The Data Science Machine: Emulating Human Intelligence in Data Science Endeavors

by

Max Kanter

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

Master of Engineering in Electrical Engineering and Computer Science

at the

MASSACHUSETTS INSTITUTE OF TECHNOLOGY

June 2015

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

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

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

The Data Science Machine:

Emulating Human Intelligence in Data Science Endeavors

by

Max Kanter

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

Abstract

Data scientists are responsible for many tasks in the data analysis process including formulating the question, generating features, building a model, and disseminating the results. The Data Science Machine is a automated system that emulates a human data scientist's ability to generate predictive models from raw data.

In this thesis, we propose the *Deep Feature Synthesis* algorithm for automatically generating features for relational datasets. We implement this algorithm and test it on 3 data science competitions that have participation from nearly 1000 data science enthusiasts. In 2 of the 3 competitions we beat a majority of competitors, and in the third, we achieve 94% of the best competitor's score.

Finally, we take steps towards incorporating the Data Science Machine into the data science process by implementing and evaluating an interface for users to interact with the Data Science Machine.

Thesis Supervisor: Kalyan Veeramachaneni
Title: Research Scientist

Acknowledgments

There a lot of people that made this thesis possible.

First, I'd like to acknowledge my adviser, Kalyan. When I started working with him well over a year ago as an undergraduate researcher, I had never worked on a research project at MIT. Reflecting now on all the time I've now spent in lab, it's easy to remember so many moments where his knowledge and passion inspired and led me. I'm truly grateful for all of his key ideas that positively impacted the direction of this thesis.

I'd especially like to thank my parents for being my biggest supporters. Feeling their continued encouragement and love has meant an immeasurable amount to me over the years. Without their good example, I wouldn't be turning in this thesis today.

Also, thank you to my labmates for being a pleasure to work with this year. I really enjoyed our conversations and the breaks we took to play pool in the CSAIL Recreation Zone. I'm excited to see what the future has in store for everyone.

Finally, I must say thank you to everyone I've lived with on Burton Third over the last four years. All the late nights and good memories have made my time at MIT very special.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Data science is an endeavor of deriving insights, knowledge, and predictive models from data. The endeavor also includes cleaning and curating at one end with dissemination of results at the other. Data collection and assimilation methods are sometimes included. Subsequent to development and proliferation of systems and software that are able to efficiently *store*, *retrieve*, and *process* data, attention has now shifted to analytics, both *predictive* and *correlative*. As a result, the number of scientists that companies are looking to hire has increased exponentially [1]. To address the immense shortage of data scientists, businesses are resorting to any means of acquiring data science solutions. Prominent among them is crowd sourcing. Kaggle and other well-reputed data science conferences have become venues for organizing predictive analytics competitions. For example, during the course of writing this thesis, three top-tier machine learning conferences, IJCAI, ECML and ACM-RecSys[1], have released datasets and announced competitions for building accurate, and generalizable predictive models. This is in addition to the ACM KDD cup organized by SIGKDD every year. Hence, our goal is to make these endeavor more efficient, enjoyable, and successful.

Many of these challenges posited either on Kaggle or via conferences have a few common properties. First, the data is structured and relational, usually presented

[1]IJCAI is the International Joint Conference on Artificial Intelligence. ECML is the European Machine Learning. ACM-RecSys is the ACM Conference on Recommender Systems.

as a set of tables with relational links. Second, the data captures some aspect of human interactions with a complex system. Third, the presented problem attempts to predict some aspect of our behavior, decisions, or activities (e.g., to predict whether a customer will buy again after a sale [IJCAI], whether a project will get funded by donors [KDD Cup 2014], or even taxi ride destinations [ECML]). In this thesis, we argue that data science endeavors for relational, human behavioral data remain *iterative*, *human-intuition driven*, *challenging*, and hence, *time consuming*.

1.1 What is Data Science?

To understand the challenges of an end-to-end data science endeavor for this type of data, we next ask: *What does a typical data scientist do?* and *What steps does it entail?*. To answer these questions, we draw from numerous data science exercises of our own, and in Fig. 1.1, we present the multiple activities a data scientist does when given a linked (relational) data source for working through a typical data science endeavor. Based on this, we delineate the following steps a data scientist takes:

Formulation of a question: The first task is to formulate an impactful question, either *predictive* or *correlative*. This is often an iterative process in itself. Assessment of whether or not the data could answer the question or whether the predictive problem defined is valuable often go hand-in-hand; therefore, it is often helpful to know apriori that the information needed to solve the predictive problem is in the captured data.

Feature synthesis: The next task is forming variables, otherwise known as features. This task is iterative in nature. The data scientist may start by using some static fields (e.g., gender, age, etc.) from the tables as features, then form some specialized features based on intuition of what might predict the outcome. With these simple features at hand, the scientist may, for example, resort to features that transform these raw features into a different space-percentile of a certain feature. The result of this step is usually an entity-feature matrix, which has

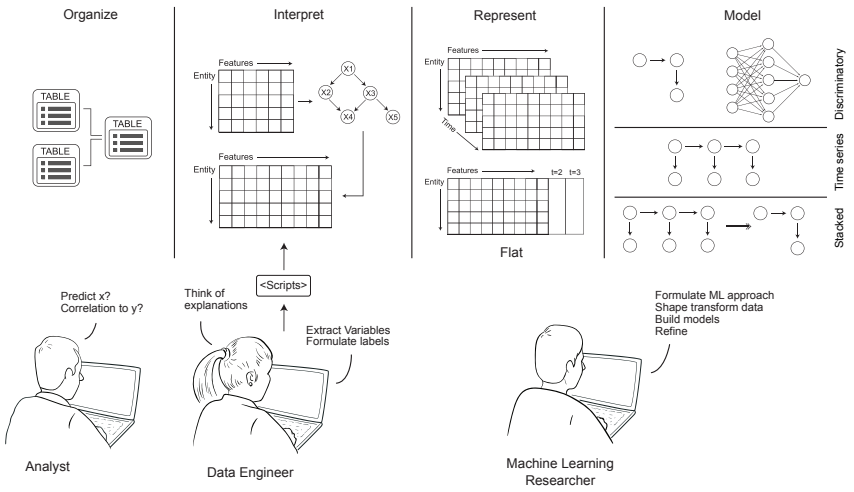


Figure 1-1: A typical data science endeavor. Previously, it started with an analyst positing a question: Could we predict if x or y is correlated to z? These questions are usually defined based on some need of the business or entity holding the data. Second, given a prediction problem, a data engineer posits explanatory variables and writes scripts to extract those variables. Given these variables/features and representations, the machine learning researcher builds models for given predictive goals and iterates over different modeling techniques. Choices are made within the space of modeling approaches to identify the best generalizable approach for the prediction problem at hand. A data scientist can span the entire process and take on the entire challenge: that is, positing the question and forming variables for building, iterating, and validating the models.

entities such as *transactions*, *trips*, *customers* as *rows* and features/variables as *columns*. If the data is longitudinal in nature, the matrix is assembled for each time slice.

Solve and refine using the machine learning approach: Once features are synthesized, the data scientist uses a machine learning approach. If the problem is of classification, a number of feature selection and classification approaches are available. The scientist may engage in classification methodology selection (*svm*, *neural networks*, etc.) and fine tune parameters. For a longitudinal problem, an alternative approach can be developed based on hidden Markov models and/or a stacked model where state probabilities from the learned hidden Markov model are used as features. Another alternative often pursued is to cluster the data and build cluster-wise models. In this step, one also engages in testing, validating, and reporting accuracy of the predictive model.

Communicate and disseminate results: Finally, the data scientist engages in disseminating outcomes and results. This often is beyond just predictive accuracy and involves identifying variables that were most predictive, interpreting the models, and evaluating model accuracy under a variety of scenarios, *for example*, different costs for false positives and negatives.

1.2 Why automate?

Through this thesis, we ask the following foundational questions: "Can we build a machine to perform most data science activities?", "What sort of technologies and steps are required to be able to automate these activities?", "How could automation aid in better discovery and enable people to have better and more rewarding interactions with the data?", "If built, how shall we measure success in developing a general-purpose data science machine?". If such a machine is built, identifying the roles humans/scientists will play is also critical.

We begin by noticing that much of feature formation, creating hypotheses to test,

and modeling could be automated, albeit that automation exposes many choices. To demonstrate this, we first present an automatic feature synthesis algorithm we call *Deep Feature Synthesis*. We show that while being automatic in nature, the algorithm captures features that are usually supported by human intuition. Once such features are generated for a dataset, we can then formulate questions, such as: *"Is this feature correlated with other features?"* or *"Can this subset of features predict the value for this feature?"*. When a time series is present in the data, we can ask, *"Can we predict the future value of a particular time series?"* These automated *question generation* mechanisms can, to a significant degree, create many questions that may be of interest to human who is trying to understand the data.

Once data is represented as variables with a supervised prediction problem, much data analysis could be automated. For example, once data is assembled and organized, we can invoke a classification system to build a classifier. If we have a time series problem, we can use a hidden Markov model, train the model, and use it to make predictions.

1.3 Contributions

The contributions in this thesis are as follows:

1. Designed *Deep Feature Synthesis* algorithm that is capable of generating features that express a rich feature space
2. Developed an end-to-end Data Science Machine with can
   (a) posit a data science question
   (b) automatically generate features via *Deep Feature Synthesis*
   (c) autotune a machine learning pathway to extract the most out of synthesized features
3. Matched human level performance competing in data science competition using the Data Science Machine

4. Designed and implemented an interface to test new challenges in human-data interaction

Chapter 2

Overview

The Data Science Machine is a automated system for generating predictive models from raw data. It starts with data in the form of a relational database and automatically generates features to be used for predictive modeling. Most parameters of the system are optimized automatically in pursuit of good general purpose performance. Figure 2-1 shows an overview of the of the system.

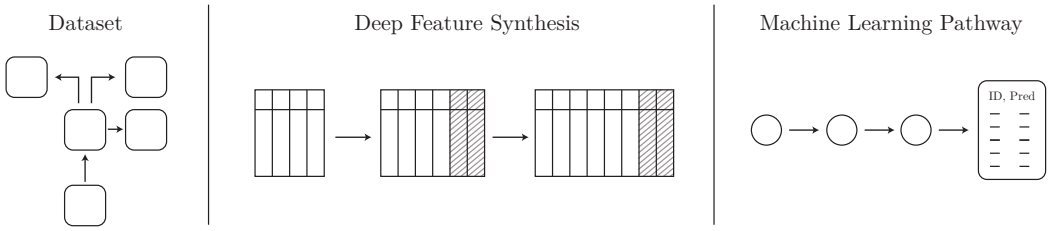


Figure 2-1: An overview of the process to automatically analyze relational data to produce predictions. The input to the system is a relational database, and the output is a predictions for a prediction problem. The prediction problem may be supplied by the user or determined automatically. The system achieves this using the *Deep Feature Synthesis* algorithm which is a capable of automatically generating features for machine learning algorithms to use. Parameters of the system are automatically optimized to achieve generalized performance across a variety of problems.

2.1 Design Goals

The Data Science Machine has ambitions in artificial intelligence, systems engineering, and human-data interaction. In this thesis, we aim to achieve all three of these ends. Approaching all three domains is necessary to successfully explore a new paradigm of data science. Without accomplishing our artificial intelligence goals we cannot design useful abstractions. If there is no effort put into a generalizable implementation, it would be difficult to iterate towards better designs of the AI system. Finally, if we do not consider how humans interact with the system, we would not have a way to fit the Data Science Machine in to the lives of data scientists. Because this research spans various disciplines of computer science, it challenged us to critically assess our priorities at each stage.

Artificial Intelligence Goal: Emulate human actions in data science endeavors We must design an approach that emulates the same expertise of human data scientists. The most important aspect of this is to develop a way to synthesize features that best describe the data. In Chapter 3, we describe the *Deep Feature Synthesis* algorithm. Then, in Chapter 5 we discuss how to employ these features to solve a particular data science problem.

System Level Goals: Automate processing An implementation serves the purpose of testing our theories in practice. By automating the process from start to finish, we can learn more, improve our design faster, and test the system on real world problems. In Chapter 4, we discuss the implementation of the *Deep Feature Synthesis*. In Chapter 6, we present an automated method for parameter tuning that enables the full pathway to work.

Human-Data Interaction: Enable human creativity We must communicate how the system functions to help users learn how to interact with it. A user can know the inputs and outputs of the system, but exposing the right parts of the inner workings of the system will enable us to tap into human intelligence. We present a step towards realizing this goal in Chapter 7.

Ultimately, the greatest challenge the Data Science Machine faces is to generalize across a variety of problems, datasets, and situations. Without touching all components of the data science process, it would fall short of this goal.

Chapter 3

Deep Feature Synthesis: Artificial Intelligence

The efficacy of a machine learning algorithm relies heavily on the input features [11]. Transforming raw data into useful features for a machine learning algorithm is the challenge known as feature engineering. Data scientists typically rely on their past experience or expert knowledge when brainstorming features. However, the time and effort it takes to implement a feature idea puts constraints on how many ideas are explored and which ideas are explored first. As a result, data scientists face the challenge of not only determining which ideas to implement first, but also when to stop trying ideas in order to focus on other parts of their job. *Deep Feature Synthesis* is the result of decomposing the features that data scientists construct into the generalized processes they follow to make them. In this Chapter, we explain the motivations for *Deep Feature Synthesis*, explain the feature generation abstractions, and present the pseudocode for running *Deep Feature Synthesis*.

3.1 Motivating *Deep Feature Synthesis*

To explain various components of our system, we make use of example data problems. In this example, we look at the dataset for the KDD Cup 2014. In Figure 3-1, we see the 9 related entities in the dataset and how they are related. The entities

are *projects*, *schools*, *teachers*, *donations*, *donors*, *resources*, *vendors*, *essays*, and *outcomes*. Projects are associated with one school, one teacher, one essay and a set of donations. Each donation is made by a specific donor. Every project has a outcome that contains details for fundraising campaigns that have completed. The goal is to predict whether or not a particular project is exciting (as specified by the Outcome entity) before knowing any information about its funding. To do this, data scientists first resort to generating features that can be used a predictors in the model

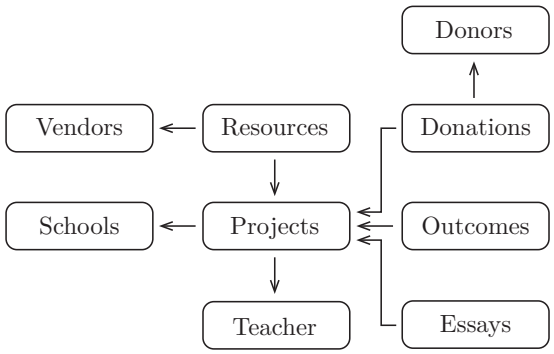


Figure 3-1: The data model for the KDD Cup 2014 problem. There are 9 entities.

If we are to think like a data scientist, we might start by asking questions that could be translated into features that describe a project. There are several questions we could ask, but a reasonable line of questioning might be: "how much money is this project requesting?", "is that amount of money more or less than average?", "do projects started in some part of the school year perform better than others?" We might also look at entities related to a project and ask questions about them. For instance, "do successful past projects by the same school or teacher make it more likely that this project will be successful?" or "does the average income of students' families at the school related to projects success?". These questions are then turned into features by following relationships, aggregating values, and calculating new features. *Deep Feature Synthesis* is a algorithm that can generate the calculations that result in these types of feature or can as proxy quantities to these ideas.

3.2 *Deep Feature Synthesis* algorithm

The input to *Deep Feature Synthesis* is a set of interconnected entities. Each entity has a primary key, which is a unique identifier for each instance of an entity that the table is based on. Optionally, an entity has a foreign key, which uniquely references an instance of a related entity. An instance of the entity has fields which fall into one the following data types: *numeric*, *categorical*, *timestamps* and *freetext*.

Notationally, for a given database, we have entities given by $E_{1...J}$, where each entity table has fields which we denote by $x_{1...|E_j|}$. $\vec{x_{ij}}$ represents the array of values for field i in entity j. Each value in this array corresponds to an instance of the entity j. $\vec{f_{ij}}$ represents an array that contains the feature values generated from values of field i in entity j.

3.2.1 Mathematical operators/functions

Our feature synthesis algorithm relies on commonly used mathematical operations and functions that used by data scientists to generate features. We formalize these types of mathematical functions below. With this formalization, we allow extensibility as more functions can be added. Given a library of these functions we will next present how to apply these functions to synthesize features and provide examples for the dataset above.

Direct: These functions do a direct translation of a field to a feature. Often times this simple translation includes operations like conversion of a categorical string datatype to a unique numeric value, which could then be used as a feature. In other cases, it is simply an exact reflection of a numerical value. Hence the input to these functions is a vectors $\vec{x_{ij}}$ and it returns $\vec{f_{ij}}$, where $\vec{f_{ij}} = d(\vec{x_{ij}})$, where d is the direct computation function.

Simple: These are predefined computation functions that translate an existing field in an entity table into another type of value. Some example computations are the translation of a time stamp into 4 distinct features - weekday (1-7), day of

the month (1-30/31), month of the year (1-12) or hour of the day (1-24). Hence this transformation is given by $\{\vec{f_{ij}^1}, \ldots \vec{f_{ij}^k}\} = s(\vec{x_{ij}})$, that is when passed of array of values for field x_{ij} it returns a set of k features for each entity. Other examples of such simple computation functions are binning a numeric value into a finite set of discrete bins.

cdf: Cumulative distribution function based features form a density function over $\vec{x_{ij}}$, and then for each entry in $\vec{x_{ij}}$, they evaluate the cumulative density value (*a. k. a. percentile*) thus forming a new feature $f_{cdf} = cdf(\vec{x_{ij}})$ (ij is dropped in the notation for f for convenience). We can also extract cumulative density in a multivariate sense given by $f_{cdf} = cdf(\vec{x_i}, \vec{x_j})$.

r-agg: Relational aggregation features (r-agg) are derived for an instance e_j of entity E_j by applying a mathematical function to $\vec{x_{il|e_j}}$ which is a collection of values for field x_i in related entity E_l, where the collection is assembled by extracting all the values for field i in entity E_l which are related to the instance e_j in Entity E_j. This transformation is given by $r - agg(\vec{x_{il|e_j}})$. Some examples of r-agg functions are *min*, *max*, and *count*.

r-dist: r-dist features are derived from the conditional distribution $P(\vec{x}_{il|e_j})$. These are usually different *moments* of the distribution. Examples include: `mean`, `median`, `std` among others.

3.2.2 Feature synthesis

Next we apply these mathematical functions at two different levels: at the entity level and at the relational level. Consider an entity E_j for which we are assembling the features. Below we present the two levels and examples.

Entity level: These are features calculated by solely considering the fields values in the table corresponding to the entity j alone. In this situation, we can apply **direct**, **simple**, and **cdf** operations on each of the fields in the table. Figure 3-2 shows the expansion of the original entity with features $\{\vec{f_1}, \ldots \vec{f_k}\}$ by applying

these operations. At this level we cannot apply **r-agg** and **r-dist** operations since we are not considering any of the relationships this entity has with other entities in the dataset.

Example 3.2.1 *For the project entity, examples of these features are day-of-the-week the project was posted or percentile relative to other projects of total students reached.*

Relational level: These features are derived by jointly analyzing a related entity E_l to the entity E_j currently under consideration. There are two possible categories of relationships between these two entities: *forward* and *backward*. An illustration of these two types of relationships is shown in Figure 3-2

Forward:A forward relationship is between an instance of entity E_j, e_j, and a single instance of another entity E_l. This is considered the forward relationship because e_j has an explicit dependence on e_l. In this case we can apply **direct** operations on the fields x_1 ... $x_{|E_l|}$ in E_l and add them as features of E_j. This is logical because an instance e_j uniquely refers to a single e_l meaning that all fields of e_l are also legal features for e_j.

Backward: The backward relation is the relationship from an instance e_j to all instances e_i that have forward relationship to e_j. In the case of backward relation, we can apply **r-agg** and **r-dist** operations. These function are appropriate in this relationship because every target entity has a collection of values associated with it in the related entity.

Example 3.2.2 *For a project, let us consider the entity school that has a backward relationship with projects. That is, teachers and schools can be associated with multiple projects. An example of a relational level feature is then the number of projects associated with each school, evaluated us* `School.COUNT(Projects)`*. Another feature could be the average number of students reached, and total number of students reached by projects for this school.*

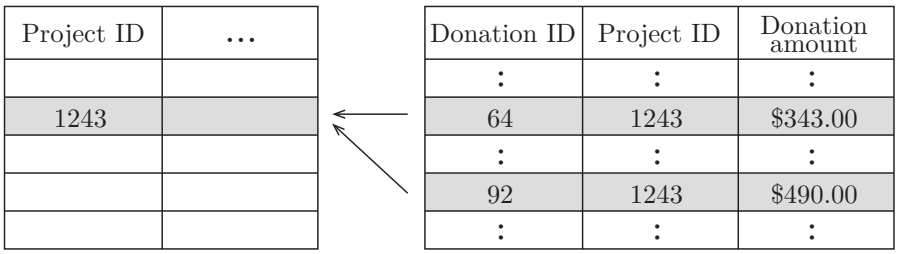


Figure 3-2: An example of *backward* and *forward* relationship. In this example, projects entity has a *backward* relationship with donations entity. There are multiple donations for the same project. While donations entity has a *forward* relationship with the projects.

3.2.3 Recursion for relational features synthesis

Now, what if before we created all these features for the *project* entity, we created features for every entity related to Projects? If features for donors and donations are made first, we would add the following feature:

Example 3.2.3 *Of the donors to this project, the average number of donations by each donor across all projects. This is evaluated as* `AVG(Donations.Donor.COUNT(DONATIONS))`.

Table 3.1 presents some features generated by recursively extracting features for the projects. We can see that computing features as above for each of the related entities first before computing the features for each project may create richer features. Generically, let us consider three related entities. That is, consider entity E_j that has a relationship with entity E_l that has a relationship with entity E_p. To define this relationship, we define a notion of depth. E_p is at depth 2 with regards to entity E_j.

Mathematical Expression	**Language Expression and intuition**
`SUM(Donations.Donor.COUNT(DONATIONS))`	The total number donations across all projects made by donors to this project
`AVG(Donations.Donor.COUNT(DONATIONS))`	Of the donors to this project, the average number of donations by each donor across all projects
`SUM(Donations.Donor.SUM(Donations.amount))`	The total amount donated across all projects by donors to this project
`AVG(Donations.Donor.SUM(Donations.amount))`	For the donors to this project, the average total amount donated across all projects
`AVG(Donations.Donor.AVG(Donations.amount))`	The average donation amount donors to this project made on average to all projects. **Intuition**: Did they donate more to this project then they do on average?

Table 3.1: Rich features extracted when recursive feature synthesis algorithm is used on the KDD Cup 2014 dataset. In this table we give the example of the computation and a simple English explanation and intuition that may actually lead the data scientist to extract the feature from the data for a particular project.

3.3 *Deep Feature Synthesis*: recursive algorithm

Next we describe the recursive algorithm to generate to generate feature vectors for a target entity. Consider that the dataset has M entities, denoted as $E_{1...M}$. Let us consider that our goal is to extract features for the entity E_i.

To generate the features for a given entity E_i, we first identify all the entities with which this entity has FORWARD relationship with and all the entities with which this entity has BACKWARD relationship with. These are denoted by sets E_F and E_B. We must then recursively call the feature generation for all entities E_B that are in backward relationship. While it is traversing through the graph of backward

relationships the algorithm keeps track of the entities it already visited in the set E_V.

Algorithm 1 Generating features for target entity

```
1: function MAKE_FEATURES(E_i, E_{1:M}, E_V)
2:     E_V = E_V ∪ E_i
3:     E_B = BACKWARD(E_i, E_{1...M})
4:     E_F = FORWARD(E_i, E_{1...M})
5:     for E_j ∈ E_B do
6:         MAKE_FEATURES(E_j, E_{1...M}, E_V)
7:         F_j =RFEAT(E_i, E_j)
8:     F_i = EFEAT(E_i)
9:     for E_j ∈ E_F do
10:        if E_j ∈ E_V then
11:            EXIT
12:        MAKE_FEATURES(E_j, E_{1...M}, E_V)
13:        F_i = F_i∪ REFLECT(E_i, E_j)
```

The algorithm *pseudocode* for MAKE_FEATURES is presented above. The algorithm stores and returns enough information to calculate the synthesized feature. This information includes not only feature values, but also *metadata* about base feature and function that were combined,

Next, we present RFEAT, which describes how relational features are generated given two entities E_i and E_j where E_i has a backward relationship with E_j. We assume that there are set of relational level functions $\mathcal{R}$ that can be applied. Some are describe in Section 4.3.1

Algorithm 2 Generating relational features

```
1: function RFEAT(E_i, E_j)
2:     F_i = {}
3:     for ∇ ∈ R do
4:         for x_j ∈ E_j do
5:             if CANAPPLY(∇, x_j) then F_i = F_i ∪ ∇(E_i, E_j)
```

To apply relational level features for entity E_i using entity E_j, *Deep Feature Synthesis* looks at all *r-agg* and *r-dist* functions that are defined in the Data Science Machine denoted by the set $\mathcal{R}$. It then determines which columns to apply these functions to. To do this it first iterates through functions $\in \mathcal{R}$ one by one. For each

function ∇, it looks at every feature in E_j to determine if the function can be applied to that feature. If the function can be applied to the feature, it is applied and the created features are added to the set of features of F_i. For example, a function to take the average of another columns is defined to only be applied to numeric columns. When this function is applied to a backward relationship between E_i and E_j, it will add the average of each numerical feature in E_j to the feature set of E_i.

If a particular entity has a second feature it can be filtered by, *Deep Feature Synthesis* will apply the function multiple times, each time filtering for one of the distinct values in the second feature.

3.4 Summary

(a) It is possible to automatically generate features that have human level interpretation or map to human intuition.

(b) We are able to generate rich features with the recursive algorithm, *Deep Feature Synthesis*.

(c) By designing the proper abstractions, the algorithm is extensible with new mathematical functions.

Chapter 4

Deep Feature Synthesis: System Engineering

Our goal when implementing *Deep Feature Synthesis* is to enable us to rapidly deploy the Data Science Machine and evaluate the features when it encounters a new dataset. As such, it is important to design a system that is agnostic to the dataset. Once we establish that the input to the system will be a relational database, we must address a variety of challenges to make it generalizable. In this chapter, we present th challenges that this brings about and how we address them.

As we argued in Chapter 2, having a system that can automatically generate features given a new relational dataset not only makes the system ready to be used by data scientists, but it also enables us iterate on our AI system because with each new dataset the system can be improved.

4.1 Platform

The Data Science Machine and accompanying *Deep Feature Synthesis* algorithm are built on top of the MySQL database using the InnoDB engine for tables. MySQL was chosen for its maturity as a database. With this choice, all raw datasets were manually converted to a MySQL schema for processing by the Data Science Machine. We implement the logic for calculating, managing, and manipulating the synthesized

features in Python.

Dataset, Entity, Feature Abstractions: We chose to use a relational database due to the natural parallels in how they store data and the requirements of *Deep Feature Synthesis*. There are a three key components in the *Deep Feature Synthesis* that must be represented in the implementation. Figure 4.1 shows how those are mapped to a relational database. A *dataset* is represented by an entire database. Within a dataset there are several entities that represent different "objects" in the dataset. Each entity is represented by a specific table in the database. Finally, each entity contains features. A single feature is represented by a single column in the database.

When we discussed the AI component of the system we spoke in terms of datasets, entities, and features. However, when we discuss the system component, it is often convenient to speak in terms of databases, tables, and columns.

The challenges for the platform are:

What platform do we build this on? The *Deep Feature Synthesis* has requirements for the type of data it must store and how it will access it. We must pick a platform that fits these requirements.

How do we represent, store, and features? The features in a dataset may vary in datatype and quantity. We need to have a general way of storing features.

How do we filter the data? There needs to be an easy way to access the data we want. Different datasets may require slicing data in different ways and our system should flexibly handle that.

How do we generate queries? We cannot have expectations for how data is labeled and organized other than that it is in a relational database. We must be able to generate queries without making assumption about naming or organization conventions.

How do we let the user help the system? Automating parts of the system may be time consuming for little practical gain. For these aspects, we should let the

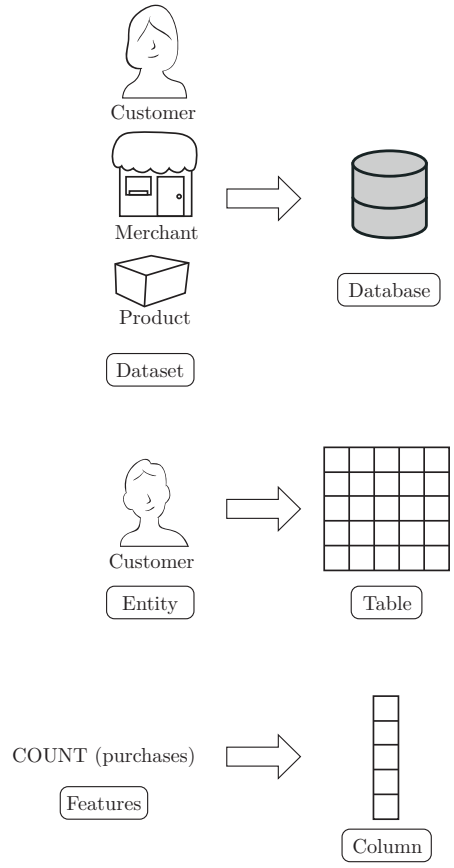


Figure 4-1: The mapping of abstractions in the Data Science Machine.

user provide information to aid the system.

4.2 Example

Let us consider an example feature for the Projects entity in the KDD Cup 2014 dataset: `AVG(Donations.Donor.SUM(Donations.amount))`. Ultimately, we must construct a database query that extracts this value. In this example, the final feature is actually constructed using synthesized features. To take advantage of this, our approach is to calculate and store those intermediate features rather than creating a monolithic query. This allows reuse of intermediate values and simplifies the implementation.

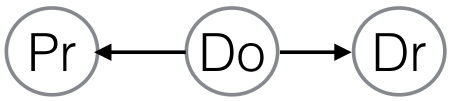


Figure 4-2: An illustration of relationship between the three entities in KDD cup 2014: Projects (Pr), Donations(Do) and Donors (Dr). Projects have a backward relationship with donations (since a project may have multiple donations), a donor has a backward relationship with donations (since a donor can be associated with with multiple donations).

Figure 4.2 shows the relationship between the three entities under consideration. The process starts with following entity relationships as described in Section 3.3. Starting with projects it recurses to the donations entity and since donations by itself does not have any *backward* relationships, it considers its *forward* relationship with donors.

Once we reach the Donor entity, we consider its *backward* relationship with donations and begin to apply the RFEAT algorithm presented in Section 3.3. The algorithm iterates over Feature Functions which each specify a calculation and rules about when it should be applied. At this level, when the algorithm considers `SUM`, the *SQL* query to generate features for the donors is created automatically as shown below:

```
UPDATE `Donors_1` target_table
LEFT JOIN ( SELECT `donor_acctid`, SUM(`amount`) as val
            FROM Donations rt
            GROUP BY `donor_acctid`
            ) b
ON `b`.`donor_acctid` = `target_table`.`donor_acctid`
SET `target_table`.`Donors_1__100` = `b`.val
WHERE `b`.`donor_acctid` = `target_table`.`donor_acctid`
```

After calculating the feature, we store the feature values as a new column in the Donors table, as well as store the metadata to use later in the process. The metadata contains information such as the base column (`Donations.amount`) and the function applied (`SUM`). At this stage we have synthesized the feature `SUM(Donations.amount)`,

considering all the donations that this donor has in the donations table.

With a new feature synthesized for each donor in the dataset[1], we build features for each donation. Because each donation uniquely references a single donor, we just reflect the feature we just calculated to the Donation table. When we reflect this feature we do not actually copy the value. Instead, using the same metadata object mentioned above, we notate where the true feature values are actually stored. In this case, it is in the Donors table. We now have the feature `Donor.SUM(Donations.amount)` for every donation.

The final step is to generate features for the original entity, Projects. Now, considering its *backward* relationship with donations, we begin to apply the `AVG` Feature Function (via the RFEAT algorithm). This is similar to applying the Feature Function before except that the feature we want to aggregate isn't actually stored in the entity it is associated with. However, using the information in the *metadata* object we know where in the database a feature is actually stored. With this information we can create a subquery that joins the appropriate tables together to make the feature columns available. After the join, we can construct the query as we did before,

If at any point, we only want a subset of of data, we use Filter Objects. For example, if we wish to calculate the same feature above, but only for Donors that live in Massachuseets, we could create a Filter Object with the triple `("state", "=", "MA")`. This Filter Object is then serialized and used in the query.

4.3 Features: functions, storage, naming and metadata

4.3.1 Feature Functions

The functions described in 3.2.1 are implemented as Python classes, and they all provide a Feature Function interface. A Feature Function takes one or two entities as inputs depending on the type of feature. Two entities are required in the case of

[1] and assuming we only have `SUM` in our function set

relational level features, and direct features where the source is another entity. The function definition is responsible for determining whether or not it can be applied to a certain columns using the column's metadata (described below 4.3.4) and how to calculate the output value. For example, some Feature Functions specify they should only be applied to date fields. Because features are often calculated using synthesised features, Feature Functions may also specify rules that govern what mathematical functions can be applied to synthesized features. For example, a rule limits the standard deviation Feature Function from being appled to a feature synthesized using the standard deviation Feature Function.

For performance reasons, the Data Science Machine builds Feature Functions on top of the functions provided by MySQL. Currently, the Data Science Machine implements the following relational level functions: `AVG()`, `MAX()`, `MIN()`, `SUM()`, `STD()`, and `COUNT()`. In addition to those functions, the system implements the following simple functions: `length()` function to calculate the number of character in a text field, `WEEKDAY()` to convert dates to the day of the week they occurred, and `MONTH()` to do the same for the month. Despite their simplicity, this small base of functions is enough to create a wide range of features for us to evaluate the Data Science Machine.

4.3.2 Feature storage

All features are stored as columns in the database. As the Data Science Machine creates new features, it uses new columns to store the data. Successively altering table columns on a large MySQL table using the `ALTER TABLE` command is expensive because MySQL must create a temporary copy of the original table while also locking the original [2]. Because of this overhead in creating the temporary table each time a new column is created, it is infact cheaper to create multiple columns simultaneously. Thus, while creating the first new column for a table, the Data Science Machine actually creates multiple new columns. To manage these new columns, the Data Science Machine implements a *Column Broker*. By default the Column Broker creates 500 new columns at a time and specifies the data types for 80% of them as floats and the rest 20% as integers. These settings are configurable using the configuration file

mentioned later in Section 4.6.

The Column Broker maintains a list of the free columns of each type. Using the Column Broker, *Deep Feature Synthesis* then can request a column of a feature's datatype on demand without the latency of issuing the `ALTER TABLE` command. Without this optimization, creating hundreds of new features would take a prohibitive amount of time.

If the Column Broker does not have a column of the requested datatype, then it will create a copy of the original table that has the same fields as the original table and initialize multiple new columns again. Even though these columns are in a separate table, by copying the original columns, we maintain the primary key information for each row. Using the primary key we can join the original table with its copies when we need access to specific features. MySQL limits InnoDB tables to have 1000 column max. Thus, without this implementation the Data Science Machine would be limited in the number of features it could synthesize for a given entity.

4.3.3 Naming features

A challenge faced by the Data Science Machine is displaying synthesized features in a human understandable way. In part, this is because simply describing a calculation does not necessarily resemble the intuitive description a human would give. The solution is to use a standardized mathematical notation. Table 3.1 shows some examples of features names and their natural language descriptions. The relationship between the name of the feature and English description is often not immediately clear.

A feature name is constructed by surrounding another feature name with the function that was applied to it. If the function is omitted, it is assumed to have been a direct function that reflected the value of the feature in a forward relationship. For example, Table 3.1 shows examples of this where Donor reflects a feature from Donations. Because the notation definition for a feature name is defined by another feature name, we get the recursive effect of functions inside functions.

4.3.4 Metadata

An important aspect to being able to use the features created by *Deep Feature Synthesis* is to understand how it came about. To accomplish this, every feature contains a metadata object.

The metadata object has the following fields:

Field	**Purpose**
entity	the entity this feature is for
real_name	a name that encodes what the feature is. eg School.MAX(Projects.SUM(Dontations.total)) which means largest amount raised by a project at this school)
path	list of path nodes that describe how this feature was calculated

A path is a list of nodes which are defined as follows:

Field	**Purpose**
Base column	the column this feature was derived from and its metadata
feature_type	the type of feature this is, eg direct, r-agg, etc
feature_type_func	the name of the function that was applied
filter	the Filter Object used by the query that generates it.

4.4 Filtering data

The Data Science Machine needs a flexible way to select subsets of data. Filter Objects play an important role for relational level Feature Functions by enabling calculations over subsets of instances. They are implemented to enable construction of new Filter Objects by combining Filter Objects using logical operators such as “and” or “or”. Using Filter Objects, a Feature Function is able to select which instances of an entity to include in a calculation.

Filter Objects implement two methods for the query generation process described in 4.5.

1. `to_where_statement()`. This converts a Filter Object to a MySQl where statement to be included in a query. For example, `FilterObject([("state", "=", "MA"), ("age", ">=", 18)])` serializes to `WHERE ‘state‘ = MA AND ‘age‘ > 18`.

2. `get_involved_columns()`. This returns all the columns that this filter will be applied to. This is used by the query generation algorithm to ensure they are selected before applying the `WHERE` statement.

Filter Objects are used to implement two useful pieces of functionality for *Deep Feature Synthesis*: conditional distributions and time intervals.

Conditional distributions Filter Objects can be used to condition a Feature Function. This is important because there are many scenarios where the conditional distribution exposes important information in a dataset. A guiding example is making features for customers of an ecommerce website. Without Filter Objects, *Deep Feature Synthesis* could create a feature for each customer that is the total amount of money that customer spent on the site by summing up all the purchases the customer made. However, this might not differentiate customers that spent the same amount of money, but in different ways. Instead, we might want to create features for the total amount spent in each of the product categories such as electronics or apparel. This is accomplished by using different Filter Objects that each select only the purchases from a single category before applying the Feature Function.

Time intervals For some prediction problems, we want to calculate the same set of features across many time intervals. This allows use to model time series data. A time interval Filter Object is implemented as a logical combination of two Filter Objects. One Filter Object selects instances that occur after a certain time and one Filter Object selects instances that occur before a certain time.

When these two functions are combined using the `AND` operator, they specify a single time interval.

When creating intervals there are 4 important parameters.

1. The column that stores the time interval
2. The time to start creating time intervals
3. The length of the time interval
4. The number of intervals.

When supplied with these four parameters, *Deep Feature Synthesis* is capable of creating time interval-based features for a target entity. *Deep Feature Synthesis* maintains metadata about the creation of each feature that can be used by a learning algorithm to determine the features associated with each time interval.

4.5 Generating queries

In order to extract features, the Data Science Machine constructs MySQL database queries on the fly. As the data required to calculate a feature may be stored in multiple tables, a process for joining tables together is necessary. All columns in the `SELECT` clause or `WHERE` clause of a query must be accessible for a query to work.

MySQL supports Views, which are virtual tables composed of columns from other tables or views. While these would simplify the design of the query generation, in the Data Science Machine design we choose to manually implement the logic to join tables together. We did this for two reasons. First, the workload of the *Deep Feature Synthesis* iteratively adds columns to tables. The changing structure of tables would mean redefining a new view each time a new column is added as MySQL views are frozen at the time of creation[3]. Second, while we may join the same tables together to make different features, we often will be selecting different columns from the joined result. To use a view for this would lead to unnecessary columns at times in our select, resulting in a potential performance hit.

To improve performance further, a query is constructed to perform multiple calculations at once. This works by cutting down on the number of times the query builder performs joins, as well reducing the number of times *Deep Feature Synthesis* has to iterate over every row in a table. A query is constructed in the following three steps:

Collect Columns The first step is determining all columns that are involved in the query. Columns may be involved in a query because they are part of the actual calculation or because the query contains a Filter Object.

Determine join path The actual source of a column may be in different table than the target of the calculation, but the column metadata stores the path to the column. Using this path data we can construct a *join path*, which is a series of MySQL `JOIN` statements that assemble a subquery for the query to execute on. If we collected multiple columns for the query, we can have many different paths, some of which share parts. The join path is constructed by first sorting the involved columns by path length. Then, starting with the longest path, we join tables in the path beginning with the table we wish to add the feature to. We proceed column by column in this sorted order. For each new column we only add tables from its path if they branch from the path that has already been constructed. By iterating over the columns according to path length, we ensure the paths with the most constraints are handled first.

Build Query With the subquery to build the joined table, the final step is to put the desired functions of the columns in a `SELECT` statement, turn the Filter Object into a `WHERE` clause using the defined API, and group by the primary key of the target entity. The `GROUP BY` is what collects the rows for relational level functions.

4.6 User configuration

The Data Science Machine takes a configuration file for specifying dataset specific options. The configuration file presents options at the tree levels of abstractions in the data science machine: dataset, entity, and feature. The Data Science Machine will infer some of these parameters to the best of its ability or revert to a sensible default value if inference is difficult or computational expensive.

4.6.1 Dataset level

Option	**Purpose**
max_depth	the maximum recursion depth of the deep feature synthesis algorithm
max_categorical_filter	the maximum number of categorical filters to apply along a feature pathway

4.6.2 Entity level

Every entity in the dataset has a copy of the following parameters

Option	**Purpose**
one_to_one	list of entity names that are in a one-to-one relationship with this entity. Since the one-to-one relationship goes both ways, a given pair only needs to be notated in at least one entity
included_functions	white list of Feature Functions to apply to an entity
excluded_functions	black list of Feature Functions to apply to an entity
train_filter	parameters for a Filter Object that specifies which instances of this entity may be used for training

One-to-one relationship are inferred by counting the number of backward relationship for every entity in a table. Backward relationships are defined in section 3.2.2.

If the number of backward relationships is one for every instance then it is considered to be part of a one-to-one relationship. This calculation is expensive in the current system, so it is cached and only performed once for each relationship. Additionally, it is also not performed on entities that contain more the t rows, where t is currently set to 10 million.

If not specified, included_functions is assumed to be every function available. Therefore included_functions and excluded_functions can be used as a white list and black list for Feature Functions. Typically this is used to avoid performing expensive computations that the user expects to not contribute to successful feature generation.

4.6.3 Feature level

Every feature in the dataset has a copy of the following parameters

Option	**Purpose**
categorical	boolean indicating whether or not this variable is categorical
categorical_filter	boolean indicating whether or not to use distinct values of this feature as Filter Object
numeric	boolean indicating whether or not this variable is numeric
ignore	boolean indicating whether or not to ignore a feature. this is used when the metadata is necessary for some other computation and the column isn't actually a feature such as label test or train data.

Categorical can often be inferred. Right now, the Data Science Machine assumes columns that have exactly two distinct values to be categorical. The Data Science Machine avoids being too liberal in calling features categorical because they lead to an growth in the feature space during the machine learning pipeline. For example, the zip code of a customer is categorical, but could create thousands of new features

during One Hot Encoding (explained in Section 5.3).

Numeric is assumed if the database column type is specified as a numeric type. This includes integers, floats, doubles, decimals, smallints, and mediumint in MySQL.

4.7 Summary

(a) We implemented *Deep Feature Synthesis* using MySQL and Python.

(b) The implementation make no assumptions about the input beyond it is a relational database.

(c) We allow for humans to guide the system as needed with a configuration file on a per dataset basis.

Chapter 5

Predictive Machine Learning Pipeline

To use the features created by *Deep Feature Synthesis*, we must implement a generalized machine learning path. For this, we have at our disposal a representation of the data as tuples given by $< e, fn, fv, t >$ where e is entity, fn is the feature name, fv is the feature value, and t is the time interval id. To build a predictive modeling framework we next design three components: an automatic prediction problem formulator, a feature assembly framework, and a machine learning framework. We use the open source scikit-learn [15] to aid the development of this pathway. In the next two subsections we describe these three components.

5.1 Defining a prediction problem

The first step is to formulate a prediction problem. This means selecting one of the features in the dataset to model. The feature could be one that already exists in the dataset or one that was generated while running *Deep Feature Synthesis*. Additionally, the feature may be a continuous value such as the the total amount of money donated by a donors or a categorical value such as whether or not a project was fully funded. We call this feature we wish to predict the target value.

For some cases, the target value is known before the analysis begin, in which case this information can be supplied to the Data Science Machine. This is the case for

the competitions we test the system on.

In other cases, when the target value is not externally provided, Data Science Machine can run the machine learning pipeline for several potential target value and suggest high performing results. To prioritize which ones to pick as target value and show the results to the user, Data Science Machine applies some basic heuristics. At this time, these heuristics are

- Preference for variables that are not derived *via* feature synthesis algorithm
- Categorical variables with a small number of distinct categories
- Numeric features with high variance

5.2 Feature assembly framework

After a target value is selected, we next assemble the features that are appropriate for use in prediction. We call these features *predictors*. For a given target value, some predictors might be invalid because they were calculated with the common base data or have an invalid time relationship. For instance, if the target value to predict is the average donation amount made by a donor, it would be invalid to predict that using the total amount the donor donated and the number of donations they made. A feature could invalid temporally, if it relies on the data that did not exist at the time of the target value. For example, if we want to predict how many donations a project will receive in it's first week, it is invalid to predict that using a feature that is the total sum of the donations for that project.

To accomplish the task of assembling features for a valid prediction problem, the Data Science Machine maintains a database of *metadata* associated with each entity-feature. This *metadata* contains information about the source fields in the original database that were used to form this feature, as well as any time dependencies contained with in it. Using this *metadata*, the Data Science Machine assembles a list of all features that were used to calculate the target feature. The Data Science Machine makes this same list for all candidate predicting features. A feature is only usable

if these lists contain no overlaps. To handle time dependencies, the Data Science Machine excludes any feature that is assigned an interval number higher than the target feature.

5.3 Reusable machine learning pathways

Table 5.1: Summary of parameters in the machine learning pipeline. These parameters are tuned in automatically by the Data Science Machine.

Param	**Default**	**Range**	**Function**
k	1	[1, 6]	The number of clusters to make
n_c	100	[10, 500]	The number of SVD dimensions
γ	100	[10, 100]	The percentage of top feature selected
rr	1	[1, 10]	The ratio to re-weight underrepresented classes in classification problems
n	200	[50, 500]	The number of decision trees to create when training a random forest
m_d	None	[1, 20]	The maximum depth of the decision trees
β	50	[1, 100]	The maximum percentage of features that are used in decision trees

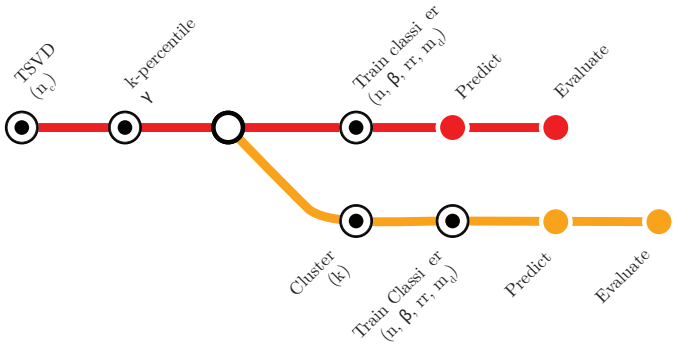


Figure 5-1: Reusable parameterized machine learning pipeline. The pipeline first performs truncated SVD (TSVD) and selects top γ percent of the features. Then for modeling there are currently two paths. In the first a random forest classifier is built and in the second one the training data is first clustered into k clusters and then a random forest classifiers is built for every clusters. The list of parameters for this pipeline are presented in Table 5.1. In the next section we present an automatic tuning method to tune the parameters to maximize cross validation accuracy.

With a target feature and predictors selected, the Data Science Machine implements a parametrized pathway for data preprocessing, feature selection, dimension-

ality reduction, modeling, and evaluation. To tune the paramters, he Data Science Machine provides a tool for performing intelligent parameter optimization. The following steps are followed for machine learning and building predictive models:

Data preprocessing

The importance of the data preprocessing step is to normalize data before continuing along the pathway. To preprocess the data, we assemble a matrix using the tuples $< e, fn, fv, t >$ for the target fn and each fn that was chosen to be a predictor. In this matrix, each row corresponds to an instance of E and each column contains the values fv for the corresponding fn.

There are several steps normalize the data

- **Null values**: We must convert null values. If a feature is a continuous value, the null value is converted to the median value of the feature. If the feature is categorical, the null value is converted to an unknown class.

- **Categorical variables:** Categorical variables are sometimes represented in this matrix as string. So, first we must convert each distinct string to a distinct integer. Next, we use One Hot Encoding to convert the categorical variable into several binary variables. If there are n classes for the categorical variable, we create n binary features in which only one is positive for each instance.

- **Feature scaling:** Different features are often at different scales depending on their unit. To prevent certain features from dominating a model, each feature column is scaled to have a mean of 0 and unit variance.

Feature selection and dimensionality reduction

Deep Feature Synthesis generates a large number of features per entity. Even after removing the features that may be invalid for predicting the target value, we are still left with a large number of predictors that we can use for predicting the target value. Many of these features may not have any predictive value, or even if they have little predictive value, could diminish the value of the good predictors because they increase the dimensionality. To overcome this, it is important to have a process of selecting the

useful features before learning the final model. The Data Science Machine employs two techniques for reducing the size of the feature space.

Truncated SVD transformation In the first method, the Data Science Machine applies Singular value decompositions to the feature matrix. Then the dimensionality of the problem is reduced by selecting n_c components of the SVD. This introduces a parameter n_c in the pathway.

k-Best: In the second method, the Data Science Machine ranks each feature by calculating its *f-value w.r.t* to the target value. The f-value is the ratio of two mean square values. If the null hypothesis is true, you expect F to have a value close to 1.0 most of the time. A large f-value means that the variation among group means is more than you'd expect to see by chance. It then selects the $\gamma\%$ best features. This introduces a parameter γ in the pathway.

Modeling

The Data Science Machine models support both continuous (regression) and categorical features (classification) as a target value. In both cases, the system uses random forests to model the relationship between the predictors and the target feature. A random forest is created by using n random decision trees. Each decision tree has a depth of m_d and uses a fraction of the features denoted by β .

In classification problems, sometimes one of the target value classes is underrepresented. In order to compensate for this, the modeling stage can re-weight an underrepresented class by a factor of rr.

Hence, at the modeling stage we have introduced four parameters: n, m_d, β, rr.

Clusterwise modeling

In many datasets, it can be powerful to have a separate model for different data points. To incorporate this concept into the Data Science Machine, the system builds a KMeans classifier to separate training points into k clusters. Then, the Data Science Machine trains a distinct random forest for each cluster. When the Data Science

Machine predicts a label for a test sample, the trained cluster classifier assigns the label to a cluster and then applies the corresponding model.

At this stage, a final parameter k, the number of clusters, is introduced. If $k = 1$ then the the pathway functions as if no cluster-wise modeling was applied.

5.4 Summary

(a) We extract value from synthesised features using a machine learning pathway

(b) It is important to expose as many parameters as possible in the pathway in order to make it generalizable to many datasets.

(c) It is possible to formulate predictive problems automatically

Chapter 6

Parameter Tuning

Many stages of the machine learning pipeline have parameters that could be tuned. Tuning these parameters has a noticeable impact on the model performance. A naive grid search of the space of parameters is not feasible. If we consider all possible combinations of parameters values, there are approximately $6 * 490 * 90 * 10 * 450 * 20 * 100 = 2,381,400,000,000$ (two trillion, three hundred eighty-one billion, four hundred million) possibilities that the Data Science Machine could use for the pathway described in Chapter 5. To aid in the exploration of this space, we use a Gaussian Copula Process to model the relationship between parameter choices and the performance of the whole pathway.

6.1 Bayesian parameter optimization using Gaussian Copula Processes

To aid in the exploration of the parameter space, we used a Gaussian Copula Process (GCP) to model the relationship between parameter choices and the performance of the whole pathway and then used the model to identify better choices. Previously, a Gaussian Processes has been used to tune parameters of classifiers whose values can have huge impacts on the final performances of the model. [18] proposed to abstract this as the Bayesian optimization of a "black box" function. Our tuning system is

much broader then simply tuning a classifier parameters (or hyperparameters) and we expect non-linear relations and correlations within the parameters themselves and between parameters and the performance. Below we describe the *Model-sample-optimize* approach.

6.1.1 Model-Sample-Optimize

The first step in finding the right parameters is to model a function f that captures the relationship between the parameters and the model's performance. Then we sample new parameters and predict what their performance would be using the model's performance. We then apply selection strategies to choose what next to sample. Below we describe each of these steps in detail.

Model: The function f models the performances of the end-to-end Data Science Machine as a function of the parameters that are chosen. The basic idea of Gaussian processes is to model f such that for any finite set of N points $\bar{p}_{1...n}$ in $\mathcal{X}$, $\{ f(\bar{p}_i) \}_{i=1}^{N}$ has a multivariate Gaussian distribution on R^N. The properties of such a process is determined by the mean function (commonly taken as zero for centered data) and the covariance function $K : P \times P \rightarrow R$. For an extensive review of Gaussian Processes, see Rasmussen and Williams [17], while their use in Bayesian optimization is largely explained in [7].

In this thesis, we introduce a novel approach for parameter optimization based on Copula processes. In particular, we focus on Gaussian Copula Processes (GCP), as defined by Wilson et al. in [21] through *warping* the output space.

Gaussian Copula Processes: In Gaussian Copula Process, as introduced in [21], instead of a Gaussian Process to model the multivariate distribution of $\{ f(\bar{p}_i) \}_{i=1}^{N}$, it is done through a mapping $\Psi : R \rightarrow R$ that transforms $\{ f(\bar{p}_i) \}_{i=1}^{N}$ in $\{ \Psi \circ f(\bar{p}_i) \}_{i=1}^{N}$ which is then modeled as a Gaussian Process. By doing this, we actually change the assumed Gaussian marginal distribution of each $f(\bar{p})$ into a more complex one.

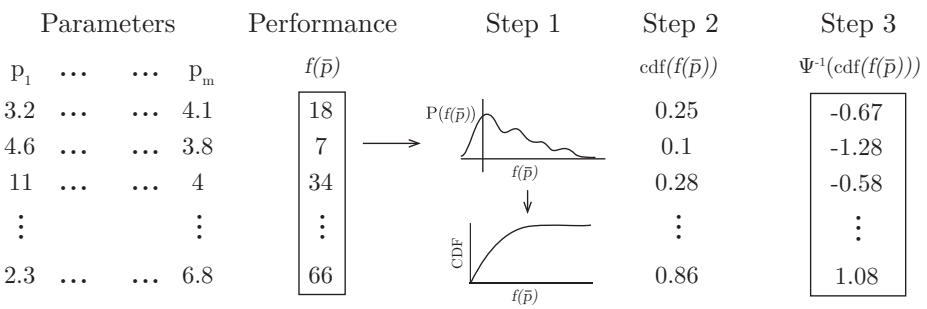


Figure 6-1: Illustration of transformation of the output value $f(\bar{p})$. The transformed value is then modeled by the regular Copula process.

The mapping function: In [21], a parametrized mapping is learned so that the transformed output is best modelled by a Gaussian Process. However, we have found that such a mapping was unstable : for many experiments on a same dataset, different mappings where learned. Moreover, the induced univariate distribution was most of the time almost Gaussian and the parametrized mapping could not offer a greater flexibility.

To overcome this, we introduce a novel approach where a marginal distribution is learned from the observed data through kernel density estimation [1]. More specifically, consider the parameters $\bar{p} = \{p_1 \dots p_m\}$ and the performance function be $f(\bar{p})$. Our first step in the transformation models the density of $\{\ f(\bar{p}_i)\ \}_{i=1}^{N}$ using a kernel density estimator and then it estimates the *cdf* of this density. We then generate the *cdf* values for each value of $\{\ f(\bar{p}_i)\ \}_{i=1}^{N}$ shown in Step 2 of Figure 6.1.1 and are given by $g = cdf(f(\bar{p}))$. Assuming that g is a sample from a standard normal we apply ψ^{-1} to values in g to generate the final values shown in Step 3 of Figure 6.1.1 and given by $h = \psi^{-1}(g)$. h represents the transformation of $f(.)$ which we wish to model using a regular Gaussian process. Hence the input to the Gaussian process modeling is $\bar{p}_{1\dots n}$ and the corresponding $h_{1\dots n}$ values.

[1] At the time of writing of this thesis, this method was currently under development by Sebastien Dubois and Kalyan Veeramachaneni. Author acknowledges Sebastien Dubois for sharing this method, and also providing writeup and help in integrating this tool into the Data Science Machine workflow.

Fitting the covariance function The covariance function used is :

$$K(\bar{p}, \bar{p}') = \theta_0.K_0(\bar{p}, \bar{p}') + \theta_1.K_1(\bar{p}, \bar{p}') + \theta_2.K_2(\bar{p}, \bar{p}') \tag{6.1}$$

with :

$$\bar{p} = [p_1, ..., p_m]^T,$$

and :

$$K_0(\bar{p}, \bar{p}') = exp(-\sum_{i=1}^{m} \theta_{3,i}.(p_i - p_i')^2) \tag{6.2}$$

$$K_1(\bar{p}, \bar{p}') = exp(-\sum_{i=1}^{m} \frac{(p_i - p_i')^2}{2.\theta_{4,i}} - \sum_{i=1}^{m} 2\Big(\frac{\sin(\pi(p_i - p_i'))}{\theta_{5,i}}\Big)^2) \tag{6.3}$$

$$K_2(\bar{p}, \bar{p}') = \prod_{i=1}^{m} \Big(1 + \frac{(p_i - p_i')^2}{\theta_{7,i}}\Big)^{-\theta_{6,i}} \tag{6.4}$$

Sample The practical method to find the maximum point $\bar{p}^*$ is to sample iteratively some points in $\mathcal{P}$, compute their corresponding value $f(\bar{p})$, and then decide which point to sample next.

Optimize This final step is usually made by maximizing the *acquisition function* a. The inputs to this function are derived from f to balance between exploration - testing points in unexplored area of the input space $\mathcal{P}$ - and exploitation - testing points for which we predict a high $f(\bar{p})$ value. In particular this enable us to avoid concentrating on the search near local optima. Given a set of observations $(p_{1...n}, f(p_n))$, we can thus randomly sample $\bar{p}_i'$ in $\mathcal{P}$, predict their output f_i' and choose the $\bar{p}'^*$ that maximizes a.

Chapter 7

Human-Data Interaction

The Data Science Machine investigates the potential for automatic feature creation in data science. Previously, we described how we can automatically harness the power of these features using the machine learning pathway (Chapter 5) and auto tuning (Chapter 6). In this form, the system is very much a black box. To utilize the power of true human intelligence, it is worth exploring how we can expose the inner workings of the Data Science Machine.

The main challenges stem from the large number of features produced by *Deep Feature Synthesis*. For many datasets, the Data Science Machine creates hundreds of features. This large set of feature causes issues for users who must localize themselves in the feature space before interacting with the features. Even more, the modeling pathway contains numerous parameters that have complex relationships with each other. These issues pose interesting challenges when designing an interface to allow human interaction with the Data Science Machine. With these challenges in mind, we built the interface with the following goals:

Explain and organize features We must inform users of all the features at their disposal in an accessible manner. Even more, the interface should provide functionality for a user to discover the features they want to use through recommendations and filtering

Expose pathway parameters We should allow users to manually tweak modeling

parameters while still exposing the power of feature optimization in the Data Science Machine.

Enable human creativity One of the greatest human strengths is to understand problems in creative ways that escape machine intelligence. The system should give users a way to experiment with ideas and use those results in a positive feedback loop.

7.1 Features

To explain and organize features, the interface tries to slowly introduce the users to the feature space, while visually demonstrating that it is synthesising features on a per entity basis.

To help users get a sense of the features they have to work with, when they begin using the tool, an animated dialog appears that summarizes the results of *Deep Feature Synthesis*. This dialog is animated to appear as if all features are being generated at that moment. This dialog is pictured in Figure 7-1.

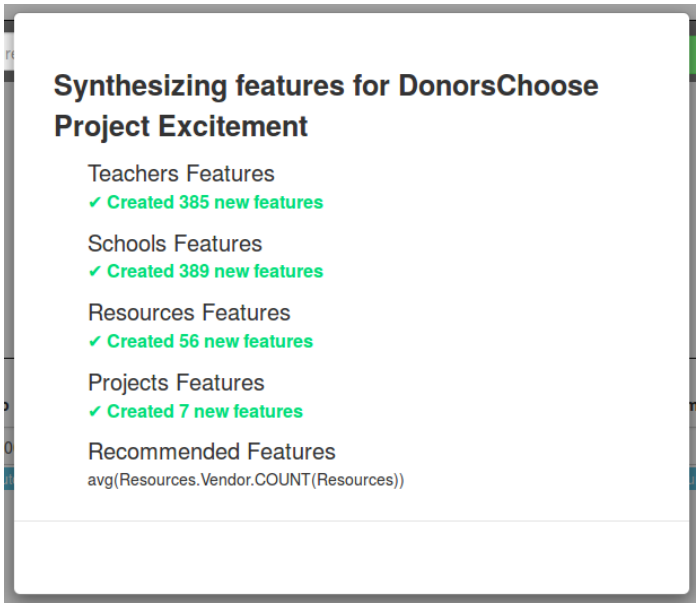


Figure 7-1: The dialog users first see when they begin using the Data Science Machine.

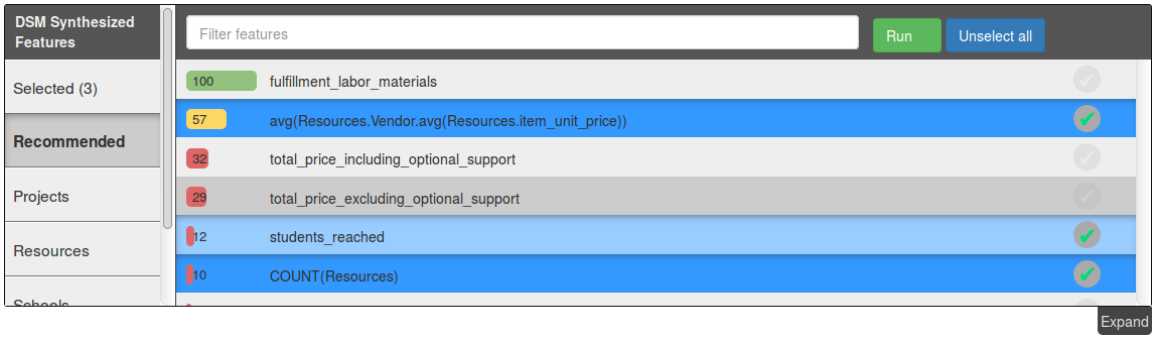


Figure 7-2: The feature selector component of the interface where users can choose features to build a predictive model with.

After clicking "continue" in the dialog, the user is introduced to the feature selector as seen in Figure 7-2. The goal of the feature selector is to provide the user with a quick way to discover and select the features to use in a prediction problem. The feature selector organizes features by entity, so users can ize themselves in the feature space as they navigate features. To help with the discovery of useful features, all features are given a score based on their estimated predication quality. Currently, this estimate is based on how well just that single feature predicts the target label. Using this score, the feature selector creates a "Recommended" features tab that sorts all features by score. If a user has an idea of a feature he or she wants to use, there is a textbox to filter the feature list in each tab. Additionally, selection of multiple features is simplified with a drag to select mechanism. At any time, a user can reset by clicking "Unselect All".

7.2 Pathway parameters

Figure 7-3: Interface for users to modify pathway parameters

The interface exposes all pathway parameters described in Table 5.1. Each parameter is initialized with the default value. From here, the user can manually change each parameter. Alternatively, a user can click "auto tune" and the parameter will be optimized by the same tuning process as the fully automated Data Science Machine. If a user does not want to modify parameters, this component of the interface is designed to not get in the way.

7.3 Experiments

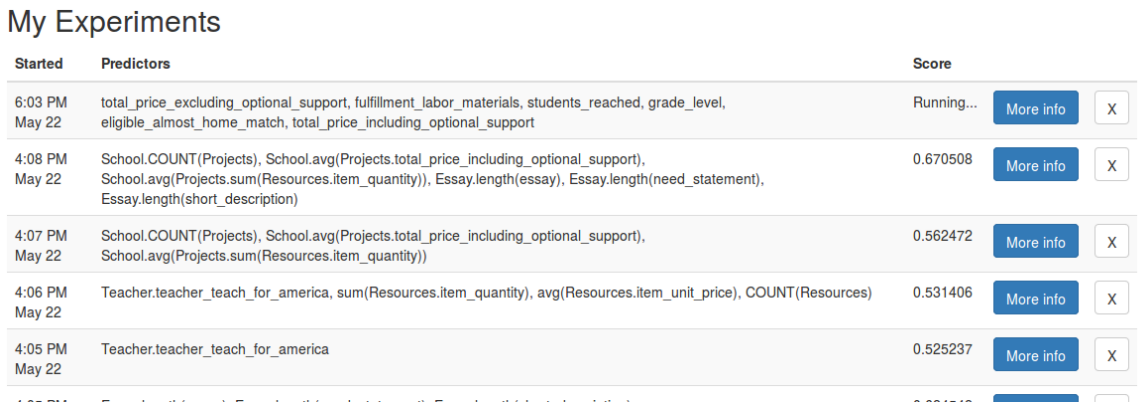


Figure 7-4: Interface for user to view experiments. Experiments are started in the background and the score is populated when they finish.

The interface for experiments is meant to encourage designing experiments, evaluating the results, and iterating on new insights. To this end, the experiments section lays out all experiments the user is currently running and has ran in the past. When a user starts a new experiment, the Data Science Machine server asynchronously runs the experiment. This means that a user can submit multiple experiments at a time. When an experiment finishes, the user is notified via a unobtrusive dialog. Each experiment returns with the a cross validated evaluation score depending on the settings for the problem. To facilitate the process of incorporating the results of the best experiments, a user can click on a past experiment and repopulate the feature selector and tuning parameters with the settings from that experiment.

Chapter 8

Experimental Results

With the Data Science Machine being first of its kind, we wish to address a number of questions that are at the intersection of "how well did the machine do?", "did it produce any interpretable features?", and "did automation work?". In the following sections we answer these questions systematically.

8.1 Datasets

In order to demonstrate the efficacy of the Data Science Machine, we will test the Data Science Machine on 3 contemporary prediction problems posed by competitive venues - KDD Cup 2014, IJCAI (ongoing), and KDD Cup 2015 (ongoing). In each one of these competitions, hundreds of data science enthusiasts participate in modeling the prediction problem defined by the organizers. To compete, data scientists engage in *feature engineering*, *modeling*, and *tuning*. All three activities can be performed by the Data Science Machine.

The diagrams in Figure 8-1 show the entities for each dataset. Table 8.1 summarizes the number of rows for each entity, and below we describe each of the problems briefly.

KDD Cup 2014: Project Excitement Using past projects' histories on DonorsChoose.org, predict if a crowd-funded project is "exciting". This is the example used throughout this thesis.

IJCAI: Repeat Buyer Prediction Using past merchant and customer shopping data, predict if a customer making a purchase with a promotion will turn into a repeat buyer.

KDD Cup 2015: Student Dropout Using student interaction with resources on an online course, predict if they will dropout in next 10 days.

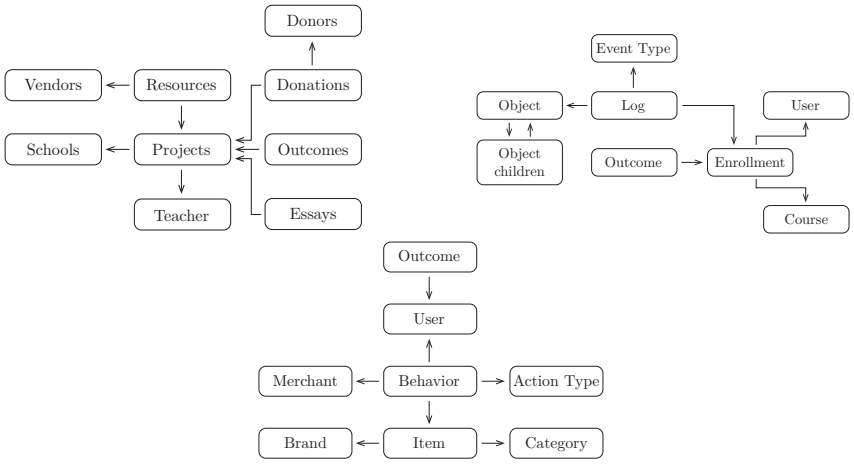


Figure 8-1: Three different data science competitions held during the period of 2014-2015. On the left is the data model for KDD Cup 2014, at the bottom center is the data model for IJCAI, and on the right is the data model of KDD Cup 2015. A total of 906 teams took part in these competitions. We note that two out of three competitions are ongoing and predictions from the Data Science Machine solution were submitted and the results are reported in this paper.

KDD Cup 2014		IJCAI		KDD Cup 2015	
Entity	# Rows	Entity	# Rows	Entity	# Rows
Projects	664,098	Merchants	4995	Enrollments	2,000,904
Schools	57,004	Users	424,170	Users	112,448
Teachers	249,555	Behaviors	54,925,330	Courses	39
Donations	2,716,319	Categories	1,658	Outcomes	120,542
Donors	1,282,092	Items	1,090,390	Log	13,545,124
Resources	3,667,217	Brands	8,443	Objects	26,750
Vendors	357	ActionType	5	ObjectChildren	26,033
Outcomes	619,326	Outcomes	522,341	EventTypes	7
Essays	664,098				

Table 8.1: The number of rows per entity in each dataset. The uncompressed sizes of the KDD Cup 2014, IJCAI, and KDD Cup 2015 are approximately 3.1 GB, 1.9 GB, and 1.0 GB, respectively.

8.2 *Deep Feature Synthesis*

We ran *Deep Feature Synthesis* on all three datasets. For KDD Cup 2014, we also constructed time interval features, and for IJCAI and KDD Cup 2015 we created a few categorical filters. These decisions were not made on the expected effectiveness, but on the constraints of dataset size. The results of running *Deep Feature Synthesis* are presented in Table 8.2.

8.3 Parameter tuning

To determine the optimal parameters for the machine learning pathway, the Data Science Machine runs a Gaussian Copula Process to model the relationship between parameter selection and model performance. At each iteration the optimization procedure must balance exploration and exploitation. The plots in Figure 8-2 show the maximum cross validation score obtained by iteration for all three datasets. The

KDD Cup 2014		IJCAI		KDD Cup 2015	
Entity	# Features	Entity	# Features	Entity	# Features
Projects	935	Merchants	43	Enrollments	450
Schools	430	Users	37	Users	202
Teachers	417	Behaviors	147	Courses	178
Donations	21	Categories	12	Outcomes	3
Donors	4	Items	60	Log	263
Resources	20	Merchants	43	Objects	304
Vendors	13	ActionType	36	ObjectChildren	3
Outcomes	13	Outcomes	82	EventTypes	2
Essays	9	Brands	12		

Table 8.2: The number of synthesized features per entity for in each dataset.

parameters of the best run are shown in Table 8.3.

Figure 8-2: the maximum cross validation AUC score found by iteration for all three datasets. From top to bottom: KDD Cup 2014, IJCAI, KDD Cup 2015

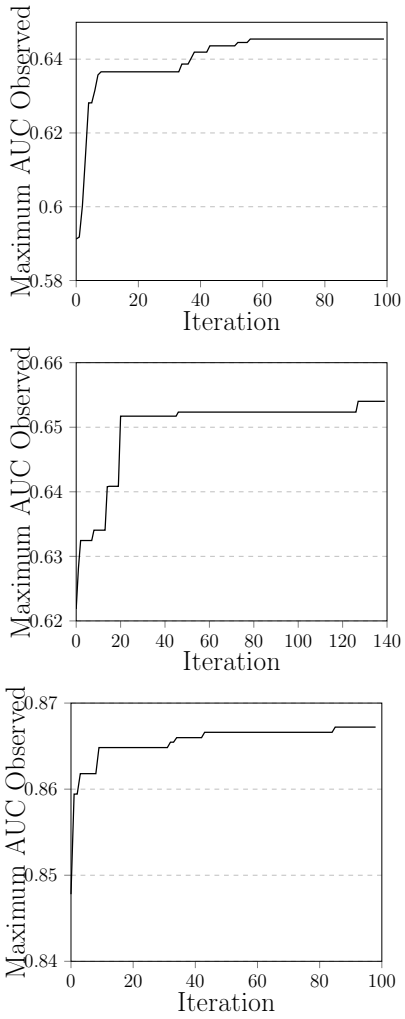


Table 8.3: Optimized pathway parameters from running GCP by dataset

Parameter	KDD Cup 2014	IJCAI	KDD Cup 2015
k	1	1	2
n_c	389	271	420
γ	32	65	18
rr	4	8	1
n	387	453	480
m_d	19	10	10
β	32	51	34

8.4 Competition results

We demonstrate the effectiveness of the Data Science Machine by applying it to the datasets where many data scientists are competing for the best results. We compare the results from our experiments with the public performance in these competitions to help us answer the question of "How does the performance of the Data Science Machine compare to human performance?".

To this end, we ran Data Science Machine with fully automated feature generation (*Deep Feature Synthesis*), feature selection, machine learning, and model tuning. There is no human involvement in the analysis of data beyond setting initial parameters of the Data Science Machine such that the computational limits are maximized. The results of running the Data Science Machine are presented in Table 8.4.

Parameter Selection	**KDD Cup 2014**		**IJCAI**		**KDD Cup 2015**	
	Local	Online	Local	Online	Local	Online
Default	.6059	.55481	.6439	.6313	.8444	.5000
GCP Optimized	.6321	.5863	.6540	.6606	.8672	.8621

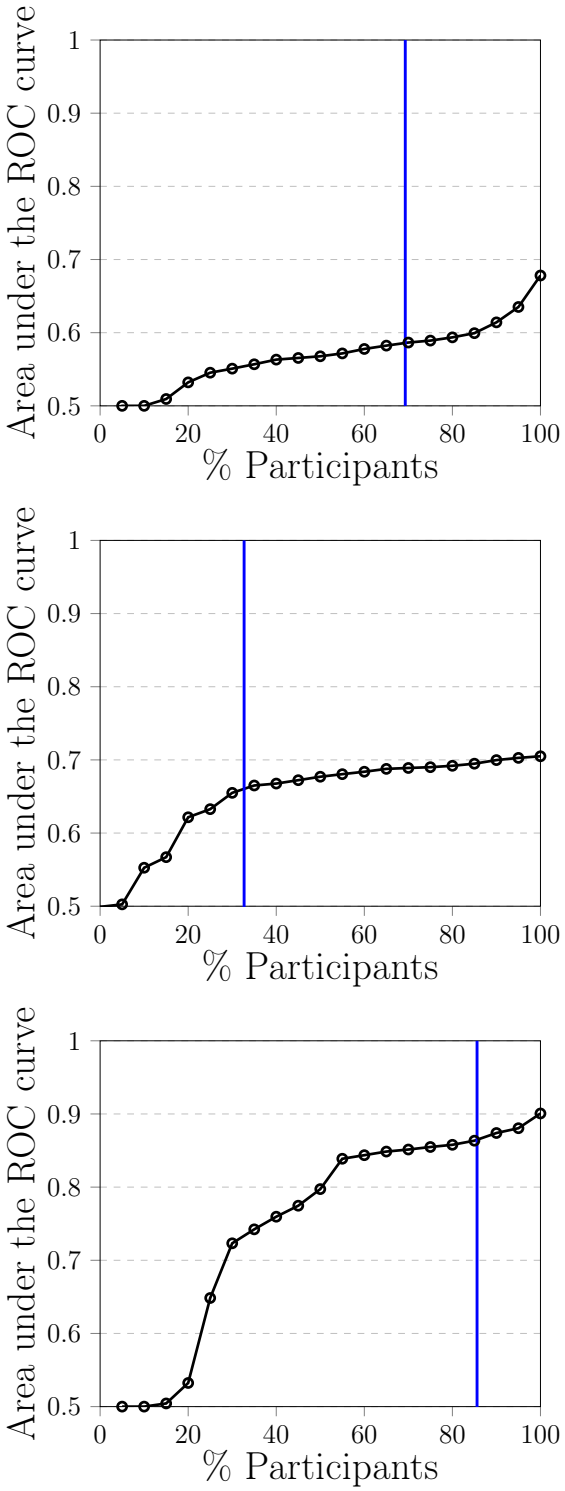
Table 8.4: The AUC scores achieved by the Data Science Machine. The "Default" score is the score achieved using the default parameters for the machine learning pathway. The "Local" score is the result of running k-folds (k=3) cross validation on the training set. The "Online" score is the score received when the result was uploaded for assessment by the competition.

Table 8.5 shows how the Data Science Machine did relative to other competitors on the leader board. The table presents some basic information about the teams that beat and the teams beaten by the Data Science Machine. To put these scores in perspective, Figure 8-3 shows how the Data Science Machine score compares to other competitors at each percentile of the leader board.

Dataset	**Teams**	**% of Best**	**% Better**	**# Submissions worse**	**# Days Saved**
KDD Cup 2014	473	86.5%	30.7%	3873	929
IJCAI	156	93.7%	67.3%	n\a	n\a
KDD Cup 2015	277	95.7	14.4%	1319	377

Table 8.5: How the Data Science Machine compares to human efforts. "% of Best" is the proportion of the best score the Data Science Machine achieved. "% Better" indicates the percentage of teams that out performed the Data Science Machine submission, while "# Submissions worse" is the count of all submissions made by teams that the Data Science Machine outperformed. To calculate "# Days Saved", we used the rule in KDD Cup 2014 and KDD Cup 2015 that teams could make up to 5 submissions a day. Using this rule, we make a lower bound estimate of the number of days spent by teams that rank below the Data Science Machine. We do not have data on number of submissions for IJCAI. KDD Cup 2015 is still an on going competition, so this is a snapshot from May 18th, 2015.

Figure 8-3: AUC scores vs % participant achieving that score. The vertical line indicates where the Data Science Machine ranked, From top to bottom: KDD Cup 2014, IJCAI, KDD Cup 2015



Chapter 9

Discussion

In many regards, the Data Science Machine is the first of its kind to emulate human aspects of the data science endeavor. In following sections, we discuss how results on all three datasets reflect the effectiveness of the Data Science Machine in emulating a data scientist.

9.1 Creating valuable synthesized features

If the synthesized features had no value, we would not expect the Data Science Machine to perform so well in the three competitions. In its best performance, it beat approximately 86% of other competitors. These results demonstrate that Data Science Machine can produce new features that at least have *some* value.

Even in the competition where the highest percentage of competitors beat the Data Science Machine (IJCAI), it is clear from the results that the system captured information about the problem similar to human intelligence. During the IJCAI competition, the sponsors released a benchmark model. The proposed model used a customer's history at merchants that are similar to the merchant that the customer is currently shopping at to make predictions. To do this, they proposed one way to measure merchant similarity. The sponsors claimed that this method would achieve an AUC of at least .65. This is noteworthy, because the Data Science Machine GCP optimized solution achieved an AUC of more than .66. This means that the Data

Science Machine was able to automatically derive a model that was at least as good as the human proposed one. Even more, if you look at the rate of improvement as a competitor's rank increases, the Data Science Machine score is just at the point where score improvement plateaus. This suggests that the Data Science Machine features captured the major aspects of the dataset while simply missing out on minor improvements.

In the other two competitions, the Data Science Machine features are enough for the Data Science Machine to beat a majority of competitors. Similar to IJCAI, the Data Science Machine is on the plateau of score improvement in KDD Cup 2015. Again, this makes a good case that the features synthesized by the Data Science Machine capture important aspects of the prediction problem. In KDD Cup 2014, the Data Science Machine features are good enough to perform better than most competitors but yet fall short of the top competitors as the rate of improvement does not plateau. We discuss the significance of this phenomena in Section 9.3.

9.2 Auto tuning effectiveness

The previous section discussed whether or not the Data Science Machine creates features of value. However, an important part of the Data Science Machine is selecting which of those features to use and how to tune the model to best use them. To solve this, the Data Science Machine employs an auto-tuning process. We can see across all three datasets that the Data Science Machine was able to increase its score, locally and online, by using auto tuning. By using an auto-tuning process, the Data Science Machine is able to design a machine learning pathway that can adapt itself to a range of problems rather than depend on the proper default parameters.

The testing with the KDD Cup 2015 dataset brought about an interesting insight with regard to the default parameters that highlights the significance of auto tuning. With the default parameters, the local cross-validation score seemed to produce a good result. However, when the predictions were uploaded to the competition website, the results were as good as if the Data Science Machine had guessed randomly.

This caused suspicions as to if there was something wrong with the training setup. However, as part of the troubleshooting process, we used the interface described in Chapter 7 to examine features the Data Science Machine created and to play around with machine learning pathway parameters. Using this interface, we were able to conclude that the issue was with the default parameters rather than with the training pathway itself.

We reached this conclusion by first turning off all features except for the simplest ones. This reduced the number of features that the modeling pathway considered by an order of magnitude. When we reran the the modeling pathway with default parameters, a cross-validation score was obtained as before, but when we uploaded it online, we observed an online score that had matched, unlike before.

Next, we re-enabled all features one by one and changed the value of each parameter from the default. When we did this, we observed that changing all parameters but one had no effect on mismatch with the online score. However, when we increased the parameter that controlled the number of trees in the random forest, the online score matched the local, cross-validated score. This suggested that our default value of 200 trees was not enough for the full dataset. Because each tree selects a small subset of features, when we had a lot of features, results did not show good generalizability. Decreasing the number of the features or increasing the number of trees solved this problem.

The process we followed is similar to the process of a practicing data scientist. First, we experimented with what features to include in our system, and then we experimented with parameters of the machine learning process.

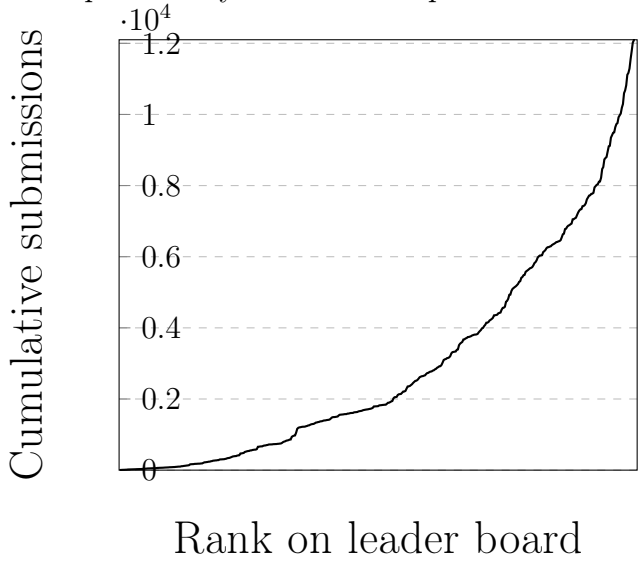
This is noteworthy because the auto-tuning process employed by the Data Science Machine did not encounter the issue described above. And, as we see in the results, it performed very well relative to competitors in KDD Cup 2015 when it came to selecting features and tuning parameters. The implication of this is that the Data Science Machine essentially removed hours of debugging from the work flow. Additionally, it highlights the difficulty in picking default parameters for machine learning algorithms. Without the auto-tuning process, the Data Science Machine would not

have achieved its goals.

9.3 Human value

The KDD Cup 2014 competition shows a good example of how human intelligence is still important. If we look at score growth by percentile, we can see that the top 10% of competitors significantly outperformed everyone else. The competition provides enough information to see that these top submissions were often made by teams of people and that the individuals involved have significant professional data science education and experience. Beyond this, the competitors clearly invested a lot of time refining their models. In the competition, 4 out of 5 of the top teams made over 200 submissions, representing well over a month of of thinking about the problem, as the competition only allowed 5 submission per day. Thus, it is reasonable to assume that top performers' gain over the other 90% of competitors were due to their "expert" knowledge of data science and close understanding of the problem domain.

Figure 9-1: The cumulative number of submissions made as leader board rank increases in KDD Cup 2014. We can see the total number of submissions made by competitors increased exponentially as we move up the leader board.



The Data Science Machine still performs very well against competitors who were

not able to figure out the insights that put the "experts" so far above everyone else. We can see this in Figure 8-3, where the Data Science Machine scores toward the end of the plateau. At some point, moving up the leader board might not be worth the cost required to do so. The large human effort to move up the leader board is shown in Figure 9-1.

9.4 Human-data interaction

The interface for the Data Science Machine is a first attempt at presenting a new paradigm to data scientists. It faced new challenges in explaining and organizing a large number of synthesized features. These challenges were, in many ways, unexpectedly hard, but they led to constructive insights.

1. Determining a systematic way to name features proved difficult. The solution proposed by the Data Science Machine works for user who invests the time to understand feature notation, but it is not welcoming to new users.

2. The feature selector interface is successful in organizing features by entity, but it is cumbersome when entities have hundreds of features. The scoring system that we implemented alleviates this to some extent, but its simplicity means it falls short of making good recommendations in problems where features must be combined for best performance.

3. The pathway parameter selection is successful at exposing raw parameters to the user, but it is unclear in which situations a user can do a better job choosing parameters than can a computer. It provides the option to do auto tuning, but further work is need to understand how to best expose parameters to users. Additionally, the current design of showing all parameters to the user will not scale if more parameters are added to the system.

4. The most successful concept in the interface is the encouragement of work flow based on designing experiments, evaluating the results, and iterating on new

insights. This process very much replicates the work flow of a data scientist but cuts out the part of the process where they have to go back to the data and extract new features. This enables faster iteration. By successfully making it easier for data scientists to test ideas, the Data Science Machine has used its algorithm to enable human creativity.

9.5 Implications for Data Scientists

The competitive success of the Data Science Machine suggests it has a role alongside data scientists. Currently, data scientists are very involved in the feature generation and selection process. However, our results show that we can automatically create features of value and figure out how to use those features in creating a model. Still, in all datasets, humans beat the Data Science Machine. However, the amount of effort put in by top competitors was likely high. Furthermore, there were probably competitors who performed worse even when they put in large amounts of effort. Thus, there seems to be a place for the Data Science Machine in data science.

First, the Data Science Machine can be used to set a benchmark. Just as the IJCAI organizers published a benchmark for competitors to use as reference, the Data Science Machine could be a performance reference for practicing data scientists. If the Data Science Machine performance is adequate for the purposes of the problem, no further work is necessary. This situation would save a lot of time if the dataset is similar to KDD Cup 2015 or IJCAI where most gains are achieved by the Data Science Machine, but further human work has diminishing marginal return. In the case of KDD Cup 2014 where further gains are significant, they appear to come at a high cost. The Data Science Machine solution with a cost-benefit analysis could make a significant impact on the time spent modeling predictive problems.

Second, the Data Science Machine can be used as a tool to enable data science creativity. The Data Science Machine works by exploring a large space of potential features and models for a problem in an attempt to find the best one. Data scientists often face the problem of having more ideas to consider than resources to test. Com-

bining these situations, we reach a proposal for the Data Science Machine. Rather than iterating on which features to select, the Data Science Machine can simply enumerate potential features and let data scientist iterate on feature selection. This suggests that the Data Science Machine can be the first step for data scientists to explore a problem. A data scientist can start with the Data Science Machine solution and then apply their expert knowledge to refine it. The interface we built is the first step toward this goal.

Chapter 10

Related Work

The key components of the Data Science Machine are

- Automated feature engineering
- Working with related data
- End-to-end system from data to predictions

Many related works have some of these components.

10.1 Automated feature engineering

Machine learning researchers and data scientists have long understood that feature engineering is the difference between success and failure. Generalized feature extraction algorithms have been well studied in various machine learning domains such as machine vision and natural language processing to solve this problem. In these domains, there are many examples of algorithms that perform well over a range of problems.

In machine vision, an early concept was Scale-invariant feature transform (SIFT) [13]. These types of features had success in generalizing to many problems and applications in machine vision such as object recognition and panorama stitching [8]. Other generalized feature extraction techniques have also emerged in machine vision

such as Histograms of oriented gradients [10] that perform well in other situations. In many cases, these generalized features have enabled machine vision researchers to spend less time deriving descriptive features for every new problem.

Similarly, in natural language processing, there are generalized features generation techniques. One simple model is term frequency-inverse document frequency (tf-idf), which is the ratio of how frequently a word shows up in a document to how often it shows up in the whole corpus of documents. This feature type has played an important role in text mining endeavours due to how easy it is to calculate and how well it performs [16]. Like vision, there has been a lot of development in generalized features to describe text. Latent Dirichlet Allocation (LDA) is a more recent technique that transforms a corpus of documents into document-topic mappings[6]. LDA has proven to be general purpose enough to be useful in many document classification tasks such as spam filtering [4] or article recommendation [20].

Importantly, while we consider these generalized algorithms to be useful, they still have to be tuned for best performance. For example, in LDA, choosing the number of topics in the corpus is a challenge big enough to encourage further research [6]. However, it is hard to call it a shortcoming when we consider all the success the basic LDA model has had. However, the importance of these algorithms is that they are a technique to generically capture information about type of data. But as we note in the next section, less work has been made toward extracting features from related data.

10.2 Working with related data

While the Data Science Machine focuses on data where we simply know that a relation between entities exists, the field of linked data strengthens these assumptions. In linked data [5], the data is structured such that it can be accessed with semantic queries. The field of automated feature generation for linked data an active area of research.

Cheng et al. developed an automated feature generation algorithm for data organized in a domain-specific knowledge base. This data is organized as entity-relationship-entity triples. For example, the Yago [19] knowledge base contains the following triple: `(Natalie Portman, hasActedIn, Black Swan)`. Using knowledge bases, the authors create a graph based language for generating features. In one example, they use natural language techniques to extract entities in tweets and relate them to entities in Yago. By relating tweets to entities, they can automatically generate new features with semantic meaning. Using this technique, they show some improvement in prediction results with their automatic semantic features. However, they conclude further improvement will likely come from more domain-specific knowledge bases and information retrieval techniques.

Other works have also focused on extracting features from knowledge bases. While Cheng et al. require the user to specify the feature type, Paulheim and Fümkranz focus on an unsupervised approach where they define 6 different generic feature generators. Some of their generic functions generate features that are not useful. To filter out these features, they apply the heuristic that removes features that have a lot of undefined values or few unique values.

Both of these works are limited to datasets that are structured in knowledge bases or can be mined for entities that are in knowledge bases. Many datasets do not fit these constraints, yet still have related entities. Developing an automated feature generation algorithm and system for making predictions in these cases are goals of the Data Science Machine.

10.3 End-to-end system

The Automatic Statistician project automatically models regression problems and produces human readable reports [12]. Their approach uses non parametric Gaussian processes to model regression functions. They demonstrate their system performing well on 13 different time series datasets, suggesting a certain level of generalizable. Their system is noteworthy in several regards. First, like the Data Science Machine,

it focuses on an input of relatively untransformed data into the system. In the case of the Automatic Statistician, it is time series data rather than relational data. Second, the approach of composing explanations one by one is similar to how a data scientist might analyze time series to generate the overall model. Finally, they demonstrate generating a report to explain the resulting model so the system can be understood by non-experts.

Chapter 11

Conclusion

We have presented the Data Science Machine: an end-to-end system for doing data science with relational data. At its core is *Deep Feature Synthesis*, an algorithm for automatically synthesizing features for machine learning. We demonstrated the expressiveness of the generated features on 3 datasets from different domains. By implementing an autotuning process, we optimize the whole pathway without human involvement, enabling it to generalize to different datasets. Overall, the system is competitive with human solutions to the datasets we tested on. We view this success as an indicator that the Data Science Machine has a role in the data science process.

11.1 Future work

We see the future direction of the Data Science Machine as a tool to empower data scientists. To this end, there are three important directions for future work.

User interface and Human-Data Interaction We took the first steps to exposing the functionality of the Data Science Machine to a user. In the discussion, we highlight several places where this succeeded and failed. Moving forward, we need to further explore how a data scientist could use our system. This should involve extensive user testing and iteration.

More datasets The Data Science Machine was tested on 3 different datasets, but

more datasets are will help us develop the generalizability to make the Data Science Machine the best tool for data science.

Improved performance A fast system enables us to iterate on new functionality and analyses new datasets efficiently. The performance level of this initial system suited our purposes, but improving performance will enable the Data Science Machine to be test on more and larger datasets.

Bibliography

[1] "data science" job trends. http://www.indeed.com/jobtrends?q=\%22Data+Science\%22&l=. Accessed: 2015-05-22.

[2] Mysql 5.6 reference manual: Alter table syntax, 2015. URL https://dev.mysql.com/doc/refman/5.6/en/alter-table.html.

[3] Mysql 5.6 reference manual: Create view syntax, 2015. URL https://dev.mysql.com/doc/refman/5.6/en/create-view.html.

[4] István Bíró, Jácint Szabó, and András A. Benczúr. Latent dirichlet allocation in web spam filtering. In *Proceedings of the 4th International Workshop on Adversarial Information Retrieval on the Web*, AIRWeb '08, pages 29–32, New York, NY, USA, 2008. ACM. ISBN 978-1-60558-159-0. doi: 10.1145/1451983.1451991. URL http://doi.acm.org/10.1145/1451983.1451991.

[5] Christian Bizer, Tom Heath, and Tim Berners-Lee. Linked data-the story so far.

[6] David M Blei, Andrew Y Ng, and Michael I Jordan. Latent dirichlet allocation. *the Journal of machine Learning research*, 3:993–1022, 2003.

[7] Eric Brochu, Vlad M. Cora, and Nando de Freitas. A tutorial on bayesian optimization of expensive cost functions, with application to active user modeling and hierarchical reinforcement learning. *CoRR*, abs/1012.2599, 2010. URL http://arxiv.org/abs/1012.2599.

[8] M. Brown and D.G. Lowe. Recognising panoramas. In *Computer Vision, 2003. Proceedings. Ninth IEEE International Conference on*, pages 1218–1225 vol.2, Oct 2003. doi: 10.1109/ICCV.2003.1238630.

[9] Weiwei Cheng, Gjergji Kasneci, Thore Graepel, David Stern, and Ralf Herbrich. Automated feature generation from structured knowledge. In *Proceedings of the 20th ACM international conference on Information and knowledge management*, pages 1395–1404. ACM, 2011.

[10] Navneet Dalal and Bill Triggs. Histograms of oriented gradients for human detection. In *Computer Vision and Pattern Recognition, 2005. CVPR 2005. IEEE Computer Society Conference on*, volume 1, pages 886–893. IEEE, 2005.

[11] Pedro Domingos. A few useful things to know about machine learning. *Communications of the ACM*, 55(10):78–87, 2012.

[12] James Robert Lloyd, David Duvenaud, Roger Grosse, Joshua B. Tenenbaum, and Zoubin Ghahramani. Automatic construction and Natural-Language description of nonparametric regression models. In *Association for the Advancement of Artificial Intelligence (AAAI)*, 2014.

[13] D.G. Lowe. Object recognition from local scale-invariant features. In *Computer Vision, 1999. The Proceedings of the Seventh IEEE International Conference on*, volume 2, pages 1150–1157 vol.2, 1999. doi: 10.1109/ICCV.1999.790410.

[14] Heiko Paulheim and Johannes Fümkranz. Unsupervised generation of data mining features from linked open data. In *Proceedings of the 2Nd International Conference on Web Intelligence, Mining and Semantics*, WIMS '12, pages 31:1–31:12, New York, NY, USA, 2012. ACM. ISBN 978-1-4503-0915-8. doi: 10.1145/2254129.2254168. URL `http://doi.acm.org/10.1145/2254129.2254168`.

[15] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[16] Anand Rajaraman and Jeffrey David Ullman. Data mining. In *Mining of Massive Datasets*, pages 1–17. Cambridge University Press, 2011. ISBN 9781139058452. URL `http://dx.doi.org/10.1017/CBO9781139058452.002`. Cambridge Books Online.

[17] Carl Edward Rasmussen. Gaussian processes for machine learning. 2006.

[18] Jasper Snoek, Hugo Larochelle, and Ryan P Adams. Practical bayesian optimization of machine learning algorithms. In *Advances in Neural Information Processing Systems*, pages 2951–2959, 2012.

[19] Fabian M Suchanek, Gjergji Kasneci, and Gerhard Weikum. Yago: a core of semantic knowledge. In *Proceedings of the 16th international conference on World Wide Web*, pages 697–706. ACM, 2007.

[20] Chong Wang and David M Blei. Collaborative topic modeling for recommending scientific articles. In *Proceedings of the 17th ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 448–456. ACM, 2011.

[21] Andrew Wilson and Zoubin Ghahramani. Copula processes. In *Advances in Neural Information Processing Systems*, pages 2460–2468, 2010.