A Relational Framework for Classifier Engineering

Benny Kimelfeld
Technion – Israel Institute of Technology
Haifa 32000, Israel
bennyk@cs.technion.ac.il

Christopher Ré
Stanford University
Stanford, CA 94305, USA
chrismre@cs.stanford.edu

ABSTRACT

In the design of analytical procedures and machine-learning solutions, a critical and time-consuming task is that of feature engineering, for which various recipes and tooling approaches have been developed. We embark on the establishment of database foundations for feature engineering. Specifically, we propose a formal framework for classification in the context of a relational database. The goal of this framework is to open the way to research and techniques to assist developers with the task of feature engineering by utilizing the database's modeling and understanding of data and queries, and by deploying the well studied principles of database management. We demonstrate the usefulness of the framework by formally defining key algorithmic challenges and presenting preliminary complexity results.

1. INTRODUCTION

A critical and time-consuming task in the development of analytics and machine-learning solutions is that of *feature engineering* [21, 35]. Given its importance, recipes and tooling have been developed for practitioners [1, 18]. With the advent of frameworks like SAS, Cloudera's IBIS and Oracle's ORE, feature engineering is often carried out over relational data. Thus, a pressing challenge is to understand how to merge these analytics with traditional database management techniques. To this end, we propose a relational framework for *classification*—a simple and popular analytic task.

The task of feature engineering is that of generating inputs (or *signals*) from available data, in order to improve the performance of the underlying model in solving a target problem. This target problem is typically classification (predicting an unknown category of a given entity), or regression (predicting the value of an unknown function on a given entity). The model makes its prediction based on various properties, called *features*, of the given entity. We focus here on *parametric models*, where the model has a pre-determined structure with numerical parameters that are tuned by fitting to training examples, a process termed *learning*.

This article is a minor revision of the work published in PODS 2017, May 14-19, 2017, Raleigh, NC, USA, ©2017 ACM. ISBN978-1-4503-4198-1/17/05...$15.00 DOI: http://dx.doi.org/10.1145/3034786.3034797

Naturally, the choice of features has a major impact on the resulting model. A suboptimal set of features may lead to *overfitting* (where the learned model does not well generalize beyond the training examples), or to *underfitting* (the model is incapable of capturing the target function due to lack of information or expressiveness) [18]. Another consideration is that of *execution cost*, as costly features may result in an impractical model [35]. There are also legal and moral considerations in cases where decision makers are required to practice fairness and lack of discrimination; there, a central challenge is to select appropriate features [10, 34, 36].

Our running example is a scenario where the security branch of a credit-card company aims to make an educated guess on whether an incoming transaction is a fraudulent purchase (e.g., made on a stolen card). To make such a guess, the company uses information available in its database, such as whether there were similar transactions in the past, whether the purchase is made in the same state of the owner's mailing address, or in the same country, the amount being paid, and so on. An employed engineer then selects a machine-learning library and learns a classifier, which can be trained based on past data on fraudulent activity.

The classifier is simply a function that maps a vector of numbers into a yes/no decision; in turn, these numbers, called *features*, encode relevant pieces of information (e.g., $f = +1/-1$ depending on whether or not the transaction is in the country of the owner). The machine-learning library typically tunes parameters (or weights) of the classifier by fitting them to the examples. In our scenario, high quality is crucial: false positives disrupt legitimate business, and false negatives cause financial losses and customer distress. So, the engineer produces additional features to consider by phrasing different questions (*feature queries*) about the transaction at stake, until she is satisfied with the results. This activity is referred to as *feature engineering*.

Analysts typically spend the bulk of their time on feature engineering [17, 21, 35]. This process includes trial and error, and stepwise addition or removal of features [18]. Involving the database semantics in feature engineering has the potential to automate some important tasks, and thereby assist the engineer. For instance, the engineer may ask whether some class of simple feature queries (e.g., select-project-join) suffices to achieve good classification based on the training data, or otherwise more expressiveness is needed. This motivates the *separability* problem that we later discuss. She may also ask the complementary question, which is whether the current set of feature queries is too detailed and allows the model to *overfit* the examples and poorly generalize to

future transactions. A traditional way to measure overfitting potential is via the *Vapnik-Chervonenkis* (*VC*) dimension [31]. The features of choice may have impact on the VC dimension, and we later refer to the problem of determining this impact as *VC dimensionality*.

The engineer may also encounter the common technical challenge where the machine-learning library fails due to incompatible data, as it requires the matrix of training data to be of a full (column) degree. Partial degree (column dependence) may be an artifact of the training examples; but it may also be an inherent error in the feature design. As an example, for US owners the features "payment is in the US," "payment is in the owner's state," and "payment is in a different US state" have an inherent linear dependence among them: the first minus the second equals the third. We could use the database to detect such problems. This leads to the problem that we later refer to as *identifiability*.

Contribution

Our goal is to establish the first steps in a database theory that embeds the automation of core tasks in feature selection. More precisely, our framework aims to open the way to novel research and techniques for utilizing the database's understanding of raw data and queries, in order to fundamentally assist with the process of feature engineering.

The framework is based on an *entity schema*, which is a relational schema with a distinguished relation symbol that represents *entities*. A *database instance* over an entity schema represents a collection of entities, along with additional (direct or indirect) knowledge about these entities. A *feature query* selects entities with a certain property, and a *statistic* is a sequence of feature queries. The central goal in classification is to train and apply a *classifier* (or a *classifier model*), which is a function that takes as input the *statistic* of an entity (i.e., the vector obtained by applying each feature query) and outputs a $+1/-1$ decision. In *training* a classifier, we are given a set of entities labeled with $+1/-1$, and produce a classifier from a predefined *model class*. A linear classifier, for example, is encoded as a vector of weights over the features in the statistic. In our framework, the collection of training examples is represented simply by an instance over the entity schema, along with a labeling function that maps each entity to $+1/-1$.

For illustration, Figure 1(a) shows an entity schema $\mathbf{S}$ where entities are transactions (identified by numbers), and Figure 1(b) shows a statistic Π over $\mathbf{S}$ with two feature queries π_1 and π_2. Figure 2 shows the training workflow in our framework: an instance I over $\mathbf{S}$ with a labeling of the entities (transactions) is transformed into a $(+1/-1)$-matrix, with a feature row per entity, and the matrix is used for building a classifier h. This classifier predicts the labels in a new instance, as illustrated in Figure 3.

The features, as defined above, are based on boolean properties of the entities: if the entity satisfies the property (i.e., it is "selected"), then the feature value is $+1$, and otherwise -1. Boolean features are highly important in practice, and in fact, we are aware of quite a few deployments where numerical values are translated into boolean ones (e.g., by means of *bucketing*, or *binning*, numbers into intervals). The more general case is that in which the feature query associates a numerical value with each entity, and we discuss this generalization as a future direction later on.

Our framework enables to formalize relevant computational problems, analyze their complexity, and ultimately design algorithmic solutions. Here, we formally define three computational problems that capture tasks in the construction and evaluation of features: *separability*—whether a perfect separator exists for a training set; *VC dimensionality*—computing the fundamental measure of the complexity of a classifier class; and *identifiability*—testing for a statistical property guaranteeing that the classifier model is uniquely defined given enough data.

Our analysis focuses on linear classifiers and features definable as conjunctive queries. Specifically, we show a tight relationship between the computational complexity of our problems and that of *query containment* (and *equivalence*): it is necessary, and often sufficient, to solve containment in order to solve our problems. In this article, we present our complexity results on the separability problem. Additional results and proofs on all three problems can be found in the conference version of this article [22].

2. PRELIMINARIES

In this section we give the basic definitions and terminology that we use throughout the article.

Relational Databases

Our relational terminology is as follows. A *schema* is a collection of *relation symbols*. Each relation symbol R has an associated *arity* k. We assume an infinite set Const of *constants*. An *instance* I over a schema $\mathbf{S}$ associates with every k-ary relation symbol $R \in \mathbf{S}$ a finite subset of Const^k. We denote by R^I the relation that I associates with the relation symbol R. The *active domain* of an instance I, denoted $adom(I)$, is the set of all the constants in Const that are mentioned in the tuples of I. A *fact* over a schema $\mathbf{S}$ is an expression of the form $R(c_1, \ldots, c_k)$, where R is a k-ary relation symbol and $c_1, \ldots, c_k$ are constants. We say that the fact $R(c_1, \ldots, c_k)$ *belongs to* an instance I over $\mathbf{S}$ if R^I contains the tuple $(c_1, \ldots, c_k)$. For convenience, we view an instance as the set of its facts. In particular, by $f \in I$ we denote that f belongs to I.

COMMENT 2.1. While schema *constraints* are important in our framework, they are excluded from the basic framework for simplicity sake. Moreover, the complexity results we give later on (Section 5) are oblivious to such constraints. We further discuss constraints in Section 7. □

Let I and J be two instances over the same schema $\mathbf{S}$. A *homomorphism* from I to J is a mapping $\mu : adom(I) \to adom(J)$ such that for every fact $f \in I$ we have $\mu(f) \in J$; here, $\mu(f)$ is the fact that is obtained from f by replacing each constant a with the constant $\mu(a)$.

Queries

A *query* over a schema $\mathbf{S}$ is a function q that is associated with an arity k, and that maps every instance I over $\mathbf{S}$ into a finite subset $q(I)$ of Const^k. A query q' *contains* a query q, in notation $q \subseteq q'$, if $q(I) \subseteq q'(I)$ for all instances I over $\mathbf{S}$. If $q \subseteq q'$ and $q' \subseteq q$ then q and q' are said to be *equivalent*. We have a special interest in *unary* queries q (i.e., where $k = 1$); then, by a slight abuse the notation, we view $q(I)$ as a set of constants a rather than a set of tuples (a).

We consider conjunctive queries *without constants*. Formally, a Conjunctive Query (CQ) over a schema $\mathbf{S}$ is a logical

Table 1: Main symbols

h	hypothesis/classifier $\{-1,1\}^n \to \{-1,1\}$
$\mathbf{H}$	hypothesis class
Lin	the class of linear classifiers
$\mathbf{S}$	relational/entity schema
η, $\eta_{\mathbf{S}}$	entity relation (unary)
I, J	database instance
η^I, $\eta^I_{\mathbf{S}}$	entity set of the instance I
e	entity in $\eta^I_{\mathbf{S}}$
$\mathbf{QL}$	query language
CQ	the class of CQs (without constants)
π	feature query (unary)
$\pi^I(e)$	$+1$ if $e \in \pi(I)$ and -1 if $e \notin \pi(I)$
Π	statistic $(\pi_1, \ldots, \pi_n)$
λ	labeling function $\eta^I_{\mathbf{S}} \to \{-1,1\}$

formula $q(\mathbf{x})$ of the form

$$\exists \mathbf{y}[\phi_1(\mathbf{x},\mathbf{y}) \wedge \cdots \wedge \phi_m(\mathbf{x},\mathbf{y})]$$

where $\mathbf{x}$ and $\mathbf{y}$ are disjoint sequences of variables and each ϕ_i is an atomic query over $\mathbf{S}$ (i.e., a formula that consists of a single relation symbol and no logical operators) without constants. Observe that in this article CQs do not contain built-in relations such as $x > y$; we use this assumption in our analysis. The atomic formula ϕ_i is called an *atom* of q. We use the conventional notation

$$q(\mathbf{x}) \leftarrow \phi_1(\mathbf{x},\mathbf{y}), \cdots, \phi_m(\mathbf{x},\mathbf{y})$$

to denote a CQ. The left side $q(\mathbf{x})$ is called the *head* and the right side $\phi_1(\mathbf{x},\mathbf{y}), \cdots, \phi_m(\mathbf{x},\mathbf{y})$ is called the *body*. We require each variable in the head to occur at least once in the body. We may refer to a CQ by mentioning only its head $q(\mathbf{x})$ or even just q. We denote by CQ the class of CQs (as defined here, i.e., without constants).

Let $\mathbf{S}$ be a schema, let q be a CQ over $\mathbf{S}$, and let I be an instance over $\mathbf{S}$. A *homomorphism* from q to I is a mapping from the variables of q to $adom(I)$, such that for every atom ϕ of q, the fact $\mu(\phi)$ belongs to I; here, $\mu(\phi)$ is the fact that is obtained from ϕ by replacing each variable z with the constant $\mu(z)$. The result of applying the CQ $q(\mathbf{x})$ to the instance I is the relation that consists of all the tuples $\mu(\mathbf{x})$, where μ is a homomorphism from q to I and $\mu(\mathbf{x})$ is obtained from $\mathbf{x}$ by replacing every variable x_i with $\mu(x_i)$. We denote this relation by $q(I)$.

Classifiers and Learning

In this work, a *classifier* is a function of the form

$$h : \{-1,1\}^n \to \{-1,1\}$$

where n is a natural number that we call the *arity* of h. A *hypothesis class* is a (possibly infinite) family $\mathbf{H}$ of classifiers, and a classifier in $\mathbf{H}$ is referred to as a *hypothesis*. We denote by $\mathbf{H}_n$ the restriction of $\mathbf{H}$ to the n-ary hypotheses in $\mathbf{H}$. An n-ary *training collection* is a multiset T of pairs $\langle \mathbf{x}, y\rangle$ where $\mathbf{x} \in \{-1,1\}^n$ and $y \in \{-1,1\}$. We denote by $\mathbf{T}_n$ the class of all n-ary training collections. A *cost function* for a hypothesis class $\mathbf{H}$ is a function of the form

$$c : \big(\cup_n (\mathbf{H}_n \times \mathbf{T}_n)\big) \to \mathbb{R}_{\geq 0}$$

where $\mathbb{R}_{\geq 0}$ is the set of nonnegative numbers. Given a training collection T and two hypotheses h_1 and h_2, the inequality $c(h_1,T) > c(h_2,T)$ implies that h_2 is preferred to h_1 according to c. In the context of a fixed hypothesis class $\mathbf{H}$ and a cost function c, *learning* a classifier is the task of finding a hypothesis $h \in \mathbf{H}_n$ that minimizes $c(h,T)$, given a training collection $T \in \mathbf{T}_n$.

It is importnat to allow T to be a multiset in order to enable the scoring function to account for the frequency (rather than pure existence) of examples. For the scope of this article, though, being a multiset does not play any role, and the reader may view T simply as a set.

We illustrate our definitions on the important class of *linear classifiers*. An n-ary linear classifier is parameterized by a vector $\mathbf{w} = (w_0, \ldots, w_n) \in \mathbb{R}^{n+1}$, denoted by $\Lambda_{\mathbf{w}}$, and defined as follows for all $\mathbf{a} \in \{-1,1\}^n$.

$$\Lambda_{\mathbf{w}}(\mathbf{a}) \stackrel{\text{def}}{=} \begin{cases} 1 & \text{if } \mathbf{a} \cdot \mathbf{w}' \geq w_0; \\ -1 & \text{otherwise.} \end{cases}$$

where $\mathbf{w}' = (w_1, \ldots, w_n)$ and "$\cdot$" denotes the operation of dot product. By Lin we denote the class of linear classifiers. An example of a cost function is the *least square* cost that is given by

$$lsq(\Lambda_{\mathbf{w}}, T) \stackrel{\text{def}}{=} \sum_{\langle \mathbf{x}, y\rangle \in T} \big(\mathbf{x} \cdot \mathbf{w}' - w_0 - y\big)^2$$

for the arguments $\Lambda_{\mathbf{w}} \in \mathsf{Lin}_n$ and $T \in \mathbf{T}_n$.

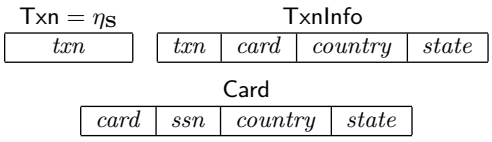
More background on the basic theory of machine-learning classifiers, as well as the relevant linear algebra discussed in the next section, can be found in standard machine-learning textbooks, such as Shalev-Shwartz and Ben-David [28].

Matrix independence. We denote by $\mathbf{0}^n$ the vector of n zeroes, and by $\mathbf{1}^n$ the vector of n ones. Let M be an $n \times m$ real matrix (consisting of n rows and m columns). A *linear column dependence* in M is a vector $\mathbf{w} \in \mathbb{R}^m$ such that $\mathbf{w} \neq \mathbf{0}^m$ and $M \cdot \mathbf{w} = \mathbf{0}^n$. A linear column dependence $\mathbf{w}$ in M is an *affine dependence* in M if $\mathbf{w} \cdot \mathbf{1}^m = 0$ (i.e., the components of $\mathbf{w}$ sum up to 0). If M does not have any linear column dependence, then we say that M is *linearly column independent*. Similarly, if M does not have any affine column dependence, then we say that M is *affinely column independent*. Note that linear independence implies affine independence, but the other direction is not necessarily true.

3. FRAMEWORK

We now present our formal framework. A basic notion in this framework is that of an *entity schema*, which is simply an ordinary relational schema with a distinguished relation symbol for representing *entities*. For simplicity, we assume that an entity is represented by a single constant (an identifier), hence the corresponding relation is unary. Formally, an entity schema is a pair $(\mathbf{S}, \eta)$, where $\mathbf{S}$ is a schema and η is a unary relation symbol in $\mathbf{S}$. An *instance* over an entity schema $(\mathbf{S}, \eta)$ is simply an instance over $\mathbf{S}$. In the remainder of this article all the schemas we consider are entity schemas. So, to simplify the presentation we refer to the entity schema $(\mathbf{S}, \eta)$ simply as $\mathbf{S}$, and refer to η as $\eta_{\mathbf{S}}$.

Let I be an instance over an entity schema $\mathbf{S}$. An *entity* of I is a constant a such that $\eta_{\mathbf{S}}(a) \in I$. Hence, I represents a set of entities along with information about the entities. This information is contained in the remaining re-



(a) An entity schema $\mathbf{S}$

$$\pi_1(t) \leftarrow \mathsf{TxnInfo}(t, n, c, s), \mathsf{Card}(n, i, c, s)$$
$$\pi_2(t) \leftarrow \mathsf{TxnInfo}(t, n, c, s), \mathsf{Card}(n, i, c, s')$$

(b) A statistic $\Pi = (\pi_1, \pi_2)$ over $\mathbf{S}$

Figure 1: An entity schema and a statistic

lations, which can be joined with $\eta_{\mathbf{S}}$. Again, by a slight abuse of notation, we treat $\eta_{\mathbf{S}}^I$ as the set of all entities of I. For example, $e \in \eta_{\mathbf{S}}^I$ means that e is an entity of I. A *feature query* (*over* $\mathbf{S}$) is a unary query π over the schema $\mathbf{S}$. When the feature query π is represented in a query language $\mathbf{QL}$ (e.g., CQ), we say that π is *in* $\mathbf{QL}$.

Example 3.1. We use a running example that instantiates the credit-card scenario from the Introduction. Figure 1(a) depicts the entity schema $\mathbf{S}$ with a unary relation Txn, which is $\eta_{\mathbf{S}}$, and two quaternary relations TxnInfo and Card. The box on the top of Figure 3 depicts an instance I'. The entities are the transaction identifiers 5, 6 and 7, and these are the members of $\eta_{\mathbf{S}}^{I'} = \{5, 6, 7\}$. Figure 1(b) shows two feature queries in CQ: the feature π_1 selects all transactions that took place in the same country and state of the owner's maling address, and the feature query π_2 selects all the ones that took place in the same country (but not necessarily the same state) of the owner. Indeed, in π_1 the two atoms use the same variable, s, for the state, while in π_2 the first atom uses s and the second uses s'. □

Let I be an instance over an entity schema $\mathbf{S}$, and let π be a feature query. We define the function $\pi^I : \eta_{\mathbf{S}}^I \to \{-1, 1\}$ as follows.

$$\pi^I(e) = \begin{cases} 1 & \text{if } e \in \pi(I); \\ -1 & \text{otherwise.} \end{cases}$$

Let $\mathbf{S}$ be an entity schema. A *statistic* (over $\mathbf{S}$) is a sequence $\Pi = (\pi_1, \ldots, \pi_n)$ of feature queries. We say that Π is in a query language $\mathbf{QL}$ if each π_i is in $\mathbf{QL}$. Given an instance I over $\mathbf{S}$, we denote by Π^I the function $(\pi_1^I, \ldots, \pi_n^I)$ from $\eta_{\mathbf{S}}^I$ to $\{-1, 1\}^n$ that maps every entity $e \in \eta_{\mathbf{S}}^I$ to the sequence $(\pi_1^I(e), \ldots, \pi_n^I(e))$.

Example 3.2. Figure 1(b) describes the statistic $\Pi = (\pi_1, \pi_2)$ over the schema $\mathbf{S}$ of Figure 1(a). The middle layer of Figure 3 contains (on its left) the tuples $\Pi^{I'}(e)$ for the entities e in the instance I' of the top box in this figure. For example, the top row corresponds to $\Pi^{I'}(5) = (1, 1)$, which is due to the fact Transaction 5 took place in the same country and state of the card holder. □

Let $\mathbf{S}$ be an entity schema. A *labeling* of an instance I over $\mathbf{S}$ is a function

$$\lambda : \eta_{\mathbf{S}}^I \to \{-1, 1\}$$

that partitions the entities into *negative examples* (i.e., entities e where $\lambda(e) = -1$) and *positive examples* (i.e., entities e where $\lambda(e) = 1$). A *training instance* over $\mathbf{S}$ is a pair (I, λ), where I is an instance over $\mathbf{S}$ and λ is a labeling of I. Taken together, a statistic Π and a training instance define a training collection, namely, the one that consists of the tuple $\langle \Pi^I(e), \lambda(e) \rangle$ for every entity $e \in \eta_{\mathbf{S}}^I$.

Example 3.3. Continuing our running example, Figure 2 depicts a training instance (I, λ) over the entity schema $\mathbf{S}$ of Figure 1(a), where λ is represented in the Txn relation. With the statistic Π of Figure 1(b) we get the training collection in the left bottom part of Figure 2. From this training instance a classifier h is learned, and is applied for prediction on future instances, as illustrated in Figure 3 for the instance I' that we referred to in the previous examples. □

4. COMPUTATIONAL PROBLEMS

We now define three computational problems that are motivated by the design of machine-learning solutions, and feature engineering in particular.

4.1 Separability

Separability is perhaps the most basic notion of learning. The traditional presentation of learning theory typically begins with the "noise free" case where the labeled examples are required to be perfectly separated by the features. In our framework, *separability* refers to the following task: given a training instance over an entity schema, determine whether there exists a statistic and a classifier that agree with (i.e., classify precisely as) the example labels. Separability is a simplification of the more general problem, where some noise is allowed (and say, 99% of the examples are required to be correctly satisfied). We adopt the simplified (textbook) task as a first step, and show that it already leads to nontrivial insights within our framework.

The problem is parameterized by two important components: the family of classifiers in consideration, and the query language used for phrasing feature queries. The formal defintion of the problem is as follows.

Let $\mathbf{S}$ be a schema, Π a statistic over $\mathbf{S}$, and $\mathbf{H}$ a hypothesis class. A training instance (I, λ) is $\mathbf{H}$-*separable* with respect to (w.r.t.) Π if there exists a hypothesis $h \in \mathbf{H}$ that *fully agrees* with λ, that is, h and Π have the same arity and $h(\Pi^I(e)) = \lambda(e)$ for every $e \in \eta_{\mathbf{S}}^I$.

Problem 1 (Separability). *For a hypothesis class* $\mathbf{H}$ *and a query language* $\mathbf{QL}$*, the problem* $(\mathbf{H}, \mathbf{QL})$-separability *is the following. Given an entity schema* $\mathbf{S}$ *and a training instance* (I, λ) *over* $\mathbf{S}$*, determine whether there exists a statistic* Π *in* $\mathbf{QL}$ *such that* (I, λ) *is* $\mathbf{H}$*-separable w.r.t.* Π.

Example 4.1. We continue with our running example, and consider the training instance (I, λ) of Figure 2. Suppose that $\mathbf{H}$ is the class Lin of linear classifiers, and that $\mathbf{QL}$ is the class CQ. Then (I, λ) is a "yes" instance of the separability problem, and a witness is the statistic $\Pi = (\pi_1, \pi_2)$ of Figure 1(b) with the classifier $\pi_2 - \pi_1 \geq 1$. Now suppose that we add an entity 5, the tuple (5, 102, US, AL) to TxnInfo, and the labeling $\lambda(5) = -1$. The new training instance then becomes a "no" instance of the separability problem since, intuitively, there is no way to distinguish between 4 and 5 using $\mathbf{QL}$ over I, and yet, λ labels 4 and 5 differently. □

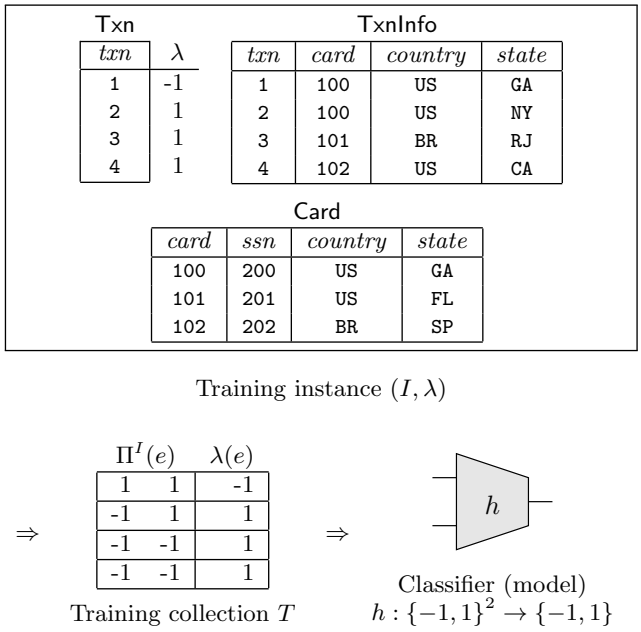


Figure 2: The training process

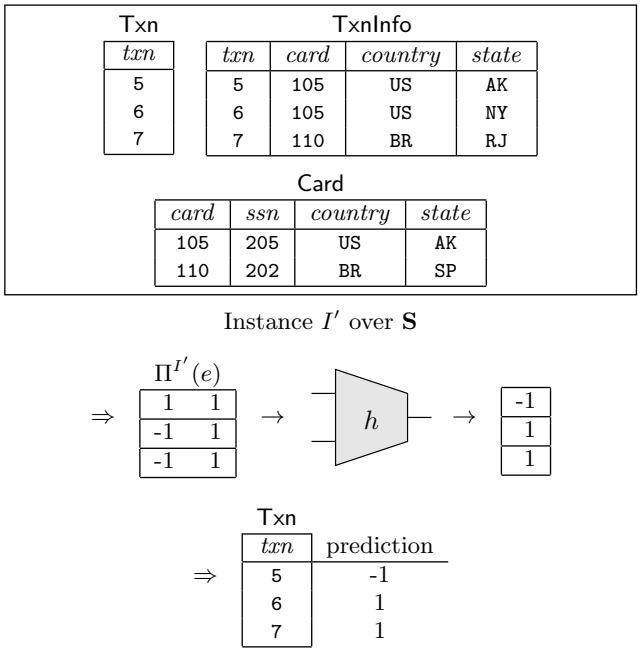


Figure 3: The prediction process

4.2 VC Dimensionality

The *Vapnik-Chervonenkis* (*VC*) dimension [31] is a measure of complexity of a hypothesis class, and is a de facto complexity measure for learnability. Bounds for *generalization* (how well a learned classifier does on unseen data) typically depend on the VC dimension. It measures the capacity of the classifier class, and is a key indicator of how much data one needs to reliably train the classifier: if this amount is low with respect to the VC dimension, then the classifier may overfit. If the amount of training data is high with respect to the VC dimension, we may be missing opportunities to devise a more accurate classifier.

As an example, the class of *polynomial* classifiers is more expressive than that of the *linear* classifiers, so there is a higher capability of a polynomial-classifier learner to overfit, that is, exploit properties that are exhibited scarcely in the training examples but are not representative of the general population. Similarly, a *deep* decision tree might be constructed to handle every individual example, while a *shallow* one will have to utilize common properties, and hence, intuitively, to better generalize. VC dimension is a mathematical measure that aims to capture this expressive power in a manner that is uniform across model classes. Higher VC dimension implies a more complicated classifier space with higher ability to overfit training data, and so, more training data is required for effective learning.

In our framework, VC dimension is a function of not only the hypothesis class, but also the statistic that translates entities into feature vectors. The formal definition follows.

Let $\mathbf{S}$ be a schema, Π a statistic over $\mathbf{S}$, and $\mathbf{H}$ a hypothesis class. An instance I over $\mathbf{S}$ is *shattered by* $\mathbf{H}$ *w.r.t.* Π if for every labeling λ of I there exists a hypothesis $h \in \mathbf{H}$ that fully agrees with λ. The *VC dimension of* $\mathbf{H}$ *w.r.t.* Π is the maximal number m such that there is an instance I over $\mathbf{S}$ where I has m entities and I is shattered by $\mathbf{H}$ w.r.t. Π.

Problem 2 (Dimensionality). *Let* $\mathbf{H}$ *be a hypothesis class and* $\mathbf{QL}$ *a query language. The computational problem* $(\mathbf{H}, \mathbf{QL})$*-dimensionality is the following. Given an entity schema* $\mathbf{S}$ *and a statistic* Π *in* $\mathbf{QL}$*, compute the VC dimension of* $\mathbf{H}$ *w.r.t.* Π.

Example 4.2. Recall $\mathbf{S}$ and Π of our running example (Figure 1). Computing the VC dimension of Lin w.r.t. Π is an instance of (Lin, CQ)-dimensionality. Our results [22] imply that this dimension is 3. Hence, there exists an instance I with three entities, such that we can find a perfect linear classifier for every labeling λ for I. Yet, no such instance exists with four or more entities. In this example, then, modeling of the features as CQs does not reduce the VC dimension compared to traditional machine learning where one can freely set the feature values. □

4.3 Statistic Identifiability

Identifiability asks whether it is possible for one to learn the parameters of the given classifier model *unambiguously* from *some* data set. Here, the question refers to a given statistic, and we consider the case where training is done by means of optimization via linear algebra; we ask whether the space of solutions is bounded. More formally, this problem boils down to deciding, given a statistic, whether there exists any training instance such that the resulting feature matrix is of full column dimension (i.e., the columns are linearly independent). We also consider the variant where linear independence is relaxed to affine independence (as defined in Section 2.) For additional background on identifiability, we refer the reader to Wainwright and Jordan's survey [32]. Next, we give the formal definition.

Let $\mathbf{S}$ be an entity schema, Π a statistic over $\mathbf{S}$, and I an instance over $\mathbf{S}$. We fix an arbitrary order over the entities of I, and denote by $[\![\Pi^I]\!]$ the matrix that consists of the rows $\Pi^I(e)$ for every $e \in \eta^I_{\mathbf{S}}$ in order. We say that Π is *linearly identifiable* if there exists an instance I over $\mathbf{S}$ such that the matrix $[\![\Pi^I]\!]$ is linearly column independent. We say that Π is *affinely identifiable* if there exists an instance I over $\mathbf{S}$ such

that the matrix $[\![\Pi^I]\!]$ is affinely column independent. Note that whenever Π is linearly identifiable, it is also affinely identifiable; the other direction is not necessarily true.

Both types of identifiability are important properties in the design of machine-learning solutions [24]. Particularly, in the case of the hypothesis class Lin and the cost function *lsq* (as defined in Section 2), linear independence implies that there is a single optimal hypothesis, whereas its absence implies that the space of optimal solutions is unbounded. Affine independence likewise arises in different cost functions such as *maximum entropy* [32]. The corresponding computational problem is formally defined as follows.

PROBLEM 3 (IDENTIFIABILITY). *Let* $\mathbf{QL}$ *be a query language. The computational problem of* linear *(respectively,* affine*)* $\mathbf{QL}$-identifiability *is that of testing, given an entity schema* $\mathbf{S}$ *and a statistic* Π *over* $\mathbf{S}$*, whether* Π *is linearly (respectively, affinely) identifiable.*

EXAMPLE 4.3. Consider again $\mathbf{S}$ and Π of our running example (Figure 1). Then $\mathbf{S}$ and Π form a "yes" instance of linear (and affine) CQ-identifiability. Indeed, Π is linearly (and affinely) identifiable, and a witness instance is I of Figure 2 with Txn restricted to the entities `1` and `2` (or `3` and `2`, but not `1` and `3`). We have shown that under certain conditions (that hold in our case), a statistic that consists of CQ feature queries is always identifiable, unless two or more of the feature queries are equivalent [22]. Hence, in the case of CQs, identifiability "comes for free" up to redundancy. □

4.4 Complexity Analysis

Complexity analysis of the three problems can be found in the conference version of this article [22], and will be presented in more detail in the full version of the paper. In the next section, we give complexity results on the first problem, namely separability.

5. COMPLEXITY OF SEPARABILITY

In this section, we discuss the complexity of separability in the case where feature queries are from the class of CQs and the hypothesis class is that of linear classifiers. The first result states coNP-completeness.

THEOREM 5.1. $(\mathsf{Lin}, \mathsf{CQ})$*-separability is a* coNP*-complete problem. Moreover, there exists a fixed entity schema* $\mathbf{S}$ *such that* $(\mathsf{Lin}, \mathsf{CQ})$*-separability is coNP-hard over* $\mathbf{S}$*.*

The proof of Theorem 5.1 consists of two parts. In the first part, we show that a given (I, λ) is Lin-separable w.r.t. a given statistic Π if and only if every two entities with different labels (i.e., one is $+1$ and the other -1) can be *distinguished* by a CQ, that is, there is a CQ feature query that returns one entity and not the other. The second part shows that this distinguishability test is coNP-complete. This proof highlights a connection to the problems of *query-by-example* and *definability* [7, 30, 33], and we are currently exploring these connections in more depth.

In the above proof of hardness we construct CQs over a fixed schema, but self joins are allowed. Next, we consider the case of self-join-free CQs. Formally, a CQ q is *self-join free* if it does not have two distinct atoms with the same relation symbol. We denote by $\mathsf{CQ}_{\mathsf{sjf}}$ the class of CQs without self joins. Interestingly, disallowing self joins (hence, restricting the space of statistics to simpler CQs) does not make the problem easier. In fact, under conventional complexity assumptions, it becomes harder!

THEOREM 5.2. $(\mathsf{Lin}, \mathbf{QL}_{\mathsf{sjf}})$*-separability is* Σ_2^{P}*-complete.*

Intuitively, the reason for the increased complexity is that self joins allow us to (efficiently) formulate a single statistic Π of representative ("canonical") feature CQs that captures the entire space of statistics; that is, if any statistic provides separation, then so does Π. In particular, with self joins the problem boils down to deciding on the existence of a homomorphism. Yet, without self joins it appears that we cannot do better than to inspect an exponential space of statistics, and solve the homomorphism problem in each. Finally, we remark that fixing the schema $\mathbf{S}$ in the case of $\mathbf{QL}_{\mathsf{sjf}}$ would make the separability problem solvable in polynomial time, since the number of possible statistics (without equivalent feature queries) is bounded by a fixed constant, and each feature query can be evaluated in polynomial time.

COMMENT 5.3. For CQs with constants, separability is trivial, since the positive examples can be hardcoded into the statistic. In Figure 2, for instance, we could encode each of the first, second, and third tuples of $\mathsf{TxnInfo}$ (and even their join with Card) in a CQ that selects precisely the corresponding transactions. It would, however, be interesting to enforce restrictions on the usage of constants (e.g., limit their number). An elegant way to formalize such restrictions was taken by Grohe and Ritzert [16] that separate the variables into *ordinary variables* and *parameters* that can be set fixed by the learning algorithm. We plan to explore this approach in future work. □

6. ADDITIONAL RELATED WORK

The task of feature engineering has been widely studied for decades [9, 17–20]. Our approach borrows heavily from the feature-engineering process identified in Guyon's seminal book [18] and those we have observed in practice. Feature engineering has received some attention from the database community [2, 3, 23, 29, 35]. That work has made algorithmic or tooling contributions to better support feature engineering, while it has not addressed the fundamental questions that our framework targets.

Frameworks and query languages that fuse logic with probabilistic semantics, to simplify the design of machine-learning models, have been proposed and developed in past decades. Examples of these include Probabilistic Relation Models pioneered by Koller and Friedman [14], PRISM [27], BLOG [25], Markov Logic Networks [26], and the recent *Probabilistic-Programming Datalog* [6]. However, these approaches focus on orthogonal formal questions: the semantics of the models and the complexity of the associated inference tasks. In contrast, we consider the interplay of the logical rules and learning properties. In particular, to the best of our knowledge this work is the first to consider separability, identifiability, and dimensionality in machine-learning models that are defined over database queries. Our formal framework draws inspiration from previous approaches to combining logical reasoning to probabilistic reasoning, which is a classic topic [5, 11], but is distinct in its goal.

There has been a lot of work in the Machine Learning community on learnability aspects of First Order formulas. For

instance, Arias and Khardon [4] considered such aspects (including VC dimension) in the context of Horn clauses, where they establish bounds that are based on syntactic properties of the clauses (e.g., number of variables, literals, clauses, etc.). Similarly, Grohe and Ritzert [16] explored PAC learning of first-order formulas over a "background structure," namely a database. Such setups are quite different from ours, since their goal is to classify a whole interpretation (database) based on a single formula (to be learned), while we consider classification of entities within a single database and focus on *feature engineering* rather than the engineering of the classifier. More technically, in our framework the goal is not necessarily to learn queries, but rather to reason about queries as features of machine-learning models that are not necessarily database queries (e.g., linear models).

7. CONCLUSIONS AND DIRECTIONS

We described a framework for feature engineering towards programming machine-learning solutions over a database, while focusing on the important task of classification. Our framework is based on simple additions to the relational data and query model, where an *entity schema* allows to represent entities along with their associated information, and where feature engineering is the task of designing a *statistic* when given a training instance over the entity schema. This framework enables us to formalize relevant computational problems, conduct nontrivial analyses, and reach insights and solutions. In particular, we have formalized three important computational problems within the framework: separability, identifiability, and VC dimensionality. These problems are parameterized by the hypothesis class in use and the query language deployed for feature extraction.

Focusing on features definable as conjunctive queries and on linear hypotheses, we have drawn connections between the studied computational problems and those of query containment and equivalence. These connections have several interesting consequences. For one, there is a tight relationship between the computational complexity of our problems and that of query containment: it is necessary, and often sufficient, to solve CQ containment in order to solve our problems. Moreover, the fact that identifiability "comes for free" (up to redundancy) gives a formal indication of the suitability of CQs as a language for feature engineering. It also motivates the challenge of finding other natural query languages that are likewise suitable. We conclude with a number of directions and extensions for future research.

Logical Analysis

Further expressiveness. We have focused on the simple class of conjunctive queries for defining statistics, and on the classifier class of linear hypotheses. An immediate future direction would be to consider more expressive classes. For features, these can be unions of conjunctive queries, queries with additional logical operators, non-monotonic features, and aggregate functions. For the hypotheses, future directions can consider any standard class, such as decision trees.

Schema constraints. The complexity of some of the tasks we have considered would be impacted by the presence of schema constraints. In particular, in the identifiability problem column independence would need to be realized by an instance that satisfies the constraints, and not by any instance of the signature. The problem of VC dimensionality would be similarly impacted. We view this direction as an important opportunity of incorporating the database's rich modeling of data into the task of feature engineering.

Text analysis. An area where machine-learning classification is crucial for even simple tasks is that of text analysis, and in particular when the text is in natural language from open domains such as Web and social media [29]. Consequently, we belive that a direction of a high potential impact is that of applying our framework to formalisms that involve queries over text, such as the *document spanners* of Fagin et al. [12, 13] that construct and manipulate relations over text spans (intervals) using extractors (e.g., regular expressions). In particular, the computational challenges will involve queries with both relational and textual operations.

Statistical Questions

Generalized learning tasks. Our features in this work were all Boolean (± 1), and it is desirable to study the natural extension of the framework to numerical features, where numbers are either directly copied from the database or indirectly computed via queries. Numerical values will likely complicate the basic model, as they entail arguing about schema constraints to ensure that a feature query associates a unique value with each entity. Orthogonally, the framework can be generalized to other prediction tasks, such as multi-label classification (e.g., predict the age group of a person) and numerical regression (e.g., predict the actual age of the person). It is important to understand how the challenges we considered are affected by such generalizations.

Separability relaxation. The separability problem, as defined in this article, can be extended by allowing for an *approximate* agreement with the training examples (e.g., the hypothesis h should agree with the labeling λ on at least $(1-\epsilon)$ of the entities, or at most k entities should be misclassified). This is a practical and crucial relaxation in practical scenarios. For one, the training data may be noisy. Moreover, our hypothesis class may be too simple to precisely cover the examples, but can do so with only a small error.

Model complexity. While extending the expressiveness of queries, it is of high importance to find the proper *restrictions* on the engineered statistics (a.k.a. *regularization*), in order to *(a)* reduce the model complexity and, consequently, reduce the risk of overfitting to the training samples, *and (b)* gain more efficient machine-leaning solutions. The common regularization limits the length of the statistic; in our framework, we can consider restrictions on the feature queries, such as size, structure, number of constants/variables, and so on. The ultimate goal is to find settings that properly balance between overfitting, underfitting, inference (classification) complexity and learning (training) complexity.

The vast literature on machine learning gives rise to many more directions for our framework to extend, such as notions of capacity beyond VC dimension (e.g., *Rademacher* and *Gaussian* complexities [8]) and the implications of the "transductive" learning environments, where we know to begin with what entities we will need to predict upon [15]. We believe that our framework can contribute to many of these directions the important angle of data and query modelling.

Acknowledgments

The authors are grateful to Stephen Bach and Alex Ratner for insightful input on this work, and for Jared Alexander Dunnmon for valuable suggestions on this article.

8. REFERENCES

[1] *SAS Report on Analytics*. `sas.com/reg/wp/corp/23876`.

[2] M. Anderson, D. Antenucci, V. Bittorf, M. Burgess, M. Cafarella, A. Kumar, F. Niu, Y. Park, C. Ré, and C. Zhang. Brainwash: A Data System for Feature Engineering. In *CIDR*, 2013.

[3] M. R. Anderson, M. J. Cafarella, Y. Jiang, G. Wang, and B. Zhang. An integrated development environment for faster feature engineering. *PVLDB*, 7(13):1657–1660, 2014.

[4] M. Arias and R. Khardon. Complexity parameters for first order classes. *Machine Learning*, 64(1-3):121–144, 2006.

[5] F. Bacchus, A. J. Grove, J. Y. Halpern, and D. Koller. From statistical knowledge bases to degrees of belief. *CoRR*, cs.AI/0307056, 2003.

[6] V. Bárány, B. ten Cate, B. Kimelfeld, D. Olteanu, and Z. Vagena. Declarative probabilistic programming with datalog. In *ICDT*, volume 48 of *LIPIcs*, pages 7:1–7:19. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2016.

[7] P. Barceló and M. Romero. The complexity of reverse engineering problems for conjunctive queries. In *ICDT*, volume 68 of *LIPIcs*, pages 7:1–7:17. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2017.

[8] P. L. Bartlett and S. Mendelson. Rademacher and gaussian complexities: Risk bounds and structural results. In *COLT*, pages 224–240, 2001.

[9] D. E. Boyce. *Optimal Subset Selection: Multiple Regression, Interdependence, and Optimal Network Algorithms* . Springer-Verlag, 1974.

[10] C. Dwork, M. Hardt, T. Pitassi, O. Reingold, and R. S. Zemel. Fairness through awareness. In *ITCS*, pages 214–226. ACM, 2012.

[11] R. Fagin, J. Y. Halpern, and N. Megiddo. A logic for reasoning about probabilities. *Inf. Comput.*, 87(1/2):78–128, 1990.

[12] R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Spanners: a formal framework for information extraction. In *PODS*, pages 37–48, 2013.

[13] R. Fagin, B. Kimelfeld, F. Reiss, and S. Vansummeren. Document spanners: A formal approach to information extraction. *J. ACM*, 62(2):12, 2015.

[14] N. Friedman, L. Getoor, D. Koller, and A. Pfeffer. Learning probabilistic relational models. In *IJCAI*, pages 1300–1309, 1999.

[15] A. Gammerman, K. S. Azoury, and V. Vapnik. Learning by transduction. In *UAI*, pages 148–155. Morgan Kaufmann, 1998.

[16] M. Grohe and M. Ritzert. Learning first-order definable concepts over structures of small degree. In *LICS*, pages 1–12. IEEE Computer Society, 2017.

[17] I. Guyon and A. Elisseeff. An introduction to variable and feature selection. *Journal of Machine Learning Research*, 3:1157–1182, 2003.

[18] I. Guyon, S. Gunn, M. Nikravesh, and L. A. Zadeh. *Feature Extraction: Foundations and Applications (Studies in Fuzziness and Soft Computing)*. Springer-Verlag New York, Inc., 2006.

[19] T. Hastie, R. Tibshirani, and J. Friedman. *The Elements of Statistical Learning: Data mining, inference, and prediction*. Springer, 2001.

[20] G. H. John, R. Kohavi, and K. Pfleger. Irrelevant features and the subset selection problem. In *Machine Learning*, pages 121–129, 1994.

[21] S. Kandel, A. Paepcke, J. M. Hellerstein, and J. Heer. Enterprise data analysis and visualization: An interview study. *IEEE Trans. Vis. Comput. Graph.*, 18(12):2917–2926, 2012.

[22] B. Kimelfeld and C. Ré. A relational framework for classifier engineering. In *PODS*, pages 5–20. ACM, 2017.

[23] A. Kumar, J. F. Naughton, J. M. Patel, and X. Zhu. To join or not to join?: Thinking twice about joins before feature selection. In *SIGMOD Conference*, pages 19–34. ACM, 2016.

[24] E. L. Lehmann and G. Casella. *Theory of point estimation*, volume 31. Springer, 1998.

[25] B. Milch, B. Marthi, S. J. Russell, D. Sontag, D. L. Ong, and A. Kolobov. Blog: Probabilistic models with unknown objects. In *IJCAI*, pages 1352–1359, 2005.

[26] M. Richardson and P. Domingos. Markov logic networks. *Mach. Learn.*, 62(1-2):107–136, 2006.

[27] T. Sato and Y. Kameya. PRISM: A language for symbolic-statistical modeling. In *IJCAI*, pages 1330–1339, 1997.

[28] S. Shalev-Shwartz and S. Ben-David. *Understanding Machine Learning: From Theory to Algorithms*. Cambridge University Press, 2014.

[29] J. Shin, S. Wu, F. Wang, C. D. Sa, C. Zhang, and C. Ré. Incremental knowledge base construction using DeepDive. *PVLDB*, 8(11):1310–1321, 2015.

[30] B. ten Cate and V. Dalmau. The product homomorphism problem and applications. In *ICDT*, volume 31 of *LIPIcs*, pages 161–176. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2015.

[31] V. N. Vapnik and A. Y. Chervonenkis. On the uniform convergence of relative frequencies of events to their probabilities. *Theory of Probability and its Applications*, 16(2):264–280, 1971.

[32] M. J. Wainwright and M. I. Jordan. Graphical models, exponential families, and variational inference. *Foundations and Trends in Machine Learning*, 1(1-2):1–305, 2008.

[33] R. Willard. Testing expressibility is hard. In *CP*, volume 6308 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 2010.

[34] R. S. Zemel, Y. Wu, K. Swersky, T. Pitassi, and C. Dwork. Learning fair representations. In *ICML*, volume 28 of *JMLR Proceedings*, pages 325–333. JMLR.org, 2013.

[35] C. Zhang, A. Kumar, and C. Ré. Materialization optimizations for feature selection workloads. In *SIGMOD Conference*, pages 265–276, 2014.

[36] I. Zliobaite. A survey on measuring indirect discrimination in machine learning. *CoRR*, abs/1511.00148, 2015.