods of time for their code to finish executing.

3.2 Problem-Specific Code

Feature Factory is designed in a fashion such that each new machine learning problem requires minimal code changes. When configuring a new machine learning problem, the following function must be implemented:

```
1 def preprocess_dataset():
2     pass
```

`preprocess_dataset()` is a system-side infrastructure function responsible for the loading of CSV files into memory as Numpy matrices as well as preprocessing the dataset. Two examples of common preprocessing tasks are hashing sensitive data fields (e.g. bank account numbers) and pruning unnecessary or redundant fields.

3.3 Template IPython Notebook

The last step in configuring a machine learning project to work with Feature Factory is to create a template notebook. The job of the template notebook is to provide step-by-step instructions on how to create a feature for the machine learning problem. At first, it was hypothesized that providing users with detailed instructions on the main Feature Factory website would be sufficient to educate new users on how to use the Feature Factory system. However, through testing, it quickly became evident that the best way to educate new users as quickly and efficiently as possible about a new problem is to simply give them a working example. The parts of the basic template for an exemplar machine learning problem are shown and described in the following subsections.

3.3.1 Setup and Registration

The first step to using Feature Factory is to import the infrastructure into the Python kernel. To do this, users must do the following:

```
1 from IJCAI_factory import *
```

As one can see, the module in this case is named IJCAI_factory, and hence is the infrastructure specific to the IJCAI machine learning problem documented in the next chapter. Each machine learning problem has a specific backend implementation that is mostly derived from a base implementation with exceptions to handle any particular intricacies.

The second step is to either login or create a new user. This is done by executing one of the following commands:

```
1 commands.create_user('johndoe', 'password')
2 commands.login('johndoe', 'password')
```

After this is completed, the user must register a specific IPython Notebook by giving it a name and executing the following function.

```
1 commands.add_notebook(‘notebook\_name\_here')
```

This registration lets Feature Factory know that there is a new notebook of the given name and that it belongs to the currently logged-in user, which in this case is johndoe.

3.3.2 Sample Dataset and Example Feature

One the user has setup his or her account and notebook, the user must load the sample data. To do this, the following command is used.

```
1 dataset = commands.get_sample_dataset(form='pandas')
```

Currently, Feature Factory only has two options for the form in which data can be retrieved: pandas dataframes and numpy matrices. This can be specified via the form parameter in the command.

After the sample dataset has been retrieved, the user can progress to taking a look at an example feature such as the one shown below for the IJCAI machine learning problem.

```
commands.add_feature('example_feature')

def example_feature(dataset):
    user_log = dataset[2]
    user_ids = user_log['user_id'].unique()
    print 'we have ' + str(len(user_ids)) + ' users'
    merchant_ids = dataset[0]['merchant_id'].unique()
    feature = {}
    for user_id in user_ids:
        feature[user_id] = {}
        user_merchant_ids = dataset[0][dataset[0]['user_id'] == user_id ↩
    ]['merchant_id'].unique()
        for merchant_id in user_merchant_ids:
            actions = user_log[(user_log['user_id'] == user_id) &( ↩
    user_log['seller_id'] == merchant_id)]['action_type'].values
            if len(actions) > 0:
                num_zeros = len(actions) - len(np.nonzero(actions))
                feature[user_id][merchant_id] = num_zeros
            else:
                feature[user_id][merchant_id] = 0.0
    result = []
    for key in sorted(feature.keys()):
        for key2 in sorted(feature[key].keys()):
            result.append([key, key2, feature[key][key2]])
    return np.matrix(result)
feature = example_feature(dataset)
```

In this particular case, the user registers the feature, named example_feature, via the `commands.add_feature` function. After the feature has been registered, it is de-

fined in a python function, taking in the dataset as the sole parameter. The feature function performs the necessary operations on the dataset to retrieve feature values, and returns them in a numpy matrix. The last line of the code block simply applies the example_feature on the sample dataset retrieved via a prior command and saves the results into an object named "feature".

3.3.3 Training and Testing

Finally, after the user has extracted a feature from the sample dataset, it is time to train and test a preselected machine learning algorithm using the feature as input. Since Feature Factory provides machine learning purely as a service, this is effectively accomplished by executing one command.

```
1 X = feature
2 Y = labels #labels are derived from the dataset
3 commands.cross_validate(X, Y, num_folds = 3)
```

`commands.cross_validate` trains and tests a preselected classifier using k-fold cross validation and outputs an accuracy or AUC, depending on the type of problem.

The training and testing completes the machine learning pipeline that the users go through in their IPython notebook. It is expected that users go through the process of registering a feature, creating a feature function, and testing it many times. Hence, a lot of time was put into making this process as clear and simple as possible.

Chapter 4

Feature Factory: First trials

In this chapter, we document the approach and execution of our first trials testing Feature Factory with users from our lab, Any Scale Learning for All (ALFA). Specifically, we document the end-to-end process of integrating, presenting, and testing three different machine learning problems using Feature Factory.

4.1 Experiment methodology

Users Ultimately, our goal with Feature Factory is to expand the number of users that participate in this part of the data science endeavor. We would also like to learn from their experiences using the platform. Hence, in order to test whether or not Feature Factory is a feasible idea for crowd-sourced feature engineering, we performed group Feature Factory exercises with lab-mates at the ALFA group at MIT's Computer Science and Artificial Intelligence Laboratory over the course of 1 month. The students who obliged to help us test Feature Factory all had backgrounds in computer science, were proficient in Python and Numpy, and had all completed at least one machine learning course beforehand.

Machine Learning Problems To test Feature Factory, we chose three very different machine learning problems. The motivation behind choosing extremely differ-

ent problems was to test whether or not Feature Factory could handle the different intricacies within each problem without requiring massive changes to the overall infrastructure design. Specifically, the main challenge at hand was to test whether or not Feature Factory could be easily adapted to different types and sizes of datasets. Extremely large datasets would require massive subsampling as well as possibly lead to performance issues. Different types of datasets, such as timeseries and relational datasets, would require specific adjustments to ensure the correctness of the Feature Factory pipeline.

The three problems chosen were:

- KDDCup 2014, a yearly competition hosted by ACM SIGKDD as part of the KDD conference - binary classification
- Route Prediction - multi-label classification
- IJCAI Repeat Buyer Competition, a competition sponsored by Alibaba and hosted by the International Joint Conferences on Artificial Intelligence - binary classification

As one can see, two of these problems (KDDCup and IJCAI) were taken from public machine learning competitions. The route prediction problem was taken from within one of ALFA"s data science projects. These problems were integrated into the Feature Factory system and presented to the selected test users. The users were asked to write features for each problem, and provide feedback on the entire Feature Factory experience.

4.2 KDDCup 2014

DonorsChoose.org is an online charity that helps students in need via school donations. Teachers in K-12 schools throughout the world can, at any time propose projects to improve their students' education. People can go online, view the projects,

and donate to them. When a project reaches its funding goal, DonorsChoose ships the materials necessary for the project to the school.

The machine learning problem defined by the organizers of ACM knowledge discovery and data mining conference organizers is to help DonorsChoose.org identify projects that are particularly exciting and interesting. By identifying these projects early, DonorsChoose can help improve funding outcomes, improve the user experience, and overall increase the number of students that the project reaches. In order for a project to be identified as exciting, it must meet all five of the following criterion

- Project has to be fully funded
- Project has to have at least one teacher-acquired donor
- Project has to have a higher than average percentage of donors leaving an original message
- Project has to have at least one green donation
- Project has to have one or more of the following criterion
  - have donations from three or more non teacher-acquired donors
  - have one non teacher-acquired donor give more than $100
  - have a donation from a thoughtful donor

4.2.1 Dataset

Full Dataset The full KDDCup dataset was presented in a relational format, spread across 5 different tables. The tables and their respective fields are shown in Table 4.1, Table 4.2, Table 4.3, Table 4.4, and Table 4.5.

Field	Description
is_exciting	ground truth of whether a project is exciting from business perspective
at_least_1_teacher_referred_donor	teacher referred = donor donated because teacher shared a link or publicized their page
fully_funded	project was successfully completed
at_least_1_green_donation	a green donation is a donation made with credit card, Paypal, Amazon, or check
great_chat	project has a comment thread with greater than average unique comments
three_or_more_non_teacher_referred _donors	non-teacher referred is a donor that landed on the site by means other than a teacher referral link/page
one_non_teacher_referred_donor _giving _100_plus	see above
donation_from_thoughtful_donor	a curated list of 15 donors that are power donors and picky choosers (we trust them selecting great projects)
great_messages_proportion	how great_chat is calculated. proportion of comments on the project page that are unique. If > avg (currently 62%) then great_chat = True
teacher_referred_count	number of donors that were teacher referred (see above)
non_teacher_referred_count	number of donors that were non-teacher referred (see above)

Table 4.1: Outcomes table within KDDCup dataset

Field	Description
donationid	unique donation identifier
projectid	unique project identifier (project that received the donation)
donor_acctid	unique donor identifier (donor that made a donation)
donor_city	city of the donor
donor_state	state of the donor
donor_zip	zip code of the donor
is_teacher_acct	donor is also a teacher
donation_timestamp	date and time of the donation
donation_to_project	amount to project and excluding optional support (tip)
donation_optional_support	amount of optional support
donation_total	donated amount
dollar_amount	donated amount in US dollars
donation_included_optional_support	whether optional support (tip) was included for DonorsChoose.org
payment_method	what card/payment option was used
payment_included_acct_credit	whether a portion of a donation used account credits redemption
payment_included_campaign_gift_card	whether a portion of a donation included corporate sponsored giftcard
payment_included_web_purchased_gift_card	whether a portion of a donation included citizen purchased giftcard (ex: friend buy a giftcard for you)
payment_was_promo_matched	whether a donation was matched 1&1 with corporate funds
via_giving_page	donation given via a giving / campaign page (example: Mustaches for Kids)
for_honoree	donation made for an honoree
donation_message	donation comment/message. Used to calculate great_chat

Table 4.2: Donations table within KDDCup dataset

Field	Description
resourceid	unique resource id
projectid	project id that requested resources for a classroom
vendorid	vendor id that supplies resources to a project
vendor_name	name of the vendor
project_resource_type	type of resource
item_name	resource name (ex: ipad 32 GB)
item_number	resource item identifier
item_unit_price	unit price of the resource
item_quantity	number of a specific item requested by a teacher

Table 4.3: Resources table within KDDCup dataset

Field	Description
projectid	project's unique identifier
teacher_acctid	teacher's unique identifier (teacher that created a project)
schoolid	school's unique identifier (school where teacher works)
school_ncesid	public National Center for Ed Statistics id
school_latitude	latitude of school
school_longitude	longitude of school
school_city	city of school
school_state	state of school
school_zip	zip of school
school_metro	metro of school
school_district	district of school
school_county	county of school
school_charter	whether a public charter school or not (no private schools in the dataset)
school_magnet	whether a public magnet school or not
school_year_round	whether a public year round school or not
school_nlns	whether a public nlns school or not
school_kipp	whether a public kipp school or not
school_charter_ready_promise	whether a public ready promise school or not

Table 4.4 – to be continued

Field	Description
teacher_prefix	teacher's gender
teacher_teach_for_america	Teach for America or not
teacher_ny_teaching_fellow	New York teaching fellow or not
primary_focus_subject	main subject for which project materials are intended
primary_focus_area	main subject area for which project materials are intended
secondary_focus_subject	secondary subject
secondary_focus_area	secondary subject area
resource_type	main type of resources requested by a project
poverty_level	school's poverty level (highest: 65%+ free of reduced lunch, high: 40-64%, moderate: 10-39%, low: 0-9%)
grade_level	grade level for which project materials are intended
fulfillment_labor_materials	cost of fulfillment
total_price_excluding_optional_support	project cost excluding optional tip that donors give to DonorsChoose.org while funding a project
total_price_including_optional_support	see above
students_reached	number of students impacted by a project (if funded)
eligible_double_your_impact_match	project was eligible for a 50% off offer by a corporate partner (logo appears on a project like Starbucks or Disney)
eligible_almost_home_match	project was eligible for a 100 boost offer by a corporate partner
date_posted	data a project went live on the site

Table 4.4: Projects table within KDDCup dataset

Sample Dataset The sample dataset was created by sampling approximately 1% of the full dataset. Specifically, the original dataset contained all the data related to 664099 unique projects, and the sample dataset contained only data relevant to

Field	Description
projectid	unique project identifier
teacher_acctid	teacher id that created a project
title	title of the project
short_description	description of a project
need_statement	need statement of a project
essay	complete project essay

Table 4.5: Essays table within KDDCup dataset

6625 unique projects.

4.2.2 Integration with Feature Factory

Integrating the KDDCup machine learning problem with Feature Factory turned out to require a bit of effort. In addition to having a very large dataset (roughly 4 GB), the KDDCup dataset was also extremely relational, with data spread across multiple tables. Because of this format, the dataset required extensive and careful subsampling to generate a reasonable sample dataset while preserving relationships between the different data tables.

For example, within the KDDCup dataset, there is a table describing all of the resources for different projects. Hence, each project can be linked to multiple resources. Because of this relationship, figuring out a proper percentage of the data to use became more difficult, given that even if the number of projects was reduced to the desired amount, there is no guarantee that the number of resources related to the select projects is at a reasonable level as well. This is due to the fact that some fields had a many-to-one mapping to projects e.g. multiple resources per project. To overcome this, we simply used the table with the most rows to compute the percentage of data we want to select as the sample dataset. This guarantees that no matter what features users write, they can be extracted from the sample dataset within a reasonable amount of time, maintaining interactivity within Feature Factory.

4.2.3 Test-Train-Submission

The data was split into a training set and a test set using k-fold cross validation with $k = 3$. This means that the dataset was partitioned into three equal folds. The machine learning service performed training and testing three times, with each iteration leaving out one of the 3 folds to use as the test set.

4.2.4 Results

A total of 8 features were written by test users for the KDDCup machine learning problem. The 8 features were

- nb_teacher_donors - number of teacher donors for a project
- total_resource_price - total cost of resources for a project
- essay_length - number of characters in essay for a project.
- count_donations - number of donations for a project
- number_of_donors - number of donors that contributed to a project
- donation_optional_percent - percentage of all donations for a project that were optional
- total_item_cost - average cost of resources for a project

Table 4.6 depicts the performance (AUC) of the individual features as well as the performance of all the features combined.

4.3 Route Prediction

The route prediction problem was inspired from data that the ALFA lab received from a car company. Effectively, the car company was able to track the locations of a

Name	Feature Name	AUC (full dataset)	AUC(sample dataset)
User 2	nb_teacher_donors	0.5004393994	0.5204
User 2	total_resource_price	0.5000565464	0.50001
User 5	essay_length	0.5	0.5
User 4	count_donations	0.5004393994	0.5056
User 3	number_of_donors	0.5001108334	0.6011
User 1	donation_optional_percent	0.5226887754	0.5502
User 1	total_item_cost	0.5036362477	0.5012
Combined	all features combined	0.5786228501	0.6122

Table 4.6: Features for KDDCup and associated AUCs

specific set of cars over the course of 4 months. With this dataset, it became natural to question whether or not it is possible to build a model to predict destinations of these cars using data collected from prior trips.

4.3.1 Data Preprocessing

Before diving into creating a specific machine learning problem using this dataset, however, we first needed to define what exactly is a destination and how it is specified. To solve this, we decided to overlay a 100 meter by 100 meter grid on to the world map. Each dot on this grid represented a possible "destination", and had a specific integer label. After constructing this grid, the provided dataset from the car company, which gave locations in terms of latitude and longitude, was transformed into using integer x,y location coordinates on the 100 meter by 100 meter grid. To transform a latitude and longitude pair into integer coordinates, we simply used the nearest-neighbor method, picking the x,y coordinate on the grid closest to actual latitude and longitude position on the map. By performing this transformation on the dataset, we could then simply treat the prediction problem as a multi-label classification problem where the label of a trip was the x,y coordinate of the trip's final destination on the grid.

4.3.2 Problem

After preprocessing the dataset and discretizing location values as integers on the 100 meter by 100 meter grid, the route prediction problem can be summarized as follows:

Provided historical trip information, predict the destination label of a new trip for a given car.

4.3.3 Dataset

Full Dataset The full dataset was data collected from multiple days worth of trips for a variety of different cars. Each day had >= 1 trips. The dataset was organized as a set of folders, with each folder representing the data available for a particular car. Within each car folder was a set of folders, with each folder representing a specific day. Within each day folder, was a set of CSV files, with each file representing the location data collected during one trip. Each CSV file had four columns, shown in Table 4.7.

Field	Description
timestamp	Unix timestamp of data point
x	the X coordinate (integer value) of the car's location on a 100 m by 100 m grid
y	the Y coordinate (integer value) of the car's location on a 100 m by 100 m grid
label	the label of this trip's destination

Table 4.7: Route prediction dataset

Sample Dataset The sample dataset for the users was created by sampling a portion of the full dataset in terms of cars. Specifically, the sample dataset was all of the trip data for one specific car, whereas the full dataset had all of the trip data for 6 different cars.

4.3.4 Integration with Feature Factory

The route prediction machine learning problem required the most adjustments out of the three machine learning problems due to the time-series nature of the data.

Since one cannot use data after a trip or data collected during a trip to predict the trip's destination, a new data partitioning function had to be created to fulfill this additional constraint.

Instead of randomly partitioning the dataset into a training set and a test set, the new data partitioning function iterates through the dataset in chronological order. With each iteration, the function picks the current data point (a trip destination) as a test set, and creates a training set using all prior data points.

4.3.5 Results

Creating features for this problem was harder than other problems given the extremely simplistic nature of the dataset. The users had only three columns to work with, and hence, they had to draw on more creativity to make good features. A total of 5 features were created for this problem and are described below.

- time_traveled_start_location - given a trip, output the difference in trip length compared to the previous trip
- last_departure - given a trip, output the label of the start location of the previous trip
- total_day_driving - given a trip, output the total amount of time a driver has been on the road so far before the given trip.
- similar_length_trip - finds a prior trip with the closest trip time, and outputs the label of that prior trip.

The prediction accuracies of the features on each of the cars in the dataset are documented in Table 4.8. It should be noted that, in hindsight, further preprocessing of

the data was necessary. In particular, the data should have been better partitioned such that the training data contained information regarding all but a selected portion of trips. For the selected portion of trips, only the label of the destination would be provided in the training data. This would ensure that users could not utilize information from *within* a trip to predict its destination because, after all, the goal here is to predict destinations before trips begin.

User	User 1	User 2	User 7	User 7
Feature Name	time_traveled _start_location	last_departure	total _day_driving	similar _length_trip
Car 1	0.4680851064	0.4255319149	0.4042553191	0.3404255319
Car 2	0.2121212121	0.1515151515	0.101010101	0.1414141414
Car 3(sample)	0.3255813953	0.2209302326	0.1395348837	0.2325581395
Car 4	0.4025974026	0.2987012987	0.3506493506	0.3506493506
Car 5	0.246031746	0.2222222222	0.2222222222	0.2142857143
Car 6	0.2736842105	0.2631578947	0.1473684211	0.2736842105
Average	0.3213501788	0.2636764524	0.2275067163	0.2588361814

Table 4.8: Features for route prediction and associated prediction accuracies. Note that due to instructions that were not perfectly clear, the time_traveled_start_location and total_day_driving features are incorrect, since they use data from the trip whose destination we are trying to predict.

4.4 IJCAI

The International Joint Conference on Artificial Intelligence, or IJCAI for short, is the main international gathering of researchers in the field of artificial intelligence. This conference is held biannually on odd-numbered years in different locations around the world.

The IJCAI repeat buyer machine learning problem is a problem hosted by IJCAI and specifically sponsored by Alibaba Group. Specifically, this problem is focused on modeling consumer behavior to predict which consumers will be one time buyers from a specific merchant or will become repeated buyers.

4.4.1 Dataset

The IJCAI dataset contains historical user behavior for different merchants provided by Tmall.com in a relational format. The dataset is split into three different tables, shown in Table 4.9, Table 4.10, and Table 4.11.

Field	Description
user_id	A unique id for the shopper
merchant_id	A unique id for the merchant
label	It is an enumerated type 0, 1, where 1 means repeat buyer, 0 is for non-repeat buyer. This field is empty for test data.

Table 4.9: Training / Test data for IJCAI repeat buyer problem

Field	Description
user_id	A unique id for the shopper
age_range	User's age range: 1 for under 18. 2 for [18,24]; 3 for [25,29]; 4 for [30,34]; 5 for [35,39]; 6 for [40,49]; 7 and 8 for >= 50;0 and NULL for unknown.
gender	User's gender: 0 for female, 1 for male, 2 and NULL for unknown.

Table 4.10: User profile data for IJCAI repeat buyer problem

Field	Description
user_id	A unique id for the shopper
item_id	A unique id for the item
cat_id	A unique id for the category that the item belongs to.
merchant_id	A unique id for the merchant.
brand_id	A unique id for the brand of the item.
time_tamp	Date the action took place (format: mmdd)
action_type	It is an enumerated type 0, 1, 2, 3, where 0 is for click, 1 is for add-to-cart, 2 is for purchase and 3 is for add-to-favourite.

Table 4.11: User behavior log data for IJCAI repeat buyer problem

4.4.2 Integration with Feature Factory

Given that the IJCAI dataset is given in a relational table format similar to KDDCup, the integration process with Feature Factory was effectively identical.

4.4.3 Results

A total of 10 features were written by users for the IJCAI machine learning problem. The 10 features were

- purchase_to_click ratio - ratio of a user's clicks to a user's purchases for all items with a merchant.
- duration - number of days between the earliest date that a user performed an action (click, add-to-cart, purchase, add-to-favorites) with a merchant and the latest date that a user performed an action with the same merchant
- actions_count - number of actions that a user performed with any of a merchant's products
- item_count - number of unique items that a user performed an action on with a merchant
- cat_count - number of unique categories of items that a user performed an action on with a merchant
- brand_count - number of unique brands of items that a user performed an action on with a merchant
- strong_item_count - number of unique items that a user performed a strong action (added-to-cart, purchased, or added-to-favorites) on with a merchant
- strong_cat_count - number of unique categories of items that a user performed a strong action on with a merchant

- strong_brand_count - number of unique brands of items that a user performed a strong action on with a merchant

The performance of these features individually, as well as combined, are shown in Table 4.12.

User	Feature Name	AUC (full dataset)	AUC (sample dataset)
User 1	num_categories	0.5004	0.5899
User 6	purchase_to_click_ratio	0.501	0.4802
User 5	duration	0.5	0.50001
User 5	actions_count	0.5	0.5
User 5	item_count	0.50004	0.5
User 5	cat_count	0.50006	0.5001
User 5	brand_count	0.50003	0.50002
User 5	strong_item_count	0.5	0.5
User 5	strong_cat_count	0.50002	0.50003
User 5	strong_brand_count	0.50002	0.50001
User 5	User 5 combined	0.5009*0.5801	
All users	Combined	0.501	0.6011

Table 4.12: Features for IJCAI repeat buyer problem and associated AUCs

4.5 Lessons Learned from Trials

The first trials with Feature Factory were extremely useful as they provided an abundant amount of useful lessons and feedback. The major lessons that were learned are documented here.

4.5.1 Feature Quality

In our trials, it quickly became evident that most of the features users wrote were simple and more direct features rather than complex and derived features. Although this was reasonable for our experiment, given the goal was to simply test the Feature

Factory system rather than get good features, it is also useful to analyze reasons why users chose the easier route.

The reason for this kind of behavior is twofold. The first reason for is the lack of incentives for users to write more complex features. Writing feature functions is not a trivial task, and users had no incentive to put additional effort into the feature discovery process.

The second reason is a result of the current front-end design of Feature Factory. Although Feature Factory provides an online repository of IPython Notebooks such that users can view the features that other users have created, Feature Factory does not provide a clear overview of all features created so far. The lack of overview along with the extra energy required for users to read other user's feature functions led to users simply writing the first features that came to mind, with no regard of whether or not the feature was created already.

In the future, incentives via gamification can be used to push users to write better features, and an easy-to-use UI can be created to allow users to easily browse features and determine which features already exist. Additionally, a number of *simple* features will be pre-extracted to focus users attention on more complex features.

Finally, it is important to note that most of the trials were done in an hour time frame which is a really short time frame for running a real feature discovery experiment.

4.5.2 Feedback on User Experience

Overall, the users in our trials provided primarily positive feedback on the entire user experience. Users stated that they had very little trouble understanding the IPython Notebook templates and instructions, and that the machine learning problems were well-described.

One criticism that users had, however, was the time it took to return accuracies and AUC's on the full dataset for features that users have submitted. Although users understood that they had a sample dataset to play with, there is obvious noise

between the performance of their feature on the sample versus the full dataset.

Another criticism was the lack of support for another languages and tools to construct features. One user enjoyed writing feature functions in SQL and felt that Python was unwieldy to use for feature discovery. These criticisms were extremely useful, and definitely play a large part in the direction Feature Factory will take in the future as documented in Chapter 6.

Summary: Our goal through these first trials was to test if users can use the system. A number of questions arose as we were designing the system: "*Would we be able to covey the complexities of a data science problem through our framework*", "*Would we be able to get feature scripts that work?*", *"Would the subsampling based strategy work?*", *"Would users follow the abstractions we specify for the feature functions?"*. After successfully finishing three experiments we are confident that this system is generalizable and would work for any data science endeavor.

Additionally, our foundational challenge is that we are crowd sourcing to get software and not simply the output (for example, simply getting predictions as is the case with Kaggle). Although we have clearly described the inputs and outputs of the software, and given example code, this challenge brings about a number of questions- "*Would the submitted scripts work on the larger dataset?*", *"What if the code is inefficient?"*. In next chapter, we present some of the interesting things we observed.

Chapter 5

Challenges Faced

Feature Factory is the first of its kind in terms of allowing users to: (1) write scripts for making features, (2) test features on a sample (not whole dataset) of the dataset and (3) share scripts. Hence, we encountered a number of challenges. We also note, that most of the challenges can be attributed to users working on a *sub-sample* of the dataset. Below we list, the major challenges and describe these challenges in detail and the methods we used to overcome some of these:

- Generating a coherent data sample from linked data model
- Debugging of User Submitted Features
- Noisy estimates of feature Accuracy
- Compute challenges with feature extraction

5.1 Example problem

To demonstrate concretely some of the problems that were encountered when developing Feature Factory, we will use the KDDCup machine learning problem mentioned before. The KDDCup machine learning problem was comprised of highly relational

data about proposed projects to help school children, with the ultimate goal to predict whether or not these proposed projects would be exciting. Whether or not a proposed project was ultimately labeled exciting or not exciting was based on a set of criterion described in Chapter 4.

5.2 Generating a coherent data sample

In Feature factory, for each machine learning problem, users are provided within their IPython Notebooks a sample dataset. The size of the sample dataset is usually anywhere between 1 to 10 percent of the true dataset size. Subsampling a dataset seems trivial on paper, but in practice, actually leads to several unforeseen issues. In particular, data sampling becomes more difficult when the data is in a highly relational format. For example, in the KDDCup dataset, the main entity to subsample is project IDs, since the goal of the problem is to classify projects as exciting or not exciting. However, to do this, not only do we need to select a select portion of project IDs, but we also need to ensure that we find all of the other entities related to the project ids that we select. For example, each project often has a variety of different donations, resources, and essays. When subsampling the dataset, for each project id that we choose to include in our sample dataset, we must go and find all of the entries in each of the other entities that relate to that particular project id and include it in our sample dataset.

5.3 Debugging user submitted features

One of the problems we discovered was the fact that users may write buggy feature extraction functions that could wind up throwing exceptions midway through the feature extraction process. This not only causes the feature extraction process to fail, but also wastes computation time given that feature extraction often takes a significant amount of time when they are applied to the larger dataset. Most common

bugs and problems found with user submitted scripts were:

Incorrect feature vector dimensions When performing any mathematical matrix operations in Python, it is critical that the dimensions of the matrices align properly or else the code will exit with an error. When dealing with extracting features from a dataset, the matrix misalignment issue occurs relatively frequently, which in testing, caused the Feature Factory infrastructure to break multiple times. The reason for this problem is because users often rely on a related entity in the relational dataset to extract a feature for the entity in question. Sometimes, the related entity may not exist for a certain element of the entity in question This is demonstrated in the following example.

Example 5.3.1. Using the same relational dataset example as before, suppose the user wanted to create a feature based on the resources table. The user would effectively grab all unique project id's within the resources table and extract a feature value for each of these project IDs. However, in the dataset, not all projects have resources, and hence the set of unique project IDs within the resources table is actually a subset of the number of project IDs in total. This becomes a problem when feature factory attempts to train and test the feature because the feature matrix is missing rows. Not only that, but when the feature matrix is missing (or has too many) rows, the feature matrix becomes effectively useless because Feature Factory has no way of reconciling which projects the feature matrix is missing.

Solution: Hence, to tackle this issue, a requirement on the output of users' feature functions was added. Users are now required to output an unique identifier for each row of the feature matrix as the first x columns, where x is the number of values in the unique identifier. For example, the unique identifier in the KDDCup dataset is the project_id. Similarly, the unique identifier in the IJCAI dataset is the user_id, merchant_id pair. These unique identifiers are critical since they allow Feature Factory to perform simple post processing on the users features. Post processing within Feature Factory is very dependent on the nature of the machine

learning problem at hand, but in general, the goal of the post processing step is to simply ensure that the user's feature matrix aligns perfectly with the label such that training and testing can be performed. If the user's feature matrix has too few rows, a generic null value is inputted for the missing rows. If the user's feature matrix has too many rows, the matrix is pruned such that there is only one feature value per unique identifier. A diagram showing the post-processing procedure is shown in Figure 5-1.

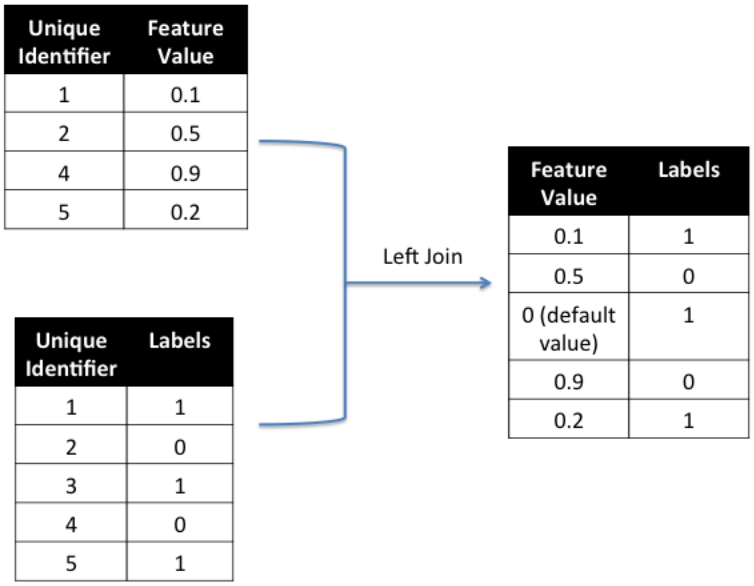

Figure 5-1: This figures shows an example of the alignment procedure in post-processing. The top left table depicts the output of a user's feature function applied to the full dataset. The bottom left table depicts the labels. Notice that the top left table does not have an entry with a unique identifier of value 3, but the bottom left table does. This implies that a feature value is missing. In this situation, te alignment procedure performs a left join on the two tables, keeping all rows from the table containing the labels, and filling in a default value of 0 as a feature value for the row with a unique identifier of 3. This ensures that the feature matrix and labels matrix align when performing training and testing.

Arithmetic and statistics errors: When computing features, there is undoubtedly an abundance of statistical and arithmetic calculations involved. One of the problems that arose with these calculations, however, is that sometimes due to the

dataset, the calculations become invalid and Python ends up throwing an exception, breaking the feature extraction process. This is extremely detrimental to the feature extraction process since feature functions can often run for many hours, and having it break down in the middle effectively can undo many hours of computation time. This is further explained in the following example.

Example 5.3.2. The most common error in practice was the notorious divide by 0 error. The code excerpt below is an example from testing where after 20 hours of computation time, the code terminated due to a divide by 0 error from line 2.

```
1 for key in sorted(donation_optional_percentage.keys()):
2     # possible divide by zero error
3     result.append([key, float(donation_optional_percentage[key][0])/ ↩
      float(donation_optional_percentage[key][1])])
4 return np.matrix(result)
```

Solution: Given that Feature Factory cannot anticipate every single type of possible error, there isn't an obvious solution that Feature Factory can implement in the backend to ensure that user created feature functions do not break. Hence, the solution that Feature Factory currently uses is to advise users to wrap any calculations in a try, except block and set a default feature value for the iterations where the an exception is thrown. An example is shown in the code excerpt below.

```
1 for key in sorted(donation_optional_percentage.keys()):
2     try: # wrap possibly buggy line with try statement
3         result.append([key, float(donation_optional_percentage[key ↩
      ][0])/float(donation_optional_percentage[key][1])])
4     except Exception as e:
5         # append null value of 0 if there is an error
6         result.append([key, 0.0])
7 return np.matrix(result)
```

In this example, the possibly faulty divide operation is wrapped with a try, except block. If an exception is thrown, the code simply appends a default value of 0 rather

than throwing an error and exiting completely.

5.4 Noisy Estimates of Feature Accuracy

As mentioned before, users are provided within their IPython Notebooks a sample dataset. The users are provided with commands that allow them to extract the feature from the sample dataset and also train and test a pre-selected machine learning algorithm on the feature. The issue that arose was that the accuracies or AUC's that the users would get from the training dataset were extremely noisy. Since the subsampling procedure of Feature Factory simply aims to reduce the dataset size, it subsamples with no regards for the distribution of the data within the dataset. Hence, in testing, the accuracies and AUC's that users received from testing on the sample dataset were often far from the true accuracies and AUC's.

Though this seems like a significant problem, it actually turns out to be not that important considering the primary goals and objectives of Feature Factory. The main goal behind Feature Factory is to provide a platform where users can generate new ideas for features as fast as possible. Rather than focusing on one feature and attempting to fine tune it, Feature Factory would like users to brain-dump as many ideas as possible. Once they have dumped their ideas and submitted them, Feature Factory does apply their feature functions on the What is 'remote'?dataset and relay back their true results. Hence, having relatively noisy estimates of feature accuracy is relatively tolerable.

5.5 Compute challenges with feature extraction

Computing user generated features on the full dataset was by far the biggest challenge in developing Feature Factory. Given the size of the datasets that were tested, some of the features that users wrote took over 30 hours to extract. To make things worse, as mentioned before, some of these features had bugs that would cause the process

to crash after already running for a significant amount of time. After getting enough submissions, we noticed the following problems with the user generated scripts:

- Users did not vectorize their operations
- Users did not notice intricacies with built-in python functions that decreased the performance of their code

Solution: To solve this problem, we create an IPython Notebook, documenting a variety of *tips and tricks* that users could use to write more efficient feature functions. The major tips were as follows:

Vectorized operations using Numpy or Pandas Vectorization is absolutely critical when performing any sort of computation or manipulation on a large amount of data. Vectorization is effectively generalizing operations on scalars to also apply seamlessly to vectors and matrices as well. A good example of an operation that benefits greatly from vectorization is the dot product. In a dot product, the multiplication step can be done in parallel each multiplication operation is independent from every other multiplication operation. Normally, with normal Python, one would write a loop to multiply elements one pair at a time. However, with vectorization, the multiplication operations can be all done in parallel using Numpy's vectorized dot product operation, heavily decreasing overall runtime. This is shown in the code block below.

```
1 import numpy as np
2
3 def non_vectorized_dot_product(a, b):
4     #A and B are vectors of same length
5     total = 0
6     for i in range(len(a)):
7         total += a[i]*b[i]
8     return total
9
```

```
10 def vectorized_dot_product(a, b):
11     #A and B are vectors of same length
12     total = np.dot(a,b)
13     return total
14
15 a = np.random.rand(100000) # generate 100000 random values
16 b = a
17
18 t1 = time.time()
19 non_vectorized_dot_product(a,b)
20 print time.time() - t1 # took 0.060693025589 seconds to execute
21
22 t1 = time.time()
23 vectorized_dot_product(a,b)
24 print time.time() - t1 # took 0.000135898590088 seconds to execute
```

As one can see, the vectorized implementation is many orders of magnitude faster. Hence, vectorization is critical in writing efficient feature functions.

Intricacies with built-in python functions One of Python's biggest strengths is its concise and easy-to-read syntax. However, the simplicity of the syntax also often leads to certain ambiguities, especially when dealing with performance. During tests, it quickly became evident that many users do not fully understand the cost of calling certain Python functions. Some of the major misconceptions were:

1. Using `keys()` to check for keys in a dictionary.
2. Generators vs. Iterators

Misconception 1 While it seems intuitive to utilize `keys()` to check whether or not a key exists in a dictionary, it is actually extremely inefficient since keys() actually returns the entire list of keys. This operation takes $O(n)$ time, which is clearly suboptimal. Instead, users should simply check if the key is in the dictionary,

without any call to `keys()`. This takes $O(1)$ time. This is demonstrated in the code block below.

```
1 if key in dictionary.keys(): #no
2 if key in dictionary: #yes
```

Misconception 2 A very subtle but important concept when using Python is the idea of generators vs. iterators. Iterators are general objects that allow for iteration. Two common examples of objects that are iterators are lists and dictionaries. One can iterate through the elements of a list as well as the keys of a dictionary. Generators are a specific type of iterator, one that calculates values on the fly. This is known as lazy evaluation. By calculating values as they are needed, generators do not incur the memory cost that normal iterators do. This is extremely important when writing Python programs that deal with large amounts of information. An example with iterators and generators is shown in the code block below.

```
1 import time
2 # iterator implementation
3 def iterator():
4     my_list = [x**2 for x in range(100000)]
5
6 # generator implementation
7 def generator():
8     my_list = (x**2 for x in range(100000))
9
10 t1 = time.time()
11 iterator()
12 print time.time() - t1 # took 0.0187940597534 seconds
13
14 t1 = time.time()
15 generator()
16 print time.time() - t1 # took 0.00206398963928 seconds
```

As one can see, the generator implementation was a whole order of magnitude faster than the iterator implementation.

5.6 Takeaways

Overall, Feature Factory faced some major challenges along the development process. Since Feature Factory provides users a lot of freedom in which they define their features, this leads to a lot of specific edge cases in which the Feature Factory infrastructure can break. On top of this, in some cases, users don't necessarily write *wrong* code, but have an implementation that is far too inefficient to apply to extremely large datasets. To overcome these challenges, we altered Feature Factory infrastructure to be more robust against flawed code as well as chose to provide as much additional resources as possible to aid users in writing efficient and bug-free feature functions. Through some preliminary trials with test users, the additional resources and improved infrastructure made a very evident difference in the quality of code that users submitted, leading to a much better Feature Factory experience.

Chapter 6

Conclusion and Future Work

Feature Factory is a working proof-of-concept, demonstrating an easily-accessible platform to effectively crowd-source feature engineering. Not only does Feature Factory provide users a fully functional machine learning pipeline, starting from users writing features to actual results, Feature Factory also narrows the users' focus onto creating effective features rather than tinkering with different classifiers and their respective parameters. By constructing an interactive, intuitive interface, along with a flexible cloud architecture, Feature Factory shows significant potential to become a major tool in the open source machine learning community.

6.1 Future Work

Given that the initial experiments with Feature Factory were a success, there is an abundance of work that can be done to improve the system and deploy it to the general public. The main areas that Feature Factory need work on are documented below.

Analytics Engine Once users have written an abundant amount of features for a particular machine learning problem, the problem of feature selection arises. Since Feature Factory promotes writing as many features as possible, it is expected that

a high percentage of the features a particular user submits will be highly correlated with an existing feature in the system. Hence, an analytics engine will be required in addition to the existing pipeline to investigate different properties of the features that users have submitted. Feature Factory should make it easy for users to write a feature and quickly determined how original the feature is in relation to the existing features using a variety of metrics. This ability will greatly reduce the amount of redundant features that are written across different users.

Scalability Currently, Feature Factory is designed purely as a proof-of-concept, and has only been used concurrently by members of the ALFA group at MIT CSAIL. In these tests, each machine learning problem integrated within Feature Factory had its own dedicated Amazon EC2 instance. With this setup, users reported no performance issues with the system. However, if Feature Factory were to be released to a much larger crowd e.g. a set of beta users, it is unclear whether or not one single Amazon EC2 instance can handle the concurrent load. If this is the case, then additional infrastructure will be required, specifically a master-slave architecture to distribute the concurrent load for a single machine learning problem within Feature Factory.

Additional Programming Languages Although Python is an extremely popular language for data science and machine learning, some potential users might highly prefer different programming languages. Languages such as R, Ruby, SQL, and C++ are also common in the data science world, and ultimately Feature Factory should be improved such that these languages are supported such that certain users are not alienated from the platform.

Bibliography

[1] Crowdsourcing feature discovery. http://radar.oreilly.com/2014/03/crowdsourcing-feature-discovery.html. Accessed: 2015-01-30.

[2] Discover feature engineering, how to engineer features and how to get good at it. http://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/. Accessed: 2014-12-14.

[3] Discovery feature engineering. http://machinelearningmastery.com/discover-feature-engineering-how-to-engineer-features-and-how-to-get-good-at-it/. Accessed: 2014-07-30.

[4] Christopher M. Bishop. *Pattern Recognition and Machine Learning*. Springer, 2006.

[5] Justin Cheng and Michael S Bernstein. Flock: Hybrid crowd-machine learning classifiers.

[6] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The Elements of Statistical Learning*. Springer Series in Statistics. Springer New York Inc., New York, NY, USA, 2001.

[7] K.P. Murphy. *Machine Learning: A Probabilistic Perspective*. Adaptive computation and machine learning series. Mit Press, 2012.

[8] Vikas Chandrakant Raykar. Scalable machine learning for massive datasets: Fast summation algorithms. 2007.

[9] Chong Sun, Narasimhan Rampalli, Frank Yang, and AnHai Doan. Chimera: Large-scale classification using machine learning, rules, and crowdsourcing. *Proceedings of the VLDB Endowment*, 7(13), 2014.

[10] Kalyan Veeramachaneni, Kiarash Adl, and Una-May O'Reilly. Feature factory: Crowd sourced feature discovery. In *Proceedings of the Second (2015) ACM Conference on Learning@ Scale*, pages 373–376. ACM, 2015.

[11] Kalyan Veeramachaneni, Una-May O'Reilly, and Colin Taylor. Towards feature engineering at scale for data from massive open online courses. *arXiv preprint arXiv:1407.5238*, 2014.