A Database Framework for Classifier Engineering

Benny Kimelfeld[1] and Christopher Ré[2]

[1] LogicBlox, Inc. and Technion, Israel
[2] Stanford University

1 Introduction

In the design of machine-learning solutions, a critical and often the most resourceful task is that of feature engineering [7, 4], for which recipes and tooling have been developed [3, 7]. In this vision paper we embark on the establishment of database foundations for feature engineering. We propose a formal framework for classification, in the context of a relational database, towards investigating the application of database and knowledge management to assist with the task of feature engineering. We demonstrate the usefulness of this framework by formally defining two key algorithmic challenges within: (1) *separability* refers to determining the existence of feature queries that agree with the given training examples, and (2) *identifiability* is the task of testing for the property of independence among features (given as queries). Moreover, we give preliminary results on these challenges, in the context of conjunctive queries. We focus here on boolean features that are represented as ordinary database queries, and view this work as the basis of various future extensions such as numerical features and more general regression tasks.

2 Formal Framework

We first present our formal framework for classification with binary features within a relational database.

2.1 Classifiers and Learning

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

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

where n is a natural number that we call the *arity* of γ. A *classifier class* is a (possibly infinite) family Γ of classifiers. We denote by Γ_n the restriction of Γ to the n-ary classifiers in Γ. 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 set of all n-ary training collections. A *cost function* for a classifier class Γ is a function of the form

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

where $\mathbb{R}_{\geq 0}$ is the set of nonnegative numbers. In the context of a classifier class $\mathit{\Gamma}$ and a cost function c, *learning* a classifier is the task of finding a classifier $\gamma \in \mathit{\Gamma}_n$ that minimizes $c(\gamma, T)$, given a training collection $T \in \mathbf{T}_n$.

We illustrate the above definitions on the important class of *linear classifiers*. An n-ary linear classifier is parameterized by a vector $\mathbf{w} \in \mathbb{R}^n$, is denoted by $\mathit{\Lambda}_{\mathbf{w}}$, and is defined as follows for all $\mathbf{a} \in \{-1, 1\}^n$.

$$\lambda_{\mathbf{w}}(\mathbf{a}) \overset{\text{def}}{=} \begin{cases} 1 & \text{if } \mathbf{a} \cdot \mathbf{w} \geq 0; \\ -1 & \text{otherwise.} \end{cases}$$

where "$\cdot$" 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 lsq that is given by

$$\mathit{lsq}(\mathit{\Lambda}_{\mathbf{w}}, T) \overset{\text{def}}{=} \sum_{\langle \mathbf{x}, y \rangle \in T} (\mathbf{x} \cdot \mathbf{w} - y)^2$$

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

2.2 Relational Formalism

Our relational terminology is as follows. A *schema* is a pair $(\mathcal{A}, \mathit{\Sigma})$, where $\mathcal{A}$ is a *signature* that consists of *relation symbols*, and $\mathit{\Sigma}$ is a set of logical integrity constraints over $\mathcal{A}$. Each relation symbol R has an associated arity. We assume an infinite set Const of *constants*. An *instance* I over a schema $\mathbf{S} = (\mathcal{A}, \mathit{\Sigma})$ associates with every k-ary relation symbol $R \in \mathcal{A}$ a finite subset R^I of Const^k, such that all the constraints of $\mathit{\Sigma}$ are satisfied. The *active domain* of an instance I, denoted $\mathit{adom}(I)$, is the set of all the constants in Const that are mentioned in I.

Let $\mathbf{S}$ be schema. A *query* (*over* $\mathbf{S}$) is a function Q that is associated with an arity k, and that maps every relation instance I over $\mathbf{S}$ into a finite subset $Q(I)$ of Const^k. A query Q' *contains* a query 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*. A query Q is *additive* if for every two instances I_1 and I_2, if $\mathit{adom}(I_1)$ and $\mathit{adom}(I_2)$ are disjoint, then

$$Q(I_1 \cup I_2) = Q(I_1) \cup Q(I_2)\,.$$

A *query class* is a mapping that associates with every schema $\mathbf{S}$ a class of queries over $\mathbf{S}$. An example of a query class is that of the *conjunctive queries*. A conjunctive query (CQ) is represented by the logical formula $q(\mathbf{x})$ that has the form

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

where $\mathbf{x}$ and $\mathbf{y}$ are disjoint sequences of variables, $\mathbf{d}$ is a sequence of constants, 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). The result of applying the CQ $Q = q(\mathbf{x})$ to the instance I consists of all the tuples $\mathbf{a}$ (of the same length as $\mathbf{x}$) such that $q(\mathbf{a})$ is true in I; we denote this result is denoted by $Q(I)$.

2.3 Classification Framework

We now present our formal framework. An *entity schema* is a triple $(\mathcal{A}, \Sigma, E)$, where $(\mathcal{A}, \Sigma)$ is a schema and E is a relation symbol in $\mathcal{A}$ that represents the *entities* (as tuples). An *instance* I over a schema $(\mathcal{A}, \Sigma, E)$ is simply an instance over $(\mathcal{A}, \Sigma)$. Intuitively, an instance I over an entity schema $(\mathcal{A}, \Sigma, E)$ represents a set of entities, namely E^I (i.e., the set of tuples in E), along with information about the entities that is contained in the remaining relations R^I. For example, E may be the relation $\mathsf{Persons}$ and $\mathcal{A}$ may include, besides $\mathsf{Persons}$, relations such as $\mathsf{PersonAddress}$, $\mathsf{PersonCompany}$, $\mathsf{CompanyCity}$, and so on. If $\mathbf{S} = (\mathcal{A}, \Sigma, E)$ is an entity schema, then the elements $\mathcal{A}$, Σ and E are denoted by $\mathcal{A}_\mathbf{S}$, $\Sigma_\mathbf{S}$ and $E_\mathbf{S}$, respectively.

Let $\mathbf{S}$ be an entity schema, and let I be an instance over $\mathbf{S}$. A *feature query* (*over* $\mathbf{S}$) is a query π over the schema $\mathbf{S}$, such that $\pi \subseteq E_\mathbf{S}$, where $E_\mathbf{S}$ is viewed as the query that, given an instance I, copies the relation $E^I_\mathbf{S}$. In other words, a feature query is a query that selects entities. For example, if E is $\mathsf{Persons}(ssn, name)$ then a feature can be the following CQ $q(s, n)$ (selecting persons working in New York City).

$$\exists_c \big[\mathsf{Persons}(s, n) \land \mathsf{PersonCompany}(s, c) \land \mathsf{CompanyCity}(c, \text{`NYC'})\big]$$

If π is a feature query, then π^I denotes the function $f : E^I_\mathbf{S} \to \{-1, 1\}$ where

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

A *statistic* (over $\mathbf{S}$) is a sequence $\Pi = (\pi_1, \dots, \pi_n)$ of feature queries. We denote by Π^I the function $(\pi^I_1, \dots, \pi^I_n)$ from $E^I_\mathbf{S}$ to $\{-1, 1\}^n$.

A *feature schema* is a pair $(\mathbf{S}, \Pi)$, where $\mathbf{S}$ is an entity schema, and Π is a statistic over $\mathbf{S}$ that produces a sequence of features for every entity of a given input instance. We say that Π and $(\mathbf{S}, \Pi)$ are in a query class $\mathbf{Q}$ if every query in Π belongs to $\mathbf{Q}$. A *training instance* over $\mathbf{S}$ is a pair (I, o), where I is an instance over $\mathbf{S}$ and $o : E^I_\mathbf{S} \to \{-1, 1\}$ is a function that partitions the entities into positive and negative examples. Given a feature schema $(\mathbf{S}, \Pi)$ and a classifier class Γ, the training instance (I, o) defines the training collection that consists of the tuple $\langle \Pi^I(e), o(e) \rangle$ for every $e \in E^I_\mathbf{S}$.

3 Feature Engineering

In *feature engineering*, one devises feature queries for a classification task. Next, we discuss two computational challenges that naturally arise in feature engineering.

3.1 Separability

Let $(\mathbf{S}, \Pi)$ be a feature schema, and let Γ be a classifier class. A training instance (I, o) is said to be *Γ-separable* with respect to (w.r.t.) Π if there exists a classifier $\gamma \in \Gamma$ that fully agrees with o; that is, γ and Π have the same arity, and $\gamma(e) = o(e)$ for every $e \in E^I_\mathbf{S}$. We define the following core computational problem.

Problem 1 (Separability). Let $\mathbf{S}$ be an entity schema, let $\mathbf{Q}$ be a query class over $\mathbf{S}$, and let Γ be a classifier class. The *separability problem* is the following. Given a training instance (I, o) over $\mathbf{S}$, determine whether there exists a statistic Π in $\mathbf{Q}$ such that (I, o) is Γ-separable w.r.t. Π.

The separability problem, as defined, can be extended in various practical directions. The input can include a bound N on the length n of the statistic Π (hence, limiting the *model complexity*, which results in classifiers that are more efficient and less overfitting). One can allow for an *approximate* agreement with o (e.g., the classifier should agree with o on at least $(1 - \epsilon)$ of the entities, or at most k examples should be misclassified). And one can impose various constraints on common query classes $\mathbf{Q}$ (e.g., limit the size of queries, number of constants, etc., again to limit the model complexity and potential overfitting). The following theorem considers the complexity of testing for separability in the case where the class of queries is that of CQs without constants,[3] which we denote by $\mathsf{CQ}^{\mathsf{nc}}$. It states that, in the absence of such extensions of the problem, it can very quickly get intractable.

Theorem 1. *Let* $\mathbf{Q}$ *be the class* $\mathsf{CQ}^{\mathsf{nc}}$*, and let* Γ *be the class* Lin*. For every entity schema* $\mathbf{S}$*, separability is in NP. Moreover, there exists an entity schema* $\mathbf{S}$ *such that separability is NP-complete.*

The proof of membership in NP is using the concept of a *canonical database* [1], and the proof of NP-hardness is by a reduction from the maximum-clique problem.

We note that a problem similar to separability has been studied in a recent paper by Cohen and Weiss [2], where data are labeled graphs and features are tree patterns.

3.2 Statistic Identifiability

We denote by $\mathbf{0}^m$ the vector of m zeroes. Let M be an $n \times k$ real matrix. A *linear column dependence* in M is a weight vector $\mathbf{w} \in \mathbb{R}^k$ such that $\mathbf{w} \neq \mathbf{0}^k$ and $M \cdot \mathbf{w} = \mathbf{0}^n$; if M does not have any linear column dependence, then we say that M is *linearly column independent*. Let $(\mathbf{S}, \Pi)$ be a feature schema, and let I be an instance of $\mathbf{S}$. We fix an arbitrary order over the entities in $E^I_{\mathbf{S}}$, and denote by $[\![\Pi^I]\!]$ the matrix that consists of the rows $\Pi^I(e)$ for every $e \in E^I_{\mathbf{S}}$ in order. The second computational problem we define is the following.

Problem 2 (Identifiability). Let $\mathbf{Q}$ be a query class. *Identifiability* is the problem of testing, given a feature schema $(\mathbf{S}, \Pi)$ in $\mathbf{Q}$, whether there exists an instance I over $\mathbf{S}$ such that the matrix $[\![\Pi^I]\!]$ is linearly column independent; in that case, we say that Π is *identifiable*.

Identifiability is an important property in the design of machine-learning solutions [5]. Particularly, in the case of the classifier class Lin and the cost function lsq, this property implies that there is a single optimal classifier, whereas its absence implies that the space of optimal solutions is unbounded.

[3] For CQs with constants, the problem is trivial and not interesting, since the positive examples can be hardcoded into the statistic.

Next, we show that in the case of CQs. identifiability amounts to query equivalence. A statistic Π is said to have *redundancy* if it contains two distinct feature queries that are equivalent.

Theorem 2. *Let* $\mathbf{Q}$ *be the query class of additive CQs, and let* $(\mathbf{S}, \Pi)$ *be a feature schema such that* Π *is in* $\mathbf{Q}$*. Then* Π *is identifiable if and only if* Π *has no redundancy.*

We conclude with comments on Theorem 2. First, this theorem is proved again by applying the concept of a canonical database of a CQ. Second, we can extend the theorem to the class of all CQs, but the condition that characterizes identifiability is significantly more complicated (and will be given in the extended version of this paper). Third, this theorem generalizes to *affine independence*, which is important in different cost functions such as *maximum entropy* [6]. Finally, by an immediate application of the NP-completeness of CQ containment [1] we get that identifiability is NP-complete in the case of additive CQs.

Acknowledgments

Benny Kimelfeld is a Taub Fellow, supported by the Taub Foundation. The research of Christopher Ré is supported by DARPA's projects XDATA (FA8750-12-2-0335), DEFT (FA8750-13-2-0039), MEMEX, and SIMPLEX. His research is also supported by NSF Career Award IIS-1353606, ONR Awards N000141210041 and N000141310129, NIH Grant U54EB020405 (awarded by NIBIB through funds provided by the trans-NIH BD2K initiative), the Sloan Research Fellowship, the Moore Foundation, American Family Insurance, Google, and Toshiba. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the authors and do not necessarily reflect the views of DARPA, AFRL, NSF, ONR, NIH, or the U.S. government.

References

1. A. K. Chandra and P. M. Merlin. Optimal implementation of conjunctive queries in relational data bases. In J. E. Hopcroft, E. P. Friedman, and M. A. Harrison, editors, *STOC*, pages 77–90. ACM, 1977.
2. S. Cohen and Y. Y. Weiss. Learning Tree Patterns from Example Graphs. In M. Arenas and M. Ugarte, editors, *18th International Conference on Database Theory (ICDT 2015)*, volume 31 of *Leibniz International Proceedings in Informatics (LIPIcs)*, pages 127–143, Dagstuhl, Germany, 2015. Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik.
3. 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., Secaucus, NJ, USA, 2006.
4. 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.
5. E. L. Lehmann and G. Casella. *Theory of point estimation*, volume 31. Springer, 1998.
6. 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.
7. C. Zhang, A. Kumar, and C. R. Materialization optimizations for feature selection workloads. In *SIGMOD Conference*, pages 265–276, 2014.