ON COLUMN HETEROGENEITY

BING TIAN DAI

B.Sc. (Hons), NUS

A THESIS SUBMITTED FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF COMPUTER SCIENCE

NATIONAL UNIVERSITY OF SINGAPORE

2011

NUS
National University
of Singapore

School *of* Computing

Dedication

This thesis is dedicated to my parents, who have been supporting me in every way they can all these years. It is with regret that I was unable to complete my Ph.D. earlier for my father to witness the convocation. Despite that, I wish that he will smile at me, knowing that I have managed to complete my Ph.D.

Acknowledgment

I would like to express my thanks to my supervisor, Professor Anthony Tung Kum Hoe, who introduced me into the world of research. His guidance through these years has lit up my bumpy road of pursuing this degree. He taught me knowledge and action must go hand in hand–it is not enough to just know something; it is more important to take the action after knowing it. Without his help, I would not have completed my study.

I would also like to thank Professor Ooi Beng Chin and Professor Tay Yong Chiang, who provided me lots of opportunities. Prof. Ooi has taught me to stay vigilant and to always reflect on my own behaviors. Prof. Tay has been an example to me since my undergraduate days and I'm aspired by his passion and diligentness in both teaching and research.

I also appreciate Divesh Srivastava, Suresh Venkatasubramanian and Nick Koudas for their guidance during and after my internships.

My last thanks go to my lab-mates in the Database E-Commerce lab, their encouragement and assistance have made my life happier and more interesting.

Abstract

Database columns are the very basic units in databases, and they determine how databases are designed, as well as how database queries are processed. In this thesis, we study one particular property about database columns–the column heterogeneity. Our study consists of three parts. The first part is on the intra-column heterogeneity, i.e., the heterogeneity within a database column. The measure of column heterogeneity is defined based on the syntactic types of the values in a column. The subsequent two parts then discuss the inter-column heterogeneity, i.e., the heterogeneity when multiple database columns are taken into consideration. We propose validating schema matching, which is to prevent database columns from becoming more heterogeneous. The schema matching validator includes a measure of integratability, a procedure of extracting sub-string matches, and an invalidation certificate as evidence of two columns being not integratable. The last part of this thesis focuses on the problem that arises from the data management of emerging community databases. An out-of-the-box approach that separates users' actions from database operations is proposed. This approach evaluates the heterogeneity across different database columns, and thus makes semantic influences among those columns. With these, this thesis shows how to deal with column heterogeneity in databases.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Columns in Relational Databases

Since Relational Database Management System (RDBMS) was introduced by Codd [Cod70], *database column* has become a fundamental concept in RDBMS. Codd defined **database columns** in the following way:

> Given sets $S_1, S_2, \cdots, S_n$ (not necessarily distinct), R is a relation on these n sets if it is a set of n-tuples each of which has its first element from S_1, its second element from S_2, and so on. More concisely, R is a subset of the Cartesian product $S_1 \times S_2 \times \cdots \times S_n$. We shall refer to S_j as the j^{th} *domain* of R.
>
> - The ordering of columns is significant–it corresponds to the ordering $S_1, S_2, \cdots, S_n$ of the domains on which R is defined.
> - The significance of each column is partially conveyed by labeling it with the name of the corresponding domain.

A tuple is thus an ordered list of elements from columns in a relation. Such elements are often referred to as *data values* or simply *values*. Since

data values at the same position of all tuples are from the same domain, each column in relational model can be viewed as a set of values and is labeled by the name of the corresponding domain.

Columns form the basis of a *relational schema*, which is a formal language describing tables in a database and relationships among such tables. On the other hand, *Structured Query Language (SQL)*, which is the query language for relational databases, is also defined on top of database columns, for creating, querying and manipulating relational databases. Therefore, database columns are considered as pillars in a modern RDBMS.

In recent years, column-oriented storage models are drawing special attentions from the database community. Due to the slow increase in disk bandwidth, a column-oriented DMBS [SAB+05] is able to avoid reading values from irrelevant columns to the main memory, and to only read values from columns that are required by a query. This has again emphasized the importance of database columns from a storage model perspective.

1.2 Heterogeneous Columns

As database columns play such a significant role in RDBMS, the heterogeneity of columns in a database can severely affect the quality of the database. By Codd's definition [Cod70], values at the same position of all tuples are expected to come from the same domain, and thus these values are also expected to have the same type. We use the following example to elaborate.

Example 1.1. *Column* `year` *consists of values which can be interpreted as years, like* `1999` *and* `2012`*. Values like* `2010.3` *and* `yyyy.mm` *are not expected as they are of different types.*

We say a column is **homogeneous** if its values are of the same type,

otherwise, it is called **heterogeneous**.

From the database design point of view, having homogeneous columns is better than having heterogeneous column. Database efficiencies are largely dependent on the indexes, which are usually built on one or a few columns. Suppose a database designer wishes to build an index for a heterogeneous column, he would have to choose an index that caters all types of values in this column. However, if the column is homogeneous, the database designer can choose an index which is optimized for values for this particular type, thus improve the efficiency. Homogeneous columns are therefore more favorable to database designers and database administrators.

From a database user's perspective, homogeneous columns are also better. In order to perform database queries, one has to understand what are the columns and what each column is about, and then write the database query accordingly. With heterogeneous columns, it is certainly more difficult to understand what the columns are about, and the query result is also harder to decipher. For instance, one may check the first few tuples in a column and obtained an idea about the column content. But she may get some "surprising" values of other types, as she thought the column is homogeneous and its values are of the same type as the first few values she saw. Thus database users also prefer homogeneous columns over heterogeneous columns.

Although database columns are designed with the favor of homogeneity, columns can still gain a significant amount of heterogeneity as database evolves. This is particularly true when there is no database administrator around, e.g., a community-based database system. Everyone has the privilege to access the database and modify the database, thus columns become more and more heterogeneous. Even for databases with administrators, the growth of heterogeneity happens too, like in the following example.

Example 1.2. *Imagine there is a customer database designed in last century, which records their home addresses in the column* `address` *and the phone numbers in the column* `contact`*. With more and more people having an email address as their preferred way of being contacted, they may just leave only their email addresses as a contact. Now the database administrator has to decide which column the email addresses should be put in as there was no column designed for email addresses at the time the database was created. Fortunately, this is an easy choice to make, the heterogeneity of that column will increase, no matter whether he puts email addresses in the column* `address` *or in the column* `contact`*. This is because the type of email addresses is different from either the type of home addresses or the type of the phone numbers.*

1.3 Column Heterogeneity and Data Quality

From the above examples, we understand that heterogeneous columns altered the intended way of database design, which may also lead to subsequent degradation of data quality. We give the following example.

Example 1.3. *Assume the database administrator in Example 1.2 puts email addresses in column* `contact`*, the column of phone numbers. Another database administrator is going to perform a database operation that inserts the country code in front of the phone numbers as the database is going to include international customers. If she does not know the column* `contact` *also contains email addresses as described in Example 1.2, inserting the country code to every value in the column* `contact` *will immediately make email addresses invalid.*

Data quality is a serious concern in every data management application,

as poor data quality can severely degrade common business practices. Industry consultants often quantify the adverse impact of poor data quality in the billions of dollars annually.

Data quality issues have been studied quite extensively in the literature [DJ03]. In particular, a variety of quality measures have been proposed, e.g., accuracy, freshness and completeness, to capture common sources of data quality degradation [Wid05]. Data profiling tools like Bellman [DJMS02] compute concise summaries of the values in database columns, to identify various errors introduced by poor database design. These include approximate keys (the presence of null values and defaults in a column may result in the approximation), and approximate functional dependencies in a table (possibly due to inconsistent values).

In order to improve data quality, the heterogeneous columns in a database are to be identified first. As columns are to be compared on their degree of heterogeneity, it is not adequate to flag one column as a heterogeneous column. Instead, a real number should be given to a column to tell how heterogeneous this column is. We therefore propose a measure, called *column heterogeneity* to measure the heterogeneity within a column. This measure makes columns comparable on the scale of column heterogeneity, and helps discover the most heterogeneous column in a database, on which the database administrator will take further actions, e.g., horizontal split that column into a few columns to make every part from the split homogeneous.

However, the column heterogeneity needs to satisfy certain criteria to correctly capture the homogeneity within a column. This will be discussed in detail in Chapter 4.

1.4 Column Heterogeneity and Data Integration

It is not enough to just identify heterogeneous columns and do data cleansing on such columns; it is in fact more important to prevent database columns from getting more heterogeneous, as prevention is the best medicine.

As discussed in Example 1.2, the heterogeneity is introduced when values from new types are inserted into a database which was not designed for such types. It becomes more serious when multiple databases are to be integrated, as any unforeseen mismatch between different types of values will result in the growth of heterogeneity.

Example 1.4. *Consider column* `contact` *in one table consisting of a large number of phone numbers and a small number of email addresses, while column* `contact` *from another table is composed by a major portion of phone numbers and a minor portion of home addresses. If the database administrator who is assigned with the task of integrating the two databases into one did not know that the two* `contact` *columns are of different types of values, and thought both of them are just phone numbers, she would have matched these two columns and merged values under the same column in the integrated database. By doing so, the heterogeneity of the new column* `contact` *is higher than the heterogeneity of either* `contact` *column from the two original database, which is not desired.*

Hence, there ought to be a procedure that checks whether the two columns to be integrated will become a more heterogeneous column if integrated together. As the basic operator which associates a column from one database to a column from another database is usually referred to as *column match*, we call the above procedure that performs the checking on the heterogeneity

as *column matching validation*, which declares if a candidate column match is valid or not. This column matching validation is essential to any schema matching method since it denies any candidate column match that degrades the quality of the database.

Note that there is no requirement on the homogeneity of the candidate columns, i.e., the columns to be integrated are not necessarily homogeneous. Even if the columns are heterogeneous, as long as the heterogeneity does not increase after integration, it is considered as a valid match. In the above example, if both columns are mixtures of phone numbers and email addresses with the same distribution, the heterogeneity will remain the same if the two columns are integrated into one. Therefore, the match between the two original columns is valid.

In fact, it is also possible to have an integrated column with reduced heterogeneity. The column match validation should announce valid for such scenarios.

We will re-visit this problem of multi-column matching validation in Chapter 5.

1.5 Column Heterogeneity and Data Semantics

For the above two problems described in Sections 1.3 and 1.4, the heterogeneity is something to be avoided, i.e., we favor columns with less heterogeneity, or a database integration is considered valid only if the heterogeneity does not increase. In this section, we introduce another interesting problem which does not prefer homogeneous or heterogeneous columns, and deems heterogeneous columns as informative as homogeneous columns.

Conventionally, only users with at least some database knowledge perform database queries, as in order to query a database, a user has to understand the database in the first place, e.g., what columns to query, and how the query should be formalized or optimized.

However, as database technologies advance, database user community has become larger and larger. Even for people who do not have any database knowledge, they are also able to search information on-line, without knowing they are actually querying some database in the world. It is therefore desired to reduce the barrier of utilizing databases to allow common users to do other kinds of database manipulation apart from querying. For example, if users are able to insert tuples into databases, they will contribute a large community database, which is valuable for studying community problems.

A potential danger of allowing community users to modify database is that their actions may alter the original database structure, i.e., its schemas. Database schemas are the blueprints on how databases are constructed and operated; it is usually forbidden to mess up with schemas. Making changes to schema needs data experts to translate from tuples in the old schemas to tuples in the new schemas. Community users hardly know or care about what schema is, and they should not be expected to take only "right" actions so that the schema is well maintained.

Therefore, we want to achieve both of the following:

- a well-formed database that follows certain schemas; and
- a free-and-easy way to enable community users to access and manipulate the database

This sounds contradicting, but it is this third problem that we target at: providing a platform for community users to access, manipulate and

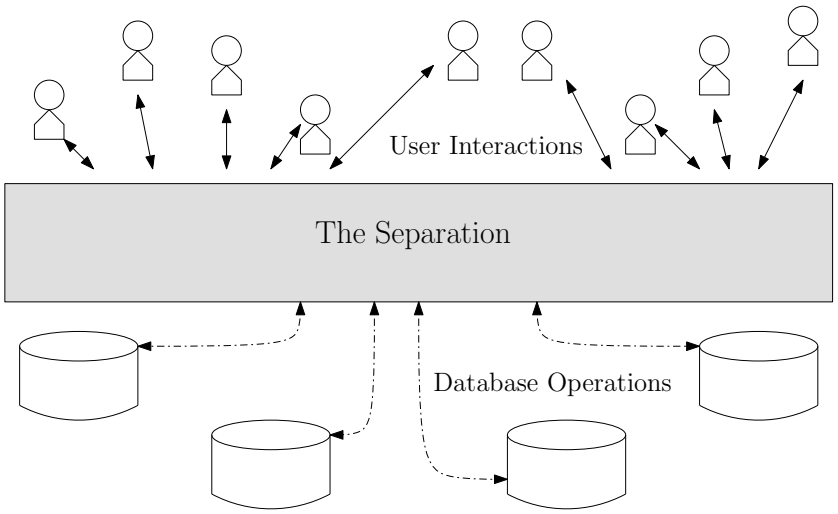


Figure 1.1: Separation between users and databases

contribute to a database with schemas properly maintained. Let us assume that users have a very generic way of accessing and manipulating the database, i.e., pairing the semantics and the values. For example, if they were to query for John's phone number, they would submit a query like "`query (name = 'John', phone = ?)`"; and if they wish to submit a tuple, they would probably write in this way "`insert (name = 'John', phone = '123-456-7890')`".

This is a basic and intuitive way of dealing with a database, without knowing the details of the database. However, from the database management point of view, this simple way of accessing databases is not acceptable, as it may corrupt the schemas. Therefore, it should not be allowed to access the databases directly. Instead, there ought to be a separation that serves as a proxy to the community users, i.e., it interprets users' request, and carry out the "right" database operations, so that the database schemas are well protected.

As shown in Figure 1.1, the separation now blocks users directly accessing the database. This separation interprets users' interactions from above, and translate to database commands below, to operate the databases. Similarly, if there are results from different databases, the separation also need to combine them and then present to the user who issued the query.

Although the platform provided to users is easy and intuitive, the challenges are now transferred to the separation, which is understanding the users' intentions from their semantics. The following example demonstrates why this separation takes semantics into consideration.

Example 1.5. *In a typical customer database, there are two columns associated with phone numbers, one is called* `phone`*, and the other is called* `mobile`*, since the original database designer intends to put customers' land numbers in the column* `phone`*, and to put customers' mobile numbers in the column* `mobile`*. However, this is unknown to the community users, and when they issue queries like "*`query (name = 'John', phone = ?)`*", they just want to find a number to call, no matter it is a land line or a mobile line. If this separation only looks at the column* `phone`*, there will be no result if John's tuple is only associated with a mobile number, but not a land number. Therefore, this separation needs to understand that the query issuer's intention is to search through both column* `phone` *and* `mobile`*.*

Given that database columns are potentially heterogeneous, it is not straightforward to associate such columns with the semantics. In Chapter 6, we will solve this problem of establishing relationship between semantics and heterogeneous columns.

1.6 Thesis Organization

This thesis is organized as follows. We will give a brief literature review in Chapter 2. Some basic but important concepts in *information theory* will then be introduced in Chapter 3. This chapter will also present the *information bottleneck method*, which is the basis of our clustering method used throughout the thesis. The following three chapters, Chapters 4, 5 and 6 will address the three problems proposed in Sections 1.3, 1.4 and 1.5 respectively. Subsequently, we will conclude in Chapter 7.

This thesis is written based on the three conference papers [DKO+06, DKS+08, LADT11] that have been published before, corresponding to each of the three technical chapters, Chapter 4 [DKO+06], Chapter 5 [DKS+08] and Chapter 6 [LADT11].

Chapter 2

Literature Review

We now give an introductory review on relevant literature related to column heterogeneity. This literature review consists of three parts. The first part talks about data quality, and supports the motivation for studying the column heterogeneity measure in Chapter 4. We then review techniques on schema matching and mapping, to explain the necessity of having a validator for multi-column schema matching in Chapter 5. The last part of the review is on a few novel database models, while we will see in Chapter 6, how we can combine our column heterogeneity and such database model to manage data from the semantics perspective.

2.1 Data Quality

Data quality issues have been studied quite extensively in the literature [DJ03, JD03, BCS04]. A variety of quality metrics have been proposed, e.g., accuracy, freshness and completeness, to associate with and query along with the data [MRV99, Wid05]. Mihaila *et al.* [MRV99] associate quality parameters with data, and extend SQL to control data retrieval based on the values of

these parameters. Widom [Wid05] proposed the *Trio* model for integrated management of data, accuracy, and lineage, focusing on data model (TDM) and query language (TriQL) issues.

When fields have poor quality data, record linkage techniques using approximate match predicates are fundamental [KS05]. These techniques return pairs of tuples from the tables. Each pair tagged with a score, signifying the degree of similarity between the tuples in the pair according to the specific approximate match predicate. Such approximate join operations have received much research attention in recent years, due to their significance and practical importance [CGGM03, KMS04].

Data profiling tools like Bellman [DJMS02] collect concise summaries of the values of the database fields. These summaries (set and multiset signatures based on min hash sampling and min hash counts) allow Bellman to determine data quality problems like (i) fields that are only approximate keys (the presence of null values and defaults may result in the approximation), (ii) approximate functional dependencies in a table (possibly due to inconsistent values), and (iii) approximate joinable keys/foreign keys. However, such profiling tools do not currently help with our heterogeneity problem. The desiderata of our heterogeneity problem will be further discussed in Chapter 4.

2.2 Schema Matching and Mapping

Schema matching and schema mapping have been studied extensively. The survey by Rahm and Bernstein [RB01] lays out a general ontology of approaches, classified by the level of granularity (schema-level, data-level, the kinds of rules generated (1-1, 1-m), and other factors. A later survey by Doan

and Halevy [DH05a] covers more recent work, as well as related research in AI and machine learning.

To the best of our knowledge, there is no prior work that formulates the idea of matching schema based on inferred syntactic types from string data, although the paper by Doan, Domingos and Halevy [DDH01] does hint at this in their discussion of possible base learners. The work by Embley *et al.* [EXD04] uses data frames and domain ontologies to match schema; these notions together instantiate a syntactic type, but are constructed from domain knowledge, rather than inferred from the data. There are however a number of works that use statistical and/or machine learning methods to learn properties of an attribute from data and examples. The SEMINT algorithm of Li and Clifton [LC00] builds discriminant vectors from numerical data and trains a neural network to learn attributes from data. SEMINT maps strings to numerical attributes by computing statistics on string length. Berlin and Motro [BM01, BM02] use example data and conditional probability to weigh matches between individual attributes, and use a global matching algorithm to find a schema matching. Kang and Naughton [KN03] employ a similar idea, using the mutual information of two columns as a measure of how likely they are to be matched to each other. In their scheme, the data is effectively treated as categorical. Other related works in this area include the work of He, Chang and Han [HCH04] that uses correlation mining to determine schema matches on the deep web. For related work in the AI community, we refer the reader to the survey by Doan and Halevy [DH05a].

A comprehensive schema mapping tool that incorporates learning-based matching methods is Clio [YMHF01]. Clio helps users define schema mappings - the source and target schemas can be any combination of relational and XML schemas. It incorporates an attribute matcher component that

suggests likely mappings by analyzing the schemas and underlying data, using a Naive-Bayes-based matching algorithm. Clio then also interprets these mappings to construct a set of database queries that transform and integrate source data to conform to the target schema. Such queries can be used to populate data warehouses or to define views and virtual tables in federated database environments.

In Chapter 5, we validate schema, rather than discovering them. One schema validation method is SPIDER [CT06a], by Chiticariu and Tan. SPIDER is a data-driven debugger for schema mappings, which works on top of Clio and operates with schema mappings based on a nested extension of tuple generating dependencies and equality generating dependencies. At the core of SPIDER is the notion of routes, which describes the relationship between source and target data with the schema mapping. SPIDER incorporates polynomial time algorithms for computing all routes for selected source or target data, and produces a complete, polynomial size representation of the (possibly exponential) set of all routes.

A feature of our method is the ability to validate schema matches when the data itself exhibits no match, but sub-strings of the data items match each other. The most closely related work is by Warren and Tompa [WT06], who study the discovery of multi-column schema mappings. In one sense, their problem is harder than ours, because they discover potential schema maps, while we validate such maps. However, their method assumes explicit value mappings, and in that sense inhabits a more restricted space than our approach. Our approach focuses on the syntactic types, and makes use of these syntactic types to validate the schema matching or the schema mapping. Thus it is more general than the work in [WT06].

2.3 Database Relaxation and Probabilistic Databases

New database models have emerged over the years, to better handle data with noise, as well as handle data with uncertainties. For example, there are techniques that make databases less constrained by schemas, as by integrating with other databases with noise, schemas are hard to obey at all times. This drives researchers to study on malleable schemas [DH05b, ZGBN07] or even database without specific schemas like Bigtable [CDG+06, CDG+08], or wide-table [CBN07, LHLG09]. Conceptually, Bigtable maps a unique tuple ID and a unique column ID to a data value. Each row of data only relates a small fraction of the total number of columns, which is in the magnitude of hundreds or thousands. Bigtable is totally schema free, the user can just insert his own data into an appropriate column without worrying about the schema of the database. User may even create a new column if there is no existing column that suits the data that is to be entered. Another column-wise storage, wide-table, stores column names and values in pairs. However, under both data models, one value is assigned to a unique column, which makes it impossible to process a column whose name does not match with the query.

Once the database constraints are relaxed, uncertainty is involved. Instead of a tuple being either an answer to a given query or not in RDBMS, each tuple in probabilistic databases is associated with a probability to match the query. Studies on probabilistic databases have been ongoing for many years [RS08, DRS09]. Both our work and current researches on probabilistic databases are focusing on finding an effective method for probabilistic data management and query processing. However, there are three essential

differences, namely, uncertainty types, data modeling and query processing. More researches on probabilistic databases have been done to deal with the uncertainty about row existing [GZM09, SIC07], value existing or both of them [Wid05, GS06].

Possible world is a popular model used for probabilistic database to resolve semantic ambiguity[SIC07, LD09]. We shall see in Chapter 6 that, possible world semantic does not fit to our framework, as we do not consider the uncertainty across columns to be mutually exclusive. There are correlations between columns, which allow us to adopt statistical inference to explore a more generic relationship between values and columns.

Since there exist uncertainties in the data, researchers also need to consider uncertainties when such data is being integrated, especially when schemas are heterogeneous. The majority of researches in data integration or information integration maps heterogeneous schema to one centralized mediated schema. But it is not easy to figure out a mediated schema that every other schema can be squeezed through without losing any information. Prior work on probabilistic mediate schema and probabilistic schema mapping have been proposed by Dong and Halevy [DHY07, DSDH08]. A mediated schema based on the source attribute clustering is constructed, and the probabilistic schema mapping is produced at the same time. To assign the probability of the possible mediated schema, it is assumed that the attributes in the source data are mutually exclusive with each other. In addition, only the more frequent attributes are considered, and the less frequent ones are just omitted. In Chapter 6, columns with similar semantics can coexist in our system, and values associated with one column can be inferred to other columns with similar semantics. At the same time, our method is mediated schema free, meaning our approach can be used for community data management pur-

poses, where no one defines the mediate schema and the mappings to the mediate schema.

Chapter 3

Preliminaries

In this chapter, we will review some basic concepts in *Information Theory*. Most of the definitions here are taken from the information theory text book *Elements of Information Theory* [CT06b].

3.1 Information Theory

Entropy

Let X be a discrete random variable in the space of $\mathcal{X}$, and the probability mass function $p(x) = p_X(x) = \Pr\{X = x\}, x \in \mathcal{X}$. For convenience, we write $p_X(x)$ as $p(x)$; and p refer to different probability mass functions with different random variables. For example, $p(x)$ and $p(y)$ denote $p_X(x)$ and $p_Y(y)$ respectively, so they are considered as different probability mass functions.

Definition 3.1. *The* entropy $H(X)$ *of a discrete random variable* X *is defined by*

$$H(X) = -\sum_{x \in \mathcal{X}} p(x) \log p(x) \tag{3.1}$$

As $p(x) \leq 1$, $\forall x \in \mathcal{X}$, we have

Corollary 3.2. $H(X) \geq 0$.

Joint Entropy

Definition 3.3. *The* joint entropy $H(X,Y)$ *of a pair of discrete random variables* (X,Y) *with a joint distribution* $p(x,y)$ *is defined as*

$$H(X,Y) = -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y) \tag{3.2}$$

Conditional Entropy

Definition 3.4. *The* conditional entropy $H(Y|X)$ *of a pair of discrete random variables* (X,Y) *with a joint distribution* $p(x,y) = p(x)p(y|x)$ *is defined as*

$$\begin{aligned} H(Y|X) &= \sum_{x\in\mathcal{X}} p(x)H(Y|X=x) \\ &= -\sum_{x\in\mathcal{X}} p(x)\sum_{y\in\mathcal{Y}} p(y|x)\log p(y|x) \\ &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x) \end{aligned} \tag{3.3}$$

With the definitions of joint entropy and conditional entropy, we have the following *chain rule*.

Theorem 3.5 (**Chain Rule**)**.**

$$H(X,Y) = H(X) + H(Y|X) \tag{3.4}$$

Proof.

$$\begin{aligned} H(X,Y) &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x,y) \\ &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x)p(y|x) \\ &= -\sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x) - \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x) \\ &= -\sum_{x\in\mathcal{X}} p(x)\log p(x) - \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(y|x) \\ &= H(X) + H(Y|X) \end{aligned}$$

□

The following corollary illustrates the relationship between three conditional entropies with three random variables involved.

Corollary 3.6.

$$H(X,Y|Z) = H(X|Z) + H(Y|X,Z)$$

Kullback-Leibler Divergence and Relative Entropy

Definition 3.7. *The* Kullback-Leibler divergence *(*KL divergence *for short) between two probability mass function $p(x)$ and $q(x)$ is defined as*

$$D_{KL}[p\|q] = \sum_{x\in\mathcal{X}} p(x)\log\frac{p(x)}{q(x)} \tag{3.5}$$

In this definition, we define $0\log\frac{0}{q} = 0$ and $p\log\frac{p}{0} = \infty$, based on continuity theorem.

As $D_{KL}[p\|q]$ can be written as

$$D_{KL}[p\|q] = \mathbb{E}_p[\log\frac{p(X)}{q(X)}] \tag{3.6}$$

It is also known as the *relative entropy* or *cross entropy* as it is a measure of the inefficiency of assuming that the distribution is q when the true distribution is p.

The KL divergence is non-negative, as shown in the following corollary.

Corollary 3.8.

$$D_{KL}[p\|q] \geq 0$$

and

$$D_{KL}[p\|q] = 0 \iff p = q$$

Proof. Jensen's inequality [BV04] states

$$f(\mathbb{E}[X]) \leq \mathbb{E}[f(X)]$$

for a *convex function* f defined on a random variable X.

As $-\log(\cdot)$ is a convex function, we have,

$$-\log(\mathbb{E}[g(X)]) \leq \mathbb{E}[-\log(g(X))] \tag{3.7}$$

where $g(\cdot)$ is a function from $\mathcal{X}$ to positive real numbers, i.e., $g\colon \mathcal{X} \to \Re^+$.

Define $g(x) = \frac{q(x)}{p(x)}$ for all $x \in \mathcal{X}$ following probability distribution $p(x)$, by Equation 3.7,

$$\begin{aligned} D_{KL}[p\|q] &= \mathbb{E}_p[\log \frac{p(X)}{q(X)}] \\ &= \sum_{x\in\mathcal{X}} p(x) \log \frac{p(x)}{q(x)} \\ &= \sum_{x\in\mathcal{X}} p(x) - \log \frac{q(x)}{p(x)} \\ &\geq -\log \sum_{x\in\mathcal{X}} p(x) \frac{q(x)}{p(x)} \\ &= -\log 1 \\ &= 0 \end{aligned}$$

□

In general, KL divergence is not symmetric and does not satisfy the triangle inequality. Nevertheless, it is still regarded as a "distance" between two probability distributions.

Mutual Information

Definition 3.9. *Consider two random variables X and Y with a joint probability mass function $p(x, y)$ and marginal probability mass functions $p(x)$ and $p(y)$. The* mutual information *$I(X;Y)$ is the relative entropy between the joint distribution and the product distribution $p(x)p(y)$, i.e.,*

$$I(X;Y) = \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \tag{3.8}$$

Corollary 3.10.

$$I(X;Y) = H(X) - H(X|Y)$$

Proof.

$$\begin{aligned}
I(X;Y) &= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x,y)}{p(x)p(y)} \\
&= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x|y)}{p(x)} \\
&= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log\frac{p(x|y)}{p(x)} \\
&= \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x|y) - \sum_{x\in\mathcal{X}}\sum_{y\in\mathcal{Y}} p(x,y)\log p(x) \\
&= \sum_{y\in\mathcal{Y}}\sum_{x\in\mathcal{X}} p(x,y)\log p(x|y) - \sum_{x\in\mathcal{X}} p(x)\log p(x) \\
&= -H(X|Y) + H(X)
\end{aligned}$$

where the last equation is given by Equation 3.1 and Equation 3.3. □

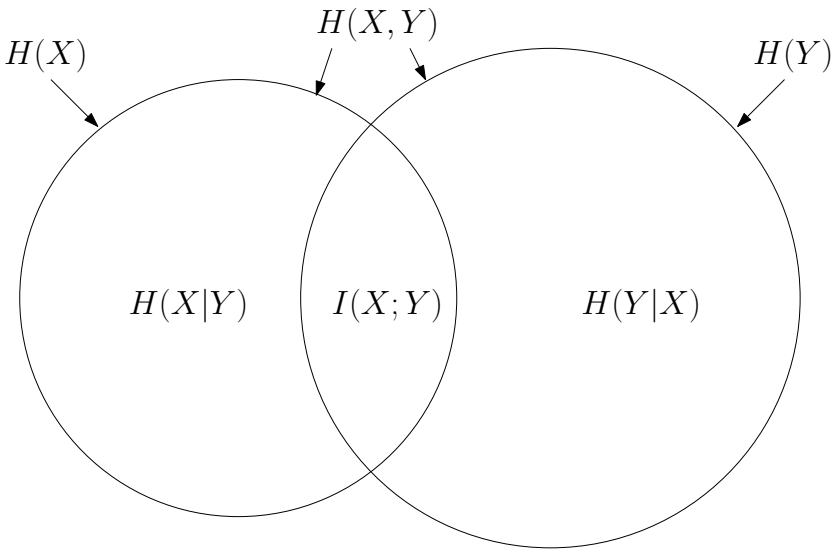


Figure 3.1: Relationships among entropy, joint entropy, conditional entropy and mutual information.

By symmetry, $I(X;Y) = H(Y) - H(Y|X)$. With Theorem 3.5, the chain rule theorem, we have $I(X;Y) = H(X) + H(Y) - H(X,Y)$.

As shown in Figure 3.1, if the two circles represent $H(X)$ and $H(Y)$ respectively, the intersection of the two circles then represents the mutual information $I(X,Y)$ and the union of the two circles represents the joint entropy $H(X,Y)$.

Jensen-Shannon divergence

Since KL divergence is not symmetric, *Jensen–Shannon divergence* [Lin91] (*JS divergence* for short) is developed based on KL divergence as a symmetric measure between two distribution.

Definition 3.11. *The* Jensen-Shannon divergence *(*JS divergence *for short)*

between two probability mass function $p(x)$ and $q(x)$ is defined as

$$D_{JS}(p, q, \pi_1, \pi_2) = \pi_1 \cdot D_{KL}[p\|r] + \pi_2 \cdot D_{KL}[q\|r] \tag{3.9}$$

where $0 \leq \pi_1, \pi_2 \leq 1$, $\pi_1 + \pi_2 = 1$, and

$$r = \pi_1 \cdot p + \pi_2 \cdot q$$

is considered as a distribution between p and q.

When $\pi_1 = \pi_2 = \frac{1}{2}$, $D_{JS}(p, q, \frac{1}{2}, \frac{1}{2}) = D_{JS}(q, p, \frac{1}{2}, \frac{1}{2})$ is a symmetric "distance" between probability distributions p and q.

Next we will give a brief introduction on the *Information Bottleneck Method* method, which is the underlying clustering method that we will be using throughout the thesis.

3.2 the Information Bottleneck Method

Here we give a very brief introduction to the *information bottleneck method*. For readers who wish to check the details of the information bottleneck method, they may proceed to Slonim's thesis [Slo03].

As illustrated in Figure 3.2, the information bottleneck method connects three spaces, $\mathcal{X}$, $\mathcal{Y}$ and $\mathcal{T}$. $\mathcal{X}$ and $\mathcal{Y}$ are known, whereas $\mathcal{T}$ can be considered as a compressed space for $\mathcal{X}$, and $\mathcal{T}$ tries to preserve as much information as $\mathcal{X}$ preserves about $\mathcal{Y}$.

The information bottleneck method is an *EM*-like algorithm that iterates through an E-step and an M-step to an optimal solution for $\mathcal{T}$. In the E-step, $p(Y|T)$ is fixed, the information bottleneck method looks for the optimal $p(T|X)$. While in the M-step, $p(T|X)$ is fixed, the information bottleneck method seeks for the best $p(Y|T)$. Since we want to minimize $I(T; X)$

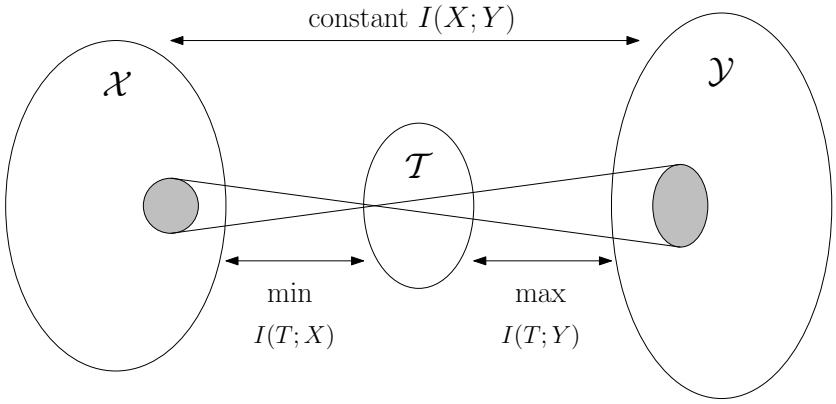


Figure 3.2: The *information bottleneck method* tries to squeeze as much information as X about Y, i.e., $I(X;Y)$, through the bottleneck $\mathcal{T}$, so that $I(T;Y)$ can be as close as $I(X;Y)$

and maximize $I(T;Y)$ at the same time, we use a function $\mathcal{L}$, called as IB functional to balance these two quantities through a parameter β, i.e.,

$$\mathcal{L} = I(T;X) - \beta \cdot I(T;Y)$$

In M-step, as $p(T|X)$ is given, the solution maximizes $I(T;Y)$ is

$$\begin{cases} p(t) = \sum_{x\in\mathcal{X}} p(x)p(t|x) \\ p(y|t) = \frac{1}{p(t)} \sum_{x\in\mathcal{X}} p(x,y)p(t|x) \end{cases} \tag{3.10}$$

Thus $\mathcal{L}$ can be written as a function of $p(T|X)$.

By setting $\frac{\partial \mathcal{L}}{\partial p(t|x_0)} = \lambda_0 \quad \forall\ t \in \mathcal{T}$ where λ_0 is the Lagrange multiplier associated with $x_0 \in \mathcal{X}$ since $\sum_{t\in\mathcal{T}} p(t|x_0) = 1$, we have

$$p(t|x) = \frac{p(t)}{Z(x,\beta)} e^{-\beta \cdot D_{KL}[p(y|x)\|p(y|t)]} \tag{3.11}$$

where $Z(x,\beta)$ is the normalization function to ensure $\sum_{t\in\mathcal{T}} p(t|x) = 1$.

Note that parameter β controls the compression of $\mathcal{X}$. When $\beta = 0$, according to Equation 3.11, $p(t|x)$ is the same for all t. Therefore, every

$x \in \mathcal{X}$ will have the identical clustering, which makes all $x \in \mathcal{X}$ under one cluster. On the other extreme, if $\beta = \infty$, every $x \in \mathcal{X}$ is a assigned into the nearest t, thus $\mathcal{L}$ is minimized when there is one t for each distinct x, which puts every distinct $x \in \mathcal{X}$ into an individual cluster.

Given parameter β, the `iIB` algorithm interates between Equation 3.10 and Equation 3.11, and stops when there is no change in $p(T|X)$.

Unlike k-means clustering controls the compression of the original data by the number of cluster k, the amount of compression of the `iIB` clustering is controled by parameter β here. We shall see how to choose the best β in Chapter 4.

Chapter 4

Intra-Column Heterogeneity: The Column Heterogeneity Measure

Database column is a fundamental unit in database design and manipulation, which makes column heterogeneity a serious concern in every data management application. Textbook database design teaches that it is desirable for a database column to be homogeneous, i.e., all values in a column should be of the same "type". If a database column contains several different types of values, each type should be represented in separate columns. Such heterogeneity of values within the same column is called intra-column heterogeneity. We say a column is homogeneous if it contains homogeneous values. For example, the column in Table 4.1(a) contains only email addresses and is quite homogeneous, even though there appears to be a wide diversity in the actual set of values present. Such homogeneity of database column values has obvious advantages, including simplicity of application-level code that accesses and modifies the database.

CUSTOMER_ID
lkjkjjk@321.zzz.info
h8742@yyy.com
kkjj+@haha.org
qwerty@keyboard.us
555-1212@fax.in
alpha@beta.ga
john.smith@noname.org
jane.doe@1973law.us
gwb.dc@universe.gov
jamesbond.007@action.com

(a)

CUSTOMER_ID
lkjkjjk@321.zzz.info
h8742@yyy.com
kkjj+@haha.org
qwerty@keyboard.us
555-1212@fax.in
(908)-555.1234
973-360-0000
360-0007
8005551212
(877)-807-4596

(b)

CUSTOMER_ID
lkjkjjk@321.zzz.info
h8742@yui.com
kkjj+@haha.org
qwerty@keyboard.us
555-1212@fax.in
alpha@beta.ga
john.smith@noname.org
jane.doe@1973law.us
gwb.dc@universe.gov
8778074596

(c)

CUSTOMER_ID
123-45-6789
135-79-2468
159-24-6837
789-12-3456
987-65-4321
(908)-555.1234
973-360-0000
360-0007
8005551212
(877)-807-4596

(d)

Table 4.1: Example homogeneous and heterogeneous columns.

In practice, operational databases evolve over time to contain a great deal of "heterogeneity" in database column values. Often, this is a consequence of large scale data integration efforts that seek to preserve the "structure" of the original databases in the integrated database, to avoid having to make extensive changes to legacy application level code. For example, one application might use email addresses as a unique customer identifier, while another might use phone numbers for the same purpose. When their databases are integrated into a common database, it is feasible that the `CUSTOMER_ID` column contains both email addresses and phone numbers, both represented as strings, as illustrated in Table 4.1(b). A third independently developed application that used, say, social security numbers as a customer identifier might then add such values to the `CUSTOMER_ID` column, and when its database is integrated into the common database. As another example, two different inventory applications might maintain machine domain names (e.g., abc.def.com) and IP addresses (e.g., 105.205.105.205) in the same `MACHINE_ID` column for the equivalent task of identifying machines connected to the network. While these examples may appear "natural" since all of these different types of values have the same function, namely, to serve as a customer identifier or a machine identifier, potential data quality problems can arise in databases that are accessed and modified by legacy applications that are unaware of the heterogeneity of values in the column.

For example, an application that assumes that the `CUSTOMER_ID` column contains only phone numbers might choose to "normalize" column values by removing all special characters (e.g., '-', '.') from the value, and writing it back into the database. While such a transformation is appropriate for phone numbers, it would clearly mangle the email addresses represented in the column and can severely degrade common business practices. For instance, the

unanticipated transformation of email addresses in the `CUSTOMER_ID` column (e.g., "`john.smith@noname.org`" to "`johnsmith@nonameorg`") may mean that a large number of customers are no longer reachable.

Locating poor quality data in large operational databases is a non-trivial task, especially since the problems may not be due to the data alone, but also due to the interactions between the data and the multitude of applications that access this data (as the previous example illustrates). Identifying heterogeneous database columns becomes important in such a scenario, permitting data quality analysts to then focus on understanding the interactions of applications with data in such columns, rather than having to simultaneously deal with the tens of thousands of columns in today's complex operational databases. If an analyst determines that a problem exists, remedial actions can include:

- Modification of the applications to explicitly check for the type of data (phone numbers, email addresses, etc.) assumed to exist in the table, or
- A horizontal splitting of the table to force homogeneity, along with a simpler modification of the applications accessing this table to access and update the newly created tables instead.

We next identify desiderata that a column heterogeneity measure should intuitively satisfy, followed by a discussion of techniques to quantify column heterogeneity that meet these desiderata.

4.1 Heterogeneity: Desiderata

Consider the example shown in Table 4.1. This illustrates many of the issues that need to be considered when coming up with a suitable measure for column heterogeneity.

Number of Semantic Types: Many semantically different types of values (email addresses, phone numbers, social security numbers, circuit identifiers, IP addresses, machine domain names, customer names, etc.) may be represented as strings in a column, with no a priori characterization of the possible semantic types present.

Intuitively, the more semantically different the types of values there are in a database column, the greater the heterogeneity should be; thus, heterogeneity is better modeled as a numerical value rather than a Boolean (yes/no). For example, we can be confident that a column with both email addresses and phone numbers (e.g., Table 4.1(b)) is more heterogeneous than one with only email addresses (e.g., Table 4.1(a)) or only phone numbers.

Distribution of Semantic Types: The semantically different types of values in a database column may occur with different frequencies.

Intuitively, the relative distribution of the semantically different types of values in a column should impact its heterogeneity. For example, we can be confident that a column with many email addresses and many phone numbers (e.g., Table 4.1(b)) is more heterogeneous than a column that has mainly email addresses with just a few outlier phone numbers (e.g., Table 4.1(c)).

Distinguishability of Semantic Types: Semantically different types of values may overlap (e.g., social security numbers and phone numbers) or be easily distinguished (e.g., email addresses and phone numbers).

Intuitively, with no a prior characterization of the set of possible semantic types present in a column, we cannot always be sure that a column is heterogeneous, and our heterogeneity measure should conservatively reflect this possibility.

The more easily distinguished are the semantically different types of values in a column, the greater should be its heterogeneity. For example, a column with roughly equal numbers of email addresses and phone numbers (e.g., Table 4.1(b)) can be said to be more heterogeneous than a column with roughly equal numbers of phone numbers and social security numbers (e.g., Table 4.1(d)), due to the greater similarity between the values (and hence the possibility of being of the same unknown semantic type) in the latter case.

4.1.1 Semantic Types and Syntactic Types

In reality, semantics are the interpretations in one's mind, which are difficult to capture. Once they are materialized to any form of representations, e.g., words, speeches, or pictures, the semantics may have changed as the interpretations of the readers may be different from the original interpretations of the writer, who materialize the semantics to the representations.

In contrast to semantic types, types of representation are termed as syntactic types. For example, there are two syntactic types in Table 4.1(b), each representing one semantic type, email addresses and phone numbers. We have a correspondence between syntactic types and semantic types here.

However, this is not true in all cases. There are cases where one semantic type is associated to several syntactic types. For instance, the 20th day of June 2006 can be written as "06/20/2006" (American), "20/06/2006" (British) or "2006-06-20" (Chinese). Each representation is a distinct syn-

tactic type, but they are of the same semantic type. The reciprocal is also true, i.e., one syntactic type can be interpreted as different semantic types. For example, phone numbers and fax numbers are of the same syntactic type, but they are semantically different. Even for one particular value, semantics can be different, depending on individuals' interpretation. Take "`TT`" for example, it means a crying emoticon in an on-line chatting environment, and it also means one of the car models made by manufacturer `Audi`. This is the intrinsic factor of semantics, that the semantics only make sense with human beings' interpretations.

Capturing the semantics of one individual value is difficult, however, when values are grouped together, it is easier to identify their semantics, provided that those values are assumed to be semantically homogeneous. The semantic types can be inferred from the syntactic types. We will come back to semantic types in Chapter 6, we now focus on syntactic types of values in database columns, as the identification of syntactic types is the prerequisite of the identification of semantic types. Our column heterogeneity thus refers to the heterogeneity of syntactic types in a column.

4.2 Quantifying Column Heterogeneity

Given the desiderata outlined above, we now present a step-wise development of our approach to quantify database column heterogeneity.

A first approach to obtaining a heterogeneity measure is to use a *hard clustering*. By partitioning values in a database column into clusters, we can get a sense of the number of different syntactic types of values in the data. However, merely counting the number of clusters does not suffice to quantify heterogeneity. Two additional issues, as outlined above, make the problem

challenging: the relative sizes of the clusters and their distinguishability. A few phone numbers in a large collection of email addresses (e.g., Table 4.1(c)) may look like a distinct cluster, but should not impact the heterogeneity of the column as much as having a significant number of phone numbers with the same collection of email addresses (e.g., Table 4.1(b)). Again, a social security number (see the first few values in Table 4.1(d)) may look similar to a phone number, and we would like the heterogeneity measure to reflect this overlap of sets of values, as well as be able to capture the idea that certain data might yield clusters that are close to each other, and other data might yield clusters that are far apart.

To take into account the relative sizes of the multiple clusters, *cluster entropy* is a better measure for quantifying heterogeneity of data in a database column than merely counting the number of clusters. Cluster entropy is computed by assigning a "probability" to each cluster equal to the fraction of the data values it contains, and computing the entropy of the resulting distribution [CT06b].

Consider a hard clustering $\mathcal{T} = \{t_1, t_2, \ldots, t_k\}$ of a set of n values in $\mathcal{X}$, where $\mathcal{T}$ is a partition on $\mathcal{X}$, i.e., for each $x_i \in \mathcal{X}$, there is one and only one t_j that x_i belongs to. Each cluster t_j has n_j values, where $n_1 + n_2 + \ldots + n_k = n$. We denote $p_j = \frac{n_j}{n}$ as the fraction of the data values cluster t_j contains. Then the *cluster entropy* of the hard clustering T is the entropy of the cluster size distribution, defined as

$$H(\mathcal{T}) = -\sum_{j=1}^{k} p_j \log(p_j)$$

By using cluster entropy, the mixture of email addresses and phone numbers in column Table 4.1(b) is $-\frac{1}{2}\log\frac{1}{2} - \frac{1}{2}\log\frac{1}{2} = 1$, while the mixture in column Table 4.1(b) is $-\frac{9}{10}\log\frac{9}{10} - \frac{1}{10}\log\frac{1}{10} = 0.469$, which is lower than the value

of heterogeneity in Table 4.1(b). This matches the intuition as Table 4.1(c) is less heterogeneous as it consists of mainly email addresses.

The cluster entropy of a hard clustering does not effectively take into account distinguishability of syntactic types in a column. For example, given a column with an equal number of phone numbers and social security numbers (e.g., Table 4.1(d)), hard clustering could either determine the column to have one cluster (in which case its cluster entropy would be 0, which is the same as that of a column with just phone numbers) or have two equal sized clusters (in which case its cluster entropy would be $\log 2$, which is the same as that of a column with equal numbers of phone numbers and email addresses as in Table 4.1(b)). Intuitively, however, the heterogeneity of such a column should be somewhere in between these two extremes to capture the uncertainty in assigning values to clusters due to the syntactic similarity of values. *Soft clustering* has the potential to address this problem; each data value in soft clustering has the flexibility of assigning a probability distribution for its cluster membership, instead of belonging to a single cluster, as in hard clustering. Heterogeneity can now be combined from the two concepts: the *cluster entropy* and the *soft clustering*.

Therefore, we propose a measure of database column heterogeneity, namely,

Heterogeneity
$=$ *the "cluster entropy" of a soft clustering of the data*

The discussion about the "cluster entropy" of a soft clustering will be left to later sections.

In Section 4.1, we gave a brief introduction on column heterogeneity and the desiderata that a column heterogeneity measure should intuitively satisfy. We also discussed how the column heterogeneity should be quantified and claimed that the column heterogeneity is the "cluster entropy" of a soft

clustering of the data.

In the next sections, we will go into the details of column heterogeneity measure by first claiming that the "cluster entropy" of a soft clustering is actually a quantity called *mutual information*. As the heterogeneity measure relies on a soft cluster, we will then discuss how soft clustering is chosen. Lastly, some technical details will be given, followed by a series of experiments to verify the correctness of our column heterogeneity measure.

4.3 Mutual Information: the Column Heterogeneity

We first give the definition of Mutual Information here.

Let $\mathcal{T} = \{t_1, t_2, \ldots, t_k\}$ be a soft clustering of a data set $\mathcal{X} = \{x_1, x_2, \ldots, x_n\}$, each x_i is associated with probability $p(x_i)$ where $\sum_{i=1}^{n} p(x_i) = 1$. Further, let $p(t_j|x_i)$ be the posterior probability of x_i belonging to t_j where $\sum_{j=1}^{k} p(t_j|x_i) = 1$, and $p(t_j) = \sum_{i=1}^{n} p(x_i, t_j)$ is the marginal probability over $\mathcal{X}$ for all $1 \leq j \leq k$. Using conditional probabilities, $p(x_i, t_j) = p(x_i) \cdot p(t_j|x_i)$, we have $p(t_j) = \sum_{i=1}^{n} p(x_i) \cdot p(t_j|x_i)$ for all $1 \leq j \leq k$, the *Mutual Information* is then defined as

$$\begin{aligned} I(\mathcal{X}, \mathcal{T}) &= \sum_{i=1}^{n} \sum_{j=1}^{k} p(x_i, t_j) \log \frac{p(x_i, t_j)}{p(x_i) \cdot p(t_j)} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{k} p(x_i) p(t_j|x_i) \log \frac{p(t_j|x_i)}{p(t_j)} \\ &= \sum_{i=1}^{n} p(x_i) \sum_{j=1}^{k} p(t_j|x_i) \log \frac{p(t_j|x_i)}{p(t_j)} \end{aligned}$$

Recall that the *Kullback–Leibler divergence (KL divergence)* from proba-

bility distribution q to probability distribution p

$$D_{KL}[p\|q] = \sum_x p(x) \log \frac{p(x)}{q(x)}$$

Therefore, the Mutual Information is

$$\begin{aligned} I(\mathcal{X}, \mathcal{T}) &= \sum_{i=1}^{n} p(x_i) \sum_{j=1}^{k} p(t_j|x_i) \log \frac{p(t_j|x_i)}{p(t_j)} \\ &= \sum_{i=1}^{n} p(x_i) D_{KL}[p(t|x_i)\|p(t)] \\ &= \mathbb{E}_x\{D_{KL}[p(t|x)\|p(t)]\} \end{aligned}$$

From the equation above, the mutual information can be understood as the expected KL divergence from $p(t|x)$ to $p(t)$. In hard clustering, $p(t|x)$ has only one element being one with the rest being zero for all x. Thus $D_{KL}[p(t|x)\|p(t)] = \log \frac{1}{p(t_0)}$, where t_0 is the cluster such that $p(t_0|x) = 1$.

$$\begin{aligned} \mathbb{E}_x\{D_{KL}[p(t|x)\|p(t)]\} &= \sum_x p(x) \log \frac{1}{p(t_0)} \\ &= \sum_t \sum_{x \in t} p(x) \log \frac{1}{p(t)} \\ &= \sum_t p(t) \log \frac{1}{p(t)} \\ &= H(\mathcal{T}) \end{aligned}$$

From the equation above, we see that mutual information is a generalization of cluster entropy in the soft clustering case.

We can now restate our heterogeneity measure as

Heterogeneity
= *the mutual information between a soft clustering and the data*

The mutual information depends on the choice of the soft clustering, in the next section, we will discuss how we choose the soft clustering.

4.3.1 A Canonical Soft Clustering

There are numerous methods for performing soft clusterings. Perhaps the most well known among them is the class of methods known as expectation-maximization, or EM [DLR77]. EM returns a probabilistic assignment of points to clusters. The main problem with this method (and others like it) is that they require the user to fix k, the number of clusters, in advance. One can circumvent this problem by finding the "right" value of k using model checking criteria like AIC [Aka74], BIC [Sch78] and others, but these are all based on assumptions about the distributions the data is drawn from, using maximum likelihood methods to estimate the "most likely" value of k.

A different approach is to use the idea of rate-distortion from information theory [CT06b]. There are two parameters that constrain any clustering method. The first is representation complexity R, or how much compression one can achieve by clustering. Let $\mathcal{T}$ be a compressed representation of $\mathcal{X}$, a standard measure for R from information theory is *the rate of coding* when compressing $\mathcal{X}$ to $\mathcal{T}$. Formally, R is the mutual information between the data $\mathcal{X}$ and the compressed representations (clusters) $\mathcal{T}$, i.e., $R = I(T; X)$. A detailed explanation of quantity R is beyond the scope of this thesis, readers may refer to any information theory book, e.g. [CT06b].

The second is the quality Q, or how accurately the clustering reflects the original data. Often, it is more convenient to think of the error E, which is typically some constant minus Q. The error E is measured by the distortion of representing $\mathcal{X}$ using $\mathcal{T}$, i.e., the average distance to the cluster centers

over all clusters.

$$\begin{aligned} E &= \langle d(x_i, t_j) \rangle_{p(x_i,t_j)} \\ &= \sum_{i=1}^{n} \sum_{j=1}^{k} p(x_i, t_j) d(x_i, t_j) \\ &= \sum_{i=1}^{n} \sum_{j=1}^{k} p(x_i) p(t_j|x_i) d(x_i, t_j) \end{aligned} \tag{4.1}$$

For any fixed level of compression (this is analogous to fixing k), one can determine the best quality representation, and for any fixed quality level, one can determine the best compression possible. There exists a trade-off between compression R and quality Q. Low value of R implies more compact representations. For example, when $R = 0$, $\mathcal{X}$ and $\mathcal{T}$ are independent, i.e., knowing about T does not give any information about knowing X, thus we can consider a clustering T with $R = I(T; X)$ as all $x \in \mathcal{X}$ are clustered into one single cluster, which gives the worst quality since every x is treated the same.

On the other extreme, when $R = I(T; X) \leq H(X)$ reaches the maximum $H(X)$, each $x \in \mathcal{X}$ is placed in separate clusters of its own. In this case, there is no distortion (best possible quality), as $d(x_i, t_j) = 0$ for all $p(t_j|x_i) = 1$. In rate-distortion theory, compression R and quality Q are parameterized by a single Lagrange parameter β, so that the two quantity R and Q are balanced by

$$\mathcal{F}(\beta) = R - \beta \cdot Q \tag{4.2}$$

through β.

For each choice β, there exists a soft clustering which gives the optimal solution with respect to function F. The soft clustering then corresponds to a point whose x-coordinate and y-coordinate represent the compression R_β

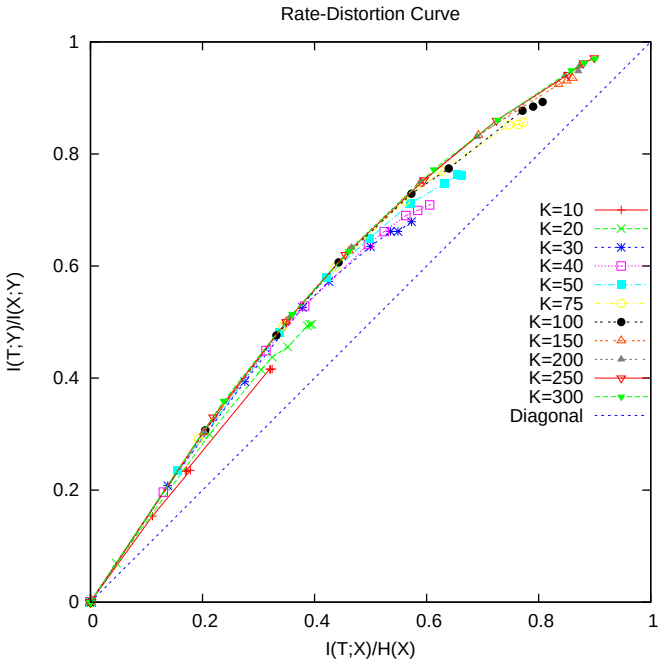


Figure 4.1: Rate-Distortion curve for a mixture of `make` values and `model` values from table `autosforsale-2`. Note that the trade-offs achieved are better than those achieved by fixing K, the number of clusters, to any specific value.

and the quality Q_β respectively. By changing β, the point (R_β, Q_β) moves along a concave curve known as the *rate-distortion curve* (see Figure 4.1).

Choosing $\beta = 0$ implies that quality does not matter, the only objective is to minimize R. Thus the optimal clustering is the one that all $x \in \mathcal{X}$ are in the same cluster. This clustering makes both R and Q zero, thus corresponds to the origin of the rate-distortion graph, with x-axis and y-axis being compression and quality respectively. Letting β go to ∞ is equivalent to making compression irrelevant; maximizing Q is the sole objective. In this case, each $x \in \mathcal{X}$ forms an individual cluster (this is the top right point of the curve). Note that the slope of the curve at any point is $\frac{1}{\beta}$.

It is important to note that each point on the rate distortion curve is a soft clustering that uses as many clusters as needed to obtain the optimal value of the rate distortion functional $\mathcal{F}(\beta)$, for the corresponding value of β. Any fixed choice of the number of clusters to use will ultimately be suboptimal, as the data separates into more and more clusters. As shown in Figure 4.1, each curve is obtained by the compression and quality values achieved by soft clusterings with different β using a fixed number K of clusters. For each such number K, there is a point at which the corresponding curve separates from the rate-distortion curve, and proceeds along a suboptimal path (less compression, lower quality). As it is not possible to supply infinite number of clusters, the envelope curve of all curves determined by different K is the optimal rate-distortion curve.

The choice of a soft clustering is thus made by choosing β. Normalizing the x-coordinate and y-coordinate by the maximum value of R and Q, the rate-distortion curve goes from $(0, 0)$ to $(1, 1)$. We choose *the point of diminishing returns*; namely the point at which the slope of the curve is one. The reason for our choice is two-fold. First, the point of diminishing returns, because it has unit slope, is the point after which the benefits of increased quality do not pay for the increasing amount of compression needed for the representation. Second, this point is the closest point to the $(0, 1)$ point, which is the point representing perfect quality with no compression space penalty.

In the next section, we will further quantify compression and quality with respect to *the information bottleneck method*.

4.3.2 The Information Bottleneck Method

Our proposed choice of β is quite general, and is independent of both the distance function used, and any distributional assumption. To perform the clustering though, we must fix a representation for the data, and an appropriate distance function. As pointed out by Banerjee *et al.* [BMDG05], choosing any Bregman distance yields a meaningful rate-distortion framework, and thus the choice of distance depends on the data representation used.

Choosing ℓ_2^2 as the distance function yields an EM-like method, which would be appropriate for vector data. Our data are strings, which we choose to represent as q-gram distributions. A q-gram of a string s is any substring of s of length exactly q. For example, the set of 1-grams of a string is the multi-set of all the letters in the string. A natural distributional model for string data is a multinomial distribution (for example, the naive Bayes method), and the corresponding Bregman distance is the Kullback-Leibler distance. Using this measure yields a familiar formulation; the information bottleneck method of Tishby, Pereira and Bialek [TPB99, Slo03].

In [Slo03], each $x \in \mathcal{X}$ is represented as a probability distribution in the space of $\mathcal{Y}$, i.e., $(p(y_1|x_i), p(y_2|x_i), \ldots, p(y_d|x_i))$ for each $x_i \in \mathcal{X}$, where $d = |\mathcal{Y}|$. In our problem, $\mathcal{Y}$ is chosen to be the space of q-grams, which are strings of length not more than q. In general, q is small and q-grams are the basic elements composing longer strings.

The information bottleneck method takes $\mathcal{Y}$ and $\mathcal{T}$ as conditionally independent given $\mathcal{X}$. Given the cluster distribution $p(t_j|x_i) : 1 \leq i \leq n, 1 \leq j \leq k$, and the representation of $x \in \mathcal{X}$ in the space of $\mathcal{Y}$, i.e.,

$p(y_l|x_i):\ 1 \le i \le n, 1 \le l \le d$,

$$\begin{aligned} p(t_j) &= \sum_{i=1}^{n} p(x_i, t_j) \\ &= \sum_{i=1}^{n} p(x_i) \cdot p(t_j|x_i) \end{aligned} \tag{4.3}$$

$$\begin{aligned} p(y_l|t_j) &= \sum_{i=1}^{n} \frac{p(x_i, y_l, t_j)}{p(t_j)} \\ &= \frac{1}{p(t_j)} \sum_{i=1}^{n} p(x_i) \cdot p(y_l|x_i) \cdot p(t_j|x_i) \end{aligned} \tag{4.4}$$

With the above definition of $p(y_l|t_j)$, we have both $x \in \mathcal{X}$ and $t \in \mathcal{T}$ expressed as probabilistic distributions in the space of $\mathcal{Y}$. So the distance from x_i to t_j in Equation 4.1 is defined by the KL divergence from $p(y_1|x_i), p(y_2|x_i), \ldots, p(y_d|x_i))$ to $p(y_1|t_j), p(y_2|t_j), \ldots, p(y_d|t_j))$. Therefore, the distortion defined in Equation 4.1 is quantified by

$$\begin{aligned} & \langle d(x_i, t_j) \rangle_{p(x_i, t_j)} + I(T;Y) \\ = & \sum_{i=1}^{n} \sum_{j=1}^{k} p(x_i) p(t_j|x_i) D_{KL}[p(y|x_i) \| p(y|t_j)] \\ & + \sum_{j=1}^{k} \sum_{l=1}^{l} p(t_j, y_l) \log \frac{p(t_j, y_l)}{p(t_j) p(y_l)} \\ = & \sum_{i=1}^{n} \sum_{j=1}^{k} \sum_{l=1}^{d} p(x_i, t_j) p(y_l|x_i) \log \frac{p(y_l|x_i)}{p(y_l|t_j)} \\ & + \sum_{i=1}^{n} \sum_{j=1}^{k} \sum_{l=1}^{l} p(x_i, t_j, y_l) \log \frac{p(y_l|t_j)}{p(y_l)} \\ = & \sum_{i=1}^{n} \sum_{j=1}^{k} \sum_{l=1}^{l} p(x_i, t_j, y_l) [\log \frac{p(y_l|x_i)}{p(y_l|t_j)} + \log \frac{p(y_l|t_j)}{p(y_l)}] \\ = & \sum_{i=1}^{n} \sum_{l=1}^{d} p(x_i, y_l) \log \frac{p(y_l|x_i)}{p(y_l)} \\ = & I(X;Y) \end{aligned}$$

Hence, the error E is expressed by $I(X;Y)-I(T;Y)$, and by Equation 4.2, $\mathcal{F}(\beta) = I(T;X)+\beta I(X;Y)-\beta I(T;Y)$. This also implies quality Q is upper bounded by $I(X;Y)$, while compression R is upper bounded by $H(X)$ since $I(T;X) = H(X) - H(X|T) \leq H(X)$. The normalized x-coordinate and y-coordinate of a point on the rate-distortion curve are given by $\frac{I(T;X)}{H(X)}$ and $\frac{I(T;Y)}{I(X;Y)}$ respectively, where T is the soft clustering corresponding to the point with a given β. Therefore, the choice of β that yields the point of unit slope on the normalized rate-distortion curve is given by $\beta^* = \frac{H(X)}{I(X;Y)}$, i.e., β^* gives the point of diminishing returns, proposed in section 4.3.1.

Next, we use the `iIB` algorithm [Slo03] to compute the soft clustering with the β^* calculated from the data.

4.3.3 Estimating Heterogeneity

Once we compute a soft clustering using the method described above, we need to estimate its heterogeneity. The central idea of this chapter is that the entropy of a soft clustering is a good measure of data heterogeneity. However, cluster entropy is defined only for a hard clustering. Since we know that any hard clustering can be expressed in terms of probabilistic cluster assignments using only zero and one for probabilities, (and uniform priors on the elements x), we would like the measure that we propose to tend towards (hard clustering) cluster entropy in this limiting case.

Using Cluster Marginals

The first approach that comes to mind is to determine the marginals $p(t_j) = \sum_{x=1}^{n} p(x_i)p(t_j|x_i)$. Our measure is then the entropy of the resulting distribution. Clearly, if all assignments are 0-1 and $p(x) = \frac{1}{n}$, $p(t_j) = \frac{|\{x_i|p(t_j|x_i)=1\}|}{n}$, and this reduces to $H(T)$. However, by aggregating the individual cluster

membership probabilities, we have lost crucial information about the data. Consider two different soft clusterings of two points x_1, x_2 into two clusters t_1, t_2. In the first clustering, $p(t_1|x_1) = 0.9, p(t_1|x_2) = 0.1$. In the second clustering, $p(t_1|x_1) = p(t_1|x_2) = 0.5$. Both clusterings yield the same marginal $p(t)$ and thus would have the same cluster entropy using the proposed measure. However, it is clear that in the first clustering, x_1 is essentially in t_1 and x_2 is essentially in t_2 (0.9 can be replaced by any number $1 - \epsilon$ for this purpose), and so the cluster entropy should be close to $\log 2$. In the second clustering, however, t_1 and t_2 are indistinguishable, which means that there is effectively only one cluster, with a cluster entropy of zero.

Two issues are illuminated by this example. First, aggregating cluster probabilities is not an appropriate equivalent of cluster entropy. Second, *the number of clusters in a soft clustering is an irrelevant parameter.* This is because two clusters having identical cluster membership probabilities will be collapsed (intuitively because they are indistinguishable from each other). Note that the membership probabilities $p(t|x)$ for a point x do not have to be identical for this to happen.

Using Superpositions of Hard Clusterings

The second approach we might take is to view the probabilistic assignments as the convex superposition of different hard clusterings. In this view, the assignments reflect the superposition of different "worlds", each with its own hard clustering. In this case, our strategy is clear; we assign each point to a cluster using the distribution $p(t|x)$, and compute the cluster entropy of the resulting hard clustering. Doing this repeatedly and taking an aggregate (mean or median) gives us an estimate. Note that this approach will yield $H(T)$, as desired, when applied to a single hard clustering, because each

point is always assigned to a specific cluster.

However, the second approach in the above example outlines a problem with this strategy. If a set of points have identical cluster memberships in a set of clusters, then in any particular sample, the points might be distributed among many clusters, rather than all being placed together as they should. For example, in some samples, x_1 might be placed in t_1, and x_2 might be placed in t_2, and in others, they might be placed together. To address this problem, we have to *merge* clusters that have similar cluster membership probabilities, using a threshold parameter and some appropriate merging strategy.

The result of doing this, for multiple data sets and over the range of values for β, is referred to as sampled cluster entropy and is shown in Figure 4.2. In the figure, the x-axis represents β/β^*.

Observe that for all data sets, this measure exhibits a clear minimum around $\beta = \beta^*$. It turns out that the relative ordering of the heterogeneity values at this value of β correctly reflects the underlying heterogeneity in the data.

The above measure appears quite suitable as a measure of heterogeneity. However, computing it requires at least two parameters; the threshold parameter that decides when the clusters are merged, and a parameter deciding the number of samples required to get an accurate estimate. In the absence of a rigorous theoretical analysis, the choice of values for these parameters will inevitably need to be determined experimentally.

Using Mutual Information $I(T; X)$

A more compact solution exploits the relationship between $H(T)$ and $I(T; X)$. Rate-distortion theory tells us that $I(T; X)$ is a measure of compression

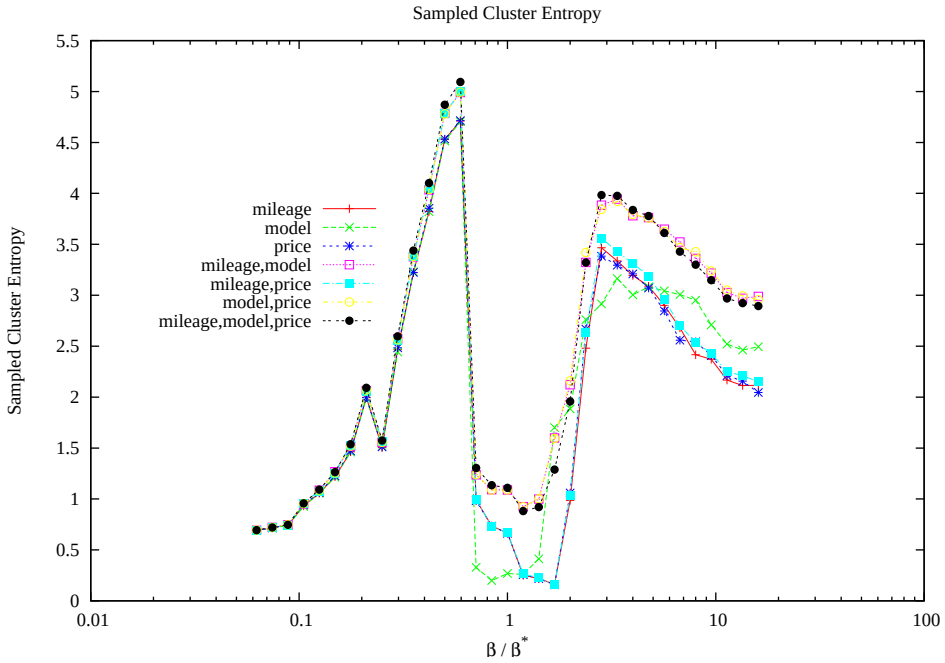


Figure 4.2: Sampled cluster entropy of the mixture from columns `mileage`, `model` and `price` in table `autosforsale-2` as a function of β/β^* on a log-scale.

complexity; it is not hard to see that if the assignments $p(t|x)$ are 0-1, $H(T|X) = p(x_i)[-\sum_{j=1}^{k} p(t_j|x_i)\log p(t_j|x_i)] = 0$, then

$$I(T;X) = H(T) - H(T|X) = H(T)$$

Thus, the measure of heterogeneity (and of cluster entropy for a soft clustering) we propose is $I(T;X)$.

A brief illustration of this idea is presented in Figure 4.3. For each value of β and for multiple data sets, we plot $I(T;X)$ for the soft clusterings obtained using `iIB`. On the x-axis, we plot β relative to the chosen value $\beta^* = \frac{H(X)}{I(X;Y)}$. As β increases from zero, there is a sharp increase in $I(T;X)$ as β approach β^*. This increase occurs for all the data sets, and in roughly the same place.

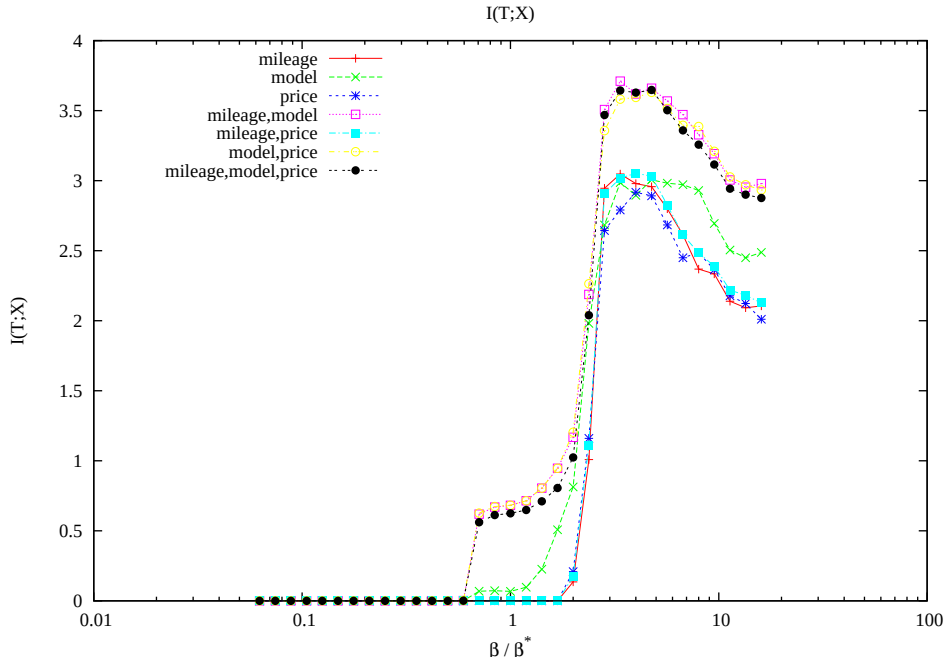


Figure 4.3: $I(T;X)$ of the mixture from columns `mileage`, `model` and `price` in table `autosforsale-2` as a function of β/β^* on a log-scale.

The measure of heterogeneity of a data set is then the value of $I(T;X)$ at the point $\beta = \beta^*$ (i.e at $\beta/\beta^* = 1$ on the graph). As we shall see experimentally in Section 4.5, this measure of heterogeneity correctly predicts the true underlying heterogeneity in the data.

$I(T;X)$ increases monotonically; sampled cluster entropy as described above exhibits a series of (increasing) local minima. It is interesting, and an indication of the robustness of our proposed choice of β, that both measures, while approaching the problem of heterogeneity differently, display the same properties: namely, a sharp change close to $\beta = \beta^*$, and a correct ordering of data sets with respect to their true underlying heterogeneity at this point.

We can now summarize the entire algorithm in Algorithm 1.

Algorithm 1 Computing Heterogeneity.

1: Convert input strings to distribution of q-grams
2: Compute the weight for each q-gram
3: Compute the weighted probabilistic vector $p(Y|x)$ in the space of q-grams for each value x
4: Add the background if necessary
5: Compute $\beta^* = \frac{H(X)}{I(X;Y)}$
6: Compute soft clustering using `iIB` with β set to β^*
7: Estimate the heterogeneity by computing $I(T;X)$

Given our choice of $I(T;X)$ as the preferred measure of heterogeneity and the cluster entropy of a soft clustering, when we refer to cluster entropy in the rest of this chapter, we mean $I(T;X)$, unless otherwise indicated.

4.4 Methodology

The previous sections described our general framework for quantifying column heterogeneity. We now present an instantiation of our framework based on the representation of string data using a multinomial distribution of q-grams, and the details of our algorithm implementation. The input to the algorithm consists of a column of data, viewed as a set of strings X.

4.4.1 Preparing The Data

Weighted q-gram Vectors

We treat all data as strings, regardless of content. We represent strings using q-gram distributions. For each string, we extract all its q-grams, and construct a histogram, with the entry for each q-gram recording its frequency.

Using a large value of q seemingly allows us to capture more sophisticated patterns in the data; however, since the number of potential q-grams (the number of "dimensions") grows exponentially with q, the effect of these (fewer number of) q-grams is muted. Algorithms that manipulate these representations will also perform much worse as q increases. We have discovered in our experiments that setting $q = 2$ provides a good balance between the efficiency of the computation and the accuracy of the results. Therefore, we construct $1-$ and $2-$grams for all strings.

Let the set of q-grams be Y. For each q-gram y, let $f(x, y)$ be the number of occurrences of y in x, and let $p(y)$ be the fraction of strings containing y. We construct a matrix S whose rows are the strings of X and whose columns are q-grams, and the entry $m_{xy} = \frac{f(x,y) \cdot w(y)}{Z}$, where Z is a normalizing constant so that the sum of all entries in M is 1, and

$$w(y) = H(p(y)) = -p(y) \log p(y) - (1 - p(y)) \log(1 - p(y))$$

Note that setting $w(y) = -\log(y)$ would yield the standard IDF weighting scheme.

In information retrieval, where q-gram representations are most prevalent, individual terms are weighted based on measures like *term frequency (t.f)* and *inverse document frequency (i.d.f.)*. The primary purpose of information retrieval systems is search, and so rare terms (having a large i.d.f.) are important to reduce the size of search results.

Note that standard IDF weighting is not appropriate for our application. IDF weighting was designed to aid in document retrieval, and captures the idea of "specificity", that a few key phrases could be very useful at identifying relevant documents. IDF weighting captures this by over-weighting terms with low relative occurrence.

However, for heterogeneity testing, specificity means that certain rare

terms occur only in a few strings. We are not concerned with such outliers, and IDF weighting will only give such strings artificially high importance. Rather, robust heterogeneity testing requires us to to identify terms that distinguish large sets of data from each other; the entropy weighting scheme captures this as it is maximized when $p = 0.5$, and decays symmetrically in either direction.

Adding Background Context

Any clustering makes sense within a context; a high concentration of points in a small range is significant only if viewed against a relatively sparse, larger background. For example, the collection of strings in Table 4.1(a) form a cluster only with respect to the set of all strings. If the background for this data is only the set of email addresses, then this set has no apparent unusual properties.

For heterogeneity testing, an appropriate background is the space of all strings. This needs to be introduced into each data set in order to define the "bounding volume" of the space. As we represent data as distributions, the background consists of random distributions, chosen from the space of all distributions. These are added to the data before soft clustering is performed, and are then removed[1].

It is well known that the uniform distribution over the d-dimensional simplex is a Dirichlet distribution, and thus a uniform sample from this space is obtained by the following a simple procedure. Sample d points $x_1, x_2, \ldots x_d$ from an exponential distribution with parameter 1, and normalize the values by dividing each by $\sum_{i=1}^{d} x_i$. The resulting d-vector is a uniform sample from

[1]This is reminiscent of the subtractive dithering used in signal processing to improve quantization.

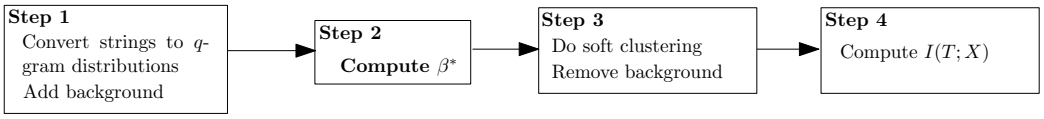


Figure 4.4: A flowchart for heterogeneity estimation.

the simplex [Dev86]. A uniform sample from an exponential distribution is computed by sampling r uniformly in $[0 \ldots 1)$ and returning the value $\ln(1/r)$.

To generate the background, we use a set of q-grams disjoint from the q-grams in Y, of the same cardinality as Y. Using the above procedure, we generate $|X|$ points, yielding a matrix N that is then normalized so all entries sum to 1. Both S and N have dimension $|X| \times |Y|$.

We fix a parameter $0 < \lambda < 1$ (the *mixing ratio*) that controls the mixture of data and background context. The final joint density matrix M is of dimension $2|X| \times 2|Y|$, containing λS as its first $|X|$ rows and $|Y|$ columns and $(1 - \lambda)N$ as its last $|X|$ rows and $|Y|$ columns. Note that M is a valid joint distribution since its entries sum to 1. We will use notation and refer to the rows of M as X and the columns as Y in what follows.

4.4.2 Computing β^* And Clustering the Data

In Section 4.3.2, we derived an expression for the value of β corresponding to the point on the rate-distortion curve that balanced error and compression. This value of β is given by $\beta^* = \frac{H(X)}{I(X;Y)}$. We compute this from the matrix M, using only the data rows and columns. We use the standard empirical estimator for entropy (which treats the normalized counts as fractions).

Given M and β^*, we now run the `iIB` algorithm [Slo03] for computing the information bottleneck. This algorithm is a generalization of the standard expectation-maximization method. Although the algorithm generates a soft

clustering, it requires as input a target set of clusters (not all of which may be used in the output). We specify a very large number of clusters ($|X|/2$). Empirically, we see that this is sufficient to find a point on the rate distortion curve. We emphasize here that we *do not* need to fix a number of clusters in advance; the number of clusters that we supply to `iIB` is merely an artifact of the implementation and only needs to be a very loose upper bound. It only affects the running time of the algorithm, and not the final heterogeneity measure computed.

The output of this algorithm is a soft clustering T, specified as the conditional probabilities $p(t|x)$, from which the cluster masses $p(t)$ and the cluster centers $p(y|t)$ can be derived using Bayes' Theorem and the conditional independence of T and Y given X, as defined in Equation 4.3 and Equation 4.4.

We depict the algorithm in Figure 4.4, paralleling the steps of algorithm 1.

4.5 Experiments

4.5.1 Datasets Description

IBM Many Eyes Dataset

The main set of data is obtained from `IBM Many Eyes`[2], by IBM Research and the IBM Cognos software group. As any user in the world is allowed to upload their data, `Many Eyes` thus have many heterogeneous tables of different schemas, and becomes a good dataset for us to study on. We downloaded more than 47,000 tables, from which, we selected those relevant to automobiles, e.g., records about car models, sales of cars, fuel economy statistics, carbon dioxide emissions. Out of these tables, we removed those without

[2]http://www-958.ibm.com/software/data/cognos/manyeyes/

significant number of rows, and has a result of 100 tables. These 100 tables contribute 1219 columns and approximately 1 million values (989 thousand) in total. For all the experiments in this thesis, we will use the selected columns from `Many Eyes` dataset to validate.

For this chapter, we artificially mix a few syntactic types from table `autosforsale-2` into one column, so that the controlled heterogeneity is available to us, to validate the proposed measure of heterogeneity.

4.5.2 Validation of Choice of β^*

The first validation is on the choice of β^*. Figures 4.2 and 4.3 have shown the effects of sampled cluster entropy and our proposed heterogeneity measure, $I(T;X)$, while changing β is a log scale. When β is near to β^*, the sampled cluster entropy reaches its local minimum, and $I(T;X)$ shows the clusters start splitting near β^*, as $I(T;X) \approx 0$ means clusters are identical.

4.5.3 Validation of Heterogeneity Measure

We now demonstrate the value of the cluster entropy $I(T;X)$ near $\beta^* = \frac{H(X)}{I(X;Y)}$.

Figure 4.5 plots $I(T;X)$ as a function of $\frac{\beta}{\beta^*}$ ranging from 0.95 to 1.05. The mixture column is mixed with equal proportion from the three columns: `mileage`, `model` and `price`.

Note there is a clear distinction between the following two groups:

1. 1-mixtures: {`mileage`}, {`model`}, {`price`}, and 2-mixture {`mileage, price`}
2. 2-mixtures {`mileage, model`}, {`model, price`} and 3-mixture {`mileage, model, price`}

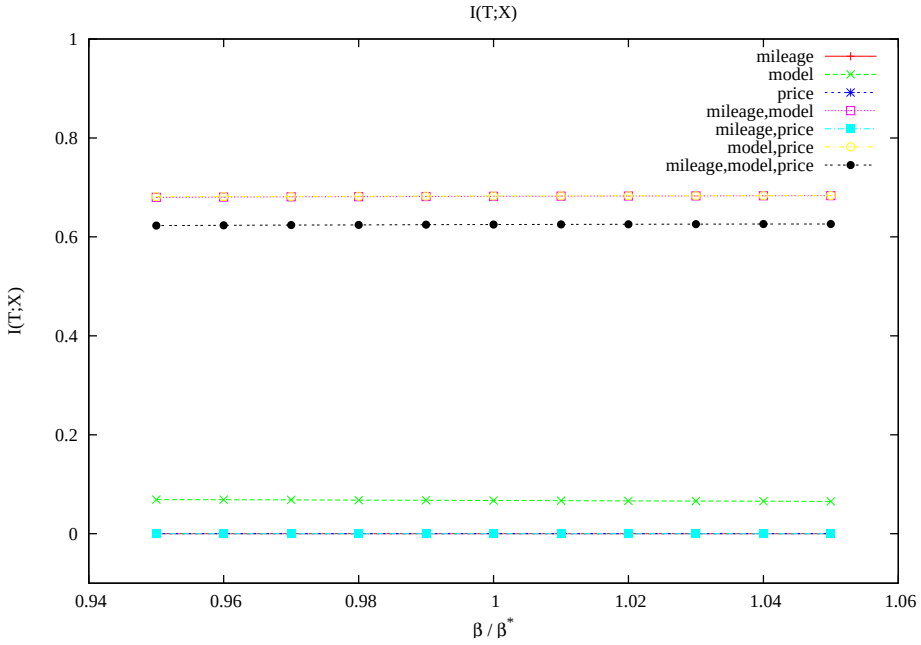


Figure 4.5: The heterogeneity measure of a column mixed from equal amount of `mileage`, `model` and `price` near β^*

The reason is as follows. Both `mileage` and `price` are of the same syntactic type, and in the table `autosforsale-2`, both are digits with commas serving as delimiters for every three digits. Therefore, it is not surprising that `mileage` and `price` are of the same syntactic type, and when they are mixed together, our heterogeneity measure marks the mixture as long as those 1-mixtures.

However, when either `mileage` or `price` is mixed with `model`, both the heterogeneity measures increase significantly. This is due to the fact that `model` is of a very different syntactic type from either `mileage` or `price`. The curve for the only 3 mixture {`mileage, model, price`} is actually lower than the two 2-mixtures with `model`. This is determined by the proportion of the two syntactic types, which reflects our requirement for heterogeneity

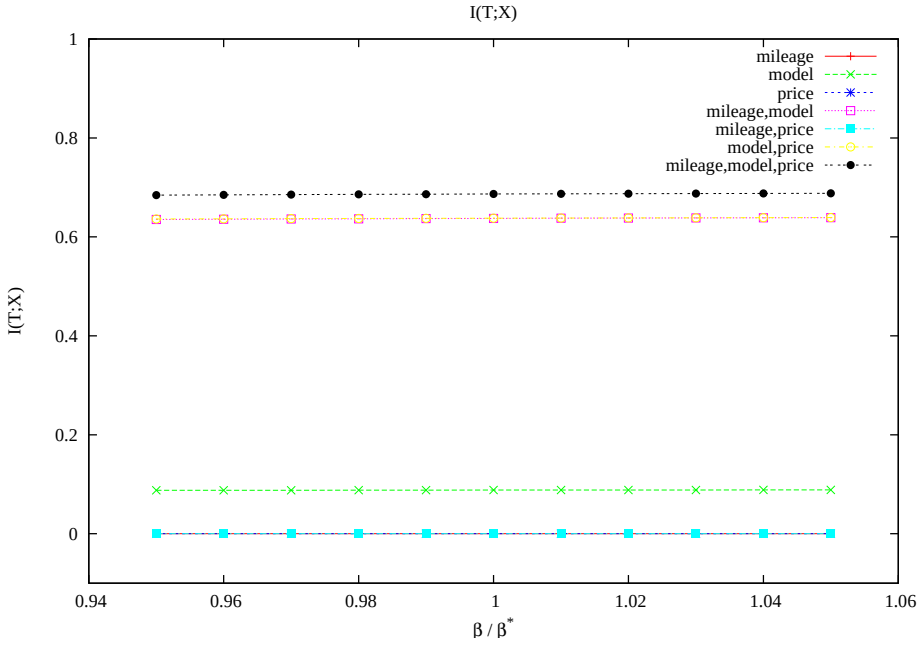


Figure 4.6: The heterogeneity measure of a column mixed from `mileage`, `model` and `price` in the ratio 1:2:1 near β^*

measure to capture the distribution of the types in Section 4.1. As the 2-mixture {`mileage, model`} consists of half of the two syntactic types each, the uncertainty about the syntactic types is more than the uncertainty of the mixture, where $\frac{2}{3}$ of the values are of the same type with the remaining $\frac{1}{3}$ of the other type, which is the case for 3-mixture {`mileage, model, price`}.

However, if we change the ratio of the mixture columns, for example, we mix the three types, `mileage`, `model` and `price` in the ratio of 1:2:1, the heterogeneity measure is maximum for the 3-mixture {`mileage, model, price`}, as shown in Figure 4.6, since each syntactic type contributes half.

If we further increase the proportion of `mileage` in the mixture, as shown in Figure 4.7, we will have similar cluster entropy for the 2-mixture {`model, price`} and the 3-mixture {`mileage, model, price`} as both are

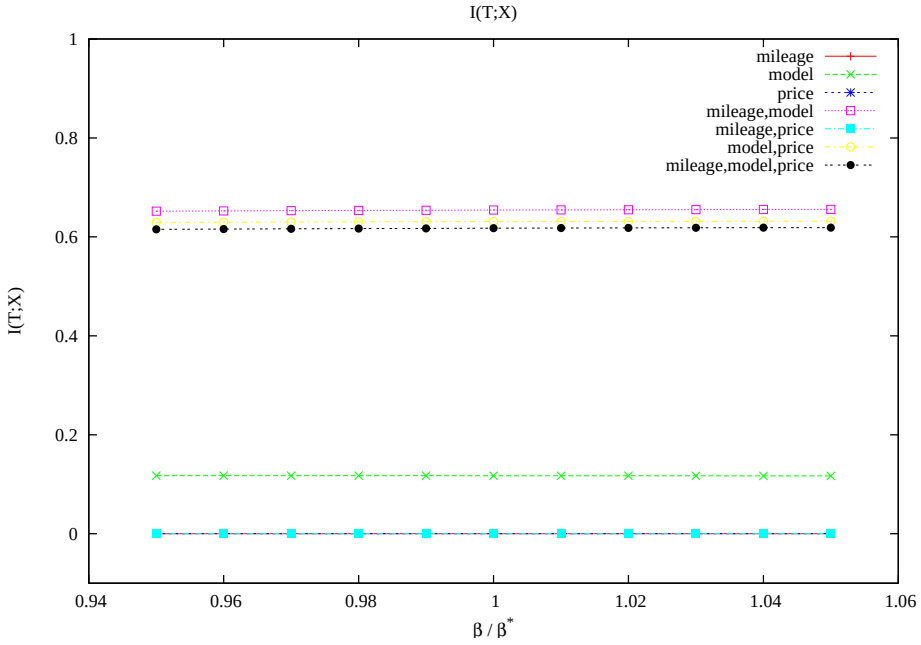
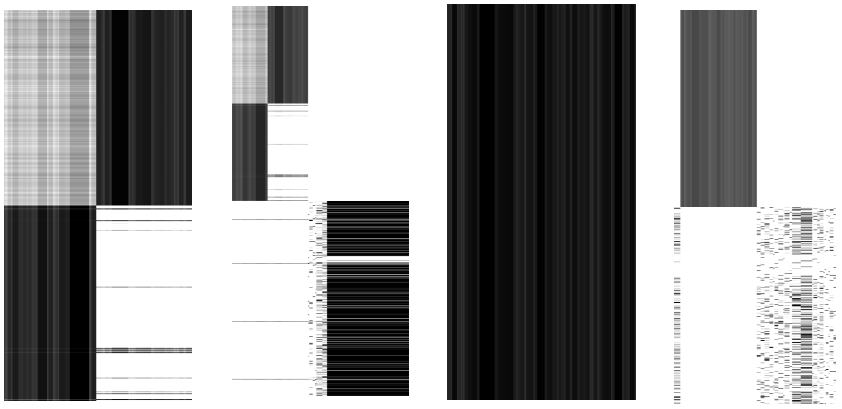


Figure 4.7: The heterogeneity measure of a column mixed from `mileage`, `model` and `price` in the ratio 3:2:1 near β^*

distributed in $\frac{1}{3}$ and $\frac{2}{3}$.

4.5.4 Validation of Soft Clustering

The cluster entropy of soft clustering, $I(T;X)$ appears to capture our intuitive notion of heterogeneity. However, it is derived from a soft clustering returned by the `iIB` algorithm. Does that soft clustering actually reflect natural groupings in the data? It turns out that this is indeed the case. In Figure 4.8, we display bitmaps that visualize the soft clusterings obtained for different mixtures. In this representation, columns are clusters, rows are data, and darker probabilities are larger. For each data value, its corresponding row in the bitmap then represents its cluster membership distribution,



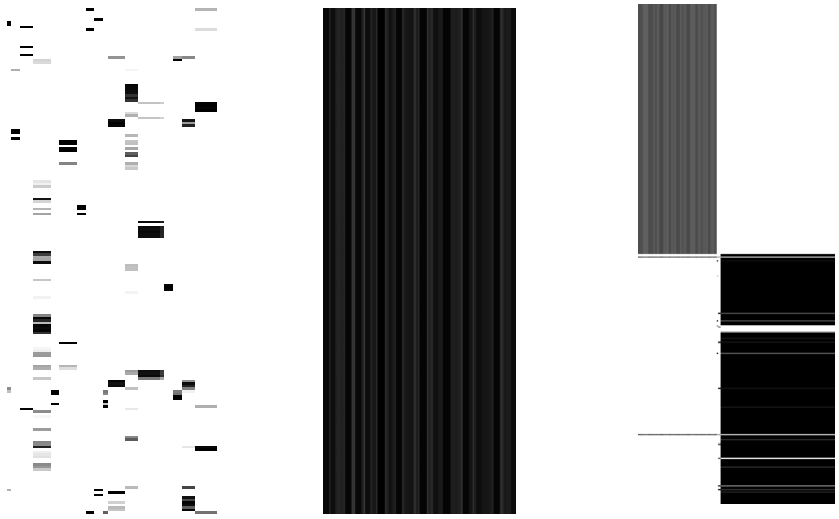
(a) `mileage, model` (c) `mileage, price`

(b) `mileage, model`, Background (d) `mileage, price`, Background

Figure 4.8: The soft clustering of columns mixed by {`mileage, model`} and by {`mileage, price`}

i.e., $p(t|x)$ for each $x \in \mathcal{X}$. A collection of data values having the same cluster membership distributions represents the same cluster. For clarity, we have reordered the rows so that all data values coming from the same source are together, and we reordered the columns that have similar $p(t|x)$ distributions.

The two bitmaps in Figure 4.8(a) and (b) (bitmap (b) has incorporated with the soft clustering of the background as well) are the soft clustering of the mixed column from `mileage` and `model` from table `autosforsale-2`. Figure 4.8(a) and (b) show that the clusters separate out quite cleanly, clearly displaying the different number of sources in the mixture. Note that this number is the natural number of cluster `iIB` achieved, without having to specify k. However, we do not observe a separation from bitmaps in in Figure 4.8(c) and (d), as the syntactic types of `mileage` are similar to that of `price`. Thus values from `mileage` have similar cluster membership distributions with values from `price`, reinforcing our observation that they form two



(a) No Background (c) With Background

(b) With Background (Data only)

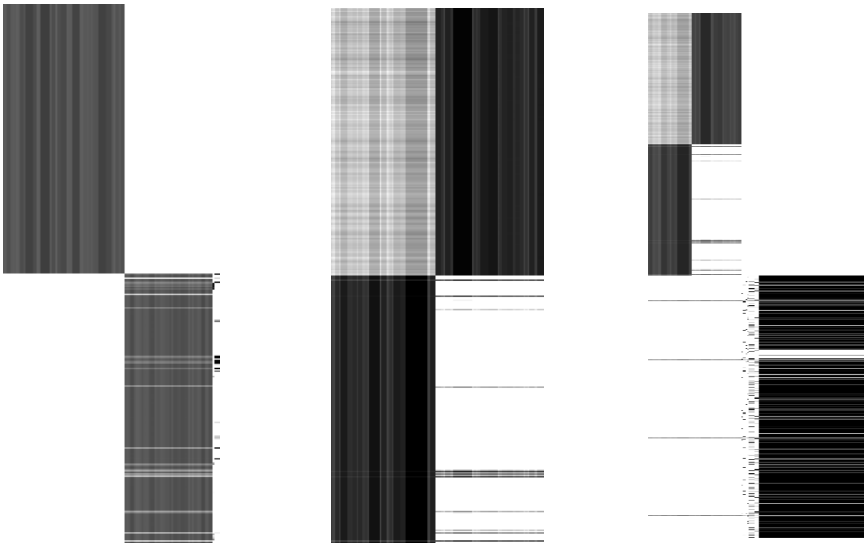
Figure 4.9: The soft clusterings of the column `model` without and with background

very close (not well-separated) clusters.

In addition to performing soft clustering and computing cluster entropy, our algorithm adds in background context to aid in the clustering process. In the next experiment, we establish the need for this step.

4.5.5 Validation of Background Addition

An important aspect of our algorithm is the addition of a background context, as discussed in Section 4.4.1. We argued that intuitively, the effect of this addition is to "expand" the space being clustered, so a set of points that is highly clustered shows up clearly in contrast to the background. Obviously, if the background level is too high, any data will get swamped, and if too low, will be useless.



(a) No Background (c) With Background

(b) With Background (Data only)

Figure 4.10: The soft clusterings of the mixture {`mileage, model`} without and with background

We further illustrate this effect by Figure 4.9 and Figure 4.10. As predicted, when a background context is not added, the data spreads over all clusters in unpredictable ways, as shown in Figure 4.9(a). Adding background in immediately and predictably collapses the data into a few clusters, as in Figure 4.9(c). The cluster distributions for the original data values, Figure 4.9(b), become very homogeneous, showing the data values form one cluster. As two sets of data with different syntactic types naturally provide this contrast to each other, the background seems unnecessary, as shown in Figure 4.10. However, without knowing the syntactic types, adding background is hence necessary to prevent the above situation.

4.5.6 Performance

The experiments are carried on a machine with Intel Core 2 Duo CPU at the speed of 2.53GHz. For each mixture in Table 4.2, we sampled 200 rows, and presented them using 1-grams or using both 1-grams and 2-grams. We then added in 200 rows of background with $\lambda = 0.5$. The run time for each `iIB` iteration is listed in Table 4.2.

Mixture	Time per iIB iteration	
	1-grams	1-grams and 2-grams
mileage	0.15	0.94
model	0.37	4.82
price	0.16	0.96
mileage, model	0.39	4.98
mileage, price	0.16	0.97
model, price	0.40	5.03
mileage, model, price	0.40	5.05

Table 4.2: Runtime (in seconds) of the different mixtures for each iteration

Mixtures with `model` took more time for each iteration as the type `model` generates more q-grams than the others. We can improve the performance by retaining only important q-grams, which will be discussed in Chapter 6.

4.6 Summary

In this chapter, we proposed the column heterogeneity measure, as the mutual information between the data values and a particular soft clustering on their types. This particular soft clustering is computed at an optimal trade-

off between clustering space and clustering quality. Our technique treats all data values as weighted vectors of q-grams, and clusters them in a probabilistic space. Background context is added to provide contrast to the clusters, in order to identify columns with just one type of values.

However, we did not give a solution on how to stop columns from becoming more heterogeneous. In the next chapter, we will propose a solution to prevent forming more heterogeneous columns when multiple columns are integrated together.

Chapter 5

Inter-column Heterogeneity (I): Multi-column Heterogeneity

In Chapter 4, we discussed the intra-column heterogeneity, i.e. the heterogeneity among values within the same database column. From this chapter onwards, our focus is shifted to heterogeneity across multiple database columns. Conventional data integration techniques often require columns to be integrated to be homogeneous, and perform column matching based on the similarities between such homogeneous columns. However, database columns are not always homogeneous, thus it is necessary to study whether heterogeneous columns can be integrated. On the other hand, some database columns are not so homogeneous as what a database administrator or an integration tool thinks. Their decisions may be made by looking at the column names or sampling a few values from the columns. A matching validator is therefore needed to evaluate whether a match between multiple columns makes sense. This chapter addresses the problem of validating matching among multiple possibly heterogeneous columns. We will begin this chapter by discussing the necessity of validating schema matching.

5.1 Validating Schema Matching: the Motivation

Schema matching is an important problem in the realm of database integration. The basic operator is called *Match* [RB01], and associates schema elements from one database with schema elements from another database, as a prelude to schema mapping and database integration. Early approaches to schema matching were based on identification using only schema labels. This was followed by techniques that took into account the actual data in the columns being matched. When schema matching is performed between databases that are likely to contain the same data (e.g., the catalogs of different on-line book sellers), the matching can check for occurrences of the same values across the databases. To allow for the possibility that schema matching is performed between databases that contain different data (as is more commonly the case), there has been work on describing data in a column via aggregate distributions and using inferences on these distributions to identify matches. Schema matching has numerous applications, and is an area of extensive study; surveys by Rahm and Bernstein [RB01] and Doan and Halevy [DH05a] cover the major issues in this area, as well as much of the relevant literature.

Essential to the success of schema matching (and hence of database integration) is a *validator* that, given a candidate match, declares it to be valid or not. If a schema matching is declared to be invalid, a good validator should provide a "certificate of invalidation". It is harder to provide a "proof" of a valid matching, since such a proof merely implies the lack of contradicting evidence. Every technique for discovering schema matchings in the literature implicitly includes a validator, possibly in conjunction with a search strategy

for identifying candidate matchings. In this chapter, we focus on the validation aspect of schema matchings and hence our technique can be combined with any search strategy that helps identify candidate matchings.

5.1.1 String Values and Syntactic Types

Surprisingly, despite the prevalence of string values in databases (e.g., person names, company names, building locations, email addresses), and the vast amount of work in schema matching, little attention has been paid to validating matchings of string-valued schema elements. Exceptions include systems like CUPID [MBR01] and the work by Embley *et al.* [EXD04] that consider multi-column matchings of the form "concatenate FN and LN" (i.e., full string concatenation), and the work by Warren and Tompa [WT06] who study the discovery and validation of multi-column sub-string matchings of the form "concatenate the first 7 characters of LN and the first character of FN". A limitation of these previous techniques is that their validator can be used only in database integration scenarios where matching schema elements have the *same values*, but not in the more common case where matching schema elements have different string values (for example, when two merging companies integrate their customer databases).

The central idea of this chapter is based on the intuition that the set of string values in a column has a syntactic type that is characterized by the distributions of q-grams (short sub-strings) of the string data in that column. Hence, columns related by a schema matching are expected to have a similar syntactic type, even when they have different values. Comparing two string-valued columns then becomes the problem of estimating a distance between the distributions corresponding to their syntactic types. Building on these intuitions, this chapter addresses the challenging problem of validating

americaDB				
propId	propTitle	piFN	piLN	coPiName
2005CN0734	who likes xml	john	pardon	wei li
2006CN0732	what's not to like about xml	johny	walker	wenyuan dai

chinaDB		
proposalInfo	piName	coPiName
1136-AA-654 2005 i like xml too	li mingfei	johnston christopher
1136-B-45 2006 the future of xml	yu kai	pitts daniel

Table 5.1: `americaDB` and `chinaDB`: Schema and Sample Data

multi-column schema matchings by syntactic type of their string values.

5.1.2 Illustrative Example

Validating by type forces us to revisit some of our basic ideas about matching (and mapping) schemas. Consider two databases maintaining information about collaborative multi-national research proposals, `americaDB` and `chinaDB`: `americaDB` tracks all research proposals where the primary investigator is based in the US, and `chinaDB` tracks all research proposals where the primary investigator is based in China. Their schemas and some sample data are given in Table 5.1.

Note that all the data in these database are represented as strings. This is fairly prevalent in real databases, due to the flexibility afforded by the use of strings. When exploring and validating schema matchings between these two databases, possibly to create an integrated target database `chinamericaDB` with the schema (`propTitle`, `americaPerson`, `chinaPerson`), there are several issues that need to be addressed.

No guarantee of common data

There is no guarantee that the principal investigators or co-principal investigators in one database will be present in the other database. Thus validating a candidate schema matching cannot be based on entire string values. However, there is the possibility of using q-grams to aid in the schema matching. Note (in the data of Table 5.1) that 2-grams like "jo" and "on" seem to occur more commonly in American names while 2-grams like "ai" and "ei" seem to occur more commonly in Chinese names. By looking at 2-grams, one could invalidate a candidate matching between `americaDB.coPiName` and `chinaDB.coPiName`.

Simple and composite matchings

This example illustrates the presence of both simple and composite matchings. For example, the matching between `americaDB.coPiName` and `chinaDB .piName` is a simple 1-1 matching, and should be validated. Since the integrated database `chinamericaDB` uses single columns to represent both American names and Chinese names, the validator would need to be able to validate a multi-column composite matching between `concat (americaDB .piLN, americaDB.piFN)` and `chinaDB.coPiName`, but invalidate a candidate multi-column composite matching between `concat (americaDB.piFN, americaDB .coPiName)` and `chinaDB.coPiName`.

Sub-string matchings

Since `propId` is not part of the target schema, in order to populate attributes like `propTitle` in the integrated database `chinamericaDB`, one would need to validate a matching between `americaDB.propTitle` and sub-strings from `(chinaDB.proposalInfo)`, i.e., `sub-string (chinaDB.proposalInfo)`. How-

ever, if there is a candidate matching between `americaDB.propTitle` and `chinaDB.proposalInfo`, it should not be validated.

5.1.3 Integratability

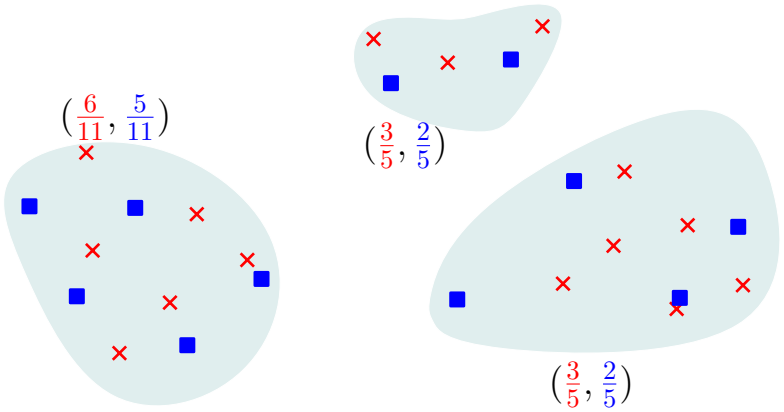
How then do we determine if two string-valued columns may be matched? To do this, we introduce the notion of *integratability*. Suppose that we are trying to determine whether the data in columns C and C' possess the same syntactic type, and thus are candidates for matching/integration. Our solution is the following simple idea:

> If the syntactic type structure of the columns C and C' is similar, then in any reasonable clustering, data from both columns will populate each cluster in roughly the same ratios.

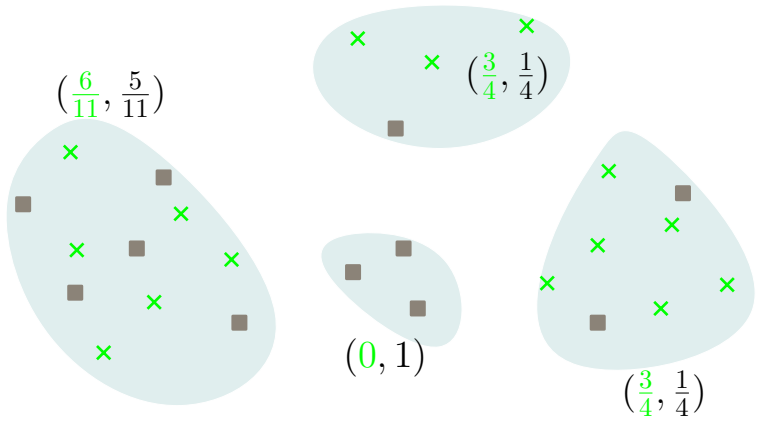
An example of this idea is shown in Figure 5.1. In both panels, the data is depicted geometrically, with crosses representing data from one column, and squares from the other. In the left hand panel, the two columns exhibit similar type distributions, and a reasonable clustering puts the same relative fractions of data from the two columns in each cluster. Note that unless the two columns have exactly the same number of items, these ratios need not be balanced (i.e., 50-50). All we expect is that the proportion is relatively constant across clusters.

In the right hand panel, the data from the two columns separate out into different clusters, and thus there are clusters with widely varying fractions of crosses and squares.

This notion can be made precise in an information-theoretic manner; in a later section we will see that this idea translates into computing a certain kind of conditional entropy. A key aspect of this notion is that it allows us to



(a) Each cluster contains roughly the same proportion of the two data sets.



(b) The cluster vary greatly in the relative ratios of the two data sets.

Figure 5.1: Computing integratability between two sets of data. In each case, a candidate clustering is given. The two sets are marked with crosses and squares.

exploit *context*; since the data from both columns are clustered together, the process discovers commonalities between the data that may not be apparent if the two columns were clustered independently. It also allows us to quantify the degree of integratability in a purely generic manner, with no recourse to metric spaces and embeddings of data.

How do we find such a reasonable clustering? There are different ways one might go about this, and indeed our approach does not rely on the use of a particular clustering method. However, our method is most powerful when the clustering itself imposes no additional constraints (like a fixed number of clusters, or even a hard assignment of items to clusters). Thus, in order to be as general as possible, we will use a soft clustering formulation; since any hard clustering can be viewed as a special soft clustering, this does not restrict us in any way.

To compute this soft clustering, we will exploit the `iIB`-based clustering in Chapter 4, where we have derived a measure of heterogeneity based on information-theoretic considerations, and compute a soft clustering of data as a by-product.

5.1.4 Validating Sub-string Matches

The second component of our type-based schema matching approach is a method for detecting semantic types embedded in string data. String data may have complex internal structure, and it is often the case that a schema match arises by splitting a string value into sub-strings, and matching the sub-strings with elements from another schema. Formally, this means that we need some kind of sub-string extraction oracle that, when applied to a column of string-valued data, returns a new column on which we can run an integratability check. However, the only information we have regarding the

location of these sub-strings is the column we wish to integrate with.

Our idea is thus as follows: rather than examining strings in the column for obvious internal delimiters that might mark the boundaries of a relevant sub-string, we use the target column to find such sub-strings. We use type information extracted from this column to segment each string and return the most promising candidate sub-string. We do this by evaluating the likelihood of individual q-grams of the string appearing in strings from the target column. If we find a set of contiguous q-grams that have an *unusually high* likelihood, we have found our desired candidate sub-string.

5.1.5 Validating Composite Matchings

Extending schema matchings and mappings to types allows us to capture composite associations that would be difficult via value-based methods. Certain composite associations can reduce to a simple matching problem. For example, determining whether the concatenation of data from two columns matches with data from another column reduces to validating a simple match.

Consider however, the problem of determining whether one set of columns *jointly* matches another set of columns. In this case, there are really two underlying problems. Suppose the joint matching is intrinsically unordered, in the sense that the order of composition of columns doesn't matter when checking for a match. In our example from Section 5.1.2, this happens when matching names; since labeling of names as *first* and *last* may be inconsistent, especially when dealing with names from Asian countries. In this case, any approach based on concatenation of column values will struggle to find a match, because there may be no consistent way of ordering data to find a match. Using our distributional approach however, we are indifferent to the issue of finding orderings. We can concatenate the columns and do a

type-based validation for the super-columns; the q-gram extraction approach allows us to identify common regions without needing to consider their location.

Conversely, if the data is ordered (and thus column data can be concatenated to form one super-column), the problem does not reduce to validating a single match by type for the reasons outlined above. But it also cannot be reduced to a set of simple matches because distributional equality on the marginals does not imply distributional equality on the joint distribution. Here, introducing *positional information* helps us identify such matches. This allows us to generalize our type-based validation to composite matching as well.

5.1.6 Invalidation Certificate

Recall that if a schema matching is declared to be invalid, a good validator should provide a "certificate of invalidation". The final component of our approach is a certificate of invalidation, presented as a "bar code", which will come from the clustering itself. A low integratability score indicates that the data from the two columns separate into different clusters in our soft clustering, which can be presented to the user.

In this chapter, we will first present a generic type-based measure of matching among multiple columns, based on information-theoretic considerations, and an algorithm to compute this measure in Section 5.2. In Section 5.3, an algorithm for "splitting" the string values in a column is proposed to identify sub-strings that are likely to match with the values in another column. This is then followed by Section 5.4, which describes procedures for validating single column and multi-column schema matchings based on the tools developed above. If the algorithm fails to validate a schema matching,

an invalidation certificate is provided, which is discussed in Section 5.5. Section 5.7 will give an experimental study that demonstrates the effectiveness of our technique.

5.2 A Measure of Integratability

Let $\{C_1, C_2\}$ be two columns of data that we wish to integrate based on their syntactic types. A soft clustering T on the union of all columns, i.e., $C_1 \cup C_2$, associates one conditional probability distribution $p(t|x)$ for each value $x \in C_1 \cup C_2$, which represents the syntactic type of value x. Considering the union of the columns to have unit probability mass, we then define

$$p(C_h) = \frac{|C_h|}{|C_1| + |C_2|} \qquad h = 1, 2$$

where $|C_h|$ is the *weight* of column C_h, which can be set equal to its cardinality if all items are considered equally significant.

As $p(t|x)$ represents the syntactic type of value x, the syntactic type of column C_h is thus an aggregation over all $x \in C_h$, i.e., the *syntactic type distribution* for each column C_h is computed as

$$\begin{aligned} p(t|C_h) &= \frac{p(t, C_h)}{p(C_h)} = \frac{\sum_{x \in C_i} p(t, x)}{p(C_i)} \\ &= \frac{\sum_{x \in C_i} p(x) p(t|x)}{p(C_i)} \end{aligned} \tag{5.1}$$

By our informal definition of integratability, the two columns have the same type structure if the two type distributions are similar. A standard measure for comparing distributions p and q is the *Kullback–Leibler divergence (KL divergence)* $D_{KL}[p\|q]$. This distance is asymmetric and can be unbounded if the target distribution (i.e. p) support is not a subset of the source distribution (i.e. q) support. Also, we would like to incorporate the relative

mass difference between $p(C_1)$ and $p(C_2)$. Therefore, we will use a symmetric variant, the *Jensen–Shannon divergence*, defined for $\alpha + \beta = 1, \alpha, \beta > 0$ as

$$D_{JS}(p, q, \alpha, \beta) = \alpha \cdot D_{KL}[p\|m] + \beta \cdot D_{KL}[q\|m] \tag{5.2}$$

where $m = \alpha \cdot p + \beta \cdot q$.

The Jensen–Shannon divergence can thus be interpreted as the average distance to the (weighted) centroid of the two distributions. The distance between the two type distributions is then $D_{js}(p(t|C_1), p(t|C_2), p(C_1), p(C_2))$.

Lemma 5.1. *If $\alpha = p(C_1)$, $\beta = p(C_2)$, and $p(t|C_h)$ are defined as Equation 5.1 for $h = 1, 2$, then $\alpha \cdot p(t|C_1) + \beta \cdot p(t|C_2) = p(t)$.*

Proof. $\forall t_j \in T$

$$\begin{aligned} \alpha \cdot p(t_j|C_1) + \beta \cdot p(t_j|C_2) &= p(C_1)p(t_j|C_1) + p(C_2)p(t_j|C_2) \\ &= p(t_j, C_1) + p(t_j, C_2) \\ &= p(t_j) \end{aligned}$$

Therefore,

$$\alpha \cdot p(t|C_1) + \beta \cdot p(t|C_2) = p(t)$$

□

Theorem 5.2. *Given two columns C_1 and C_2, the JS divergence between the syntactic types of column C_1 and C_2, as defined in Equation 5.2, is the mutual information between the syntactic types and the column set $C = \{C_1, C_2\}$; i.e.,*

$$D_{JS}(p(t|C_1), p(t|C_2), p(C_1), p(C_2)) = I(C; T)$$

Proof. Let $\alpha = p(C_1)$ and $\beta = p(C_2)$, by Lemma 5.1,

$$\alpha \cdot p(t|C_1) + \beta \cdot p(t|C_2) = p(t)$$

Therefore,

$$\begin{aligned}
& D_{JS}(p(t|C_1), p(t|C_2), p(C_1), p(C_2)) \\
= \; & \alpha \cdot D_{KL}[p(t|C_1)\|p(t)] + \beta \cdot D_{KL}[p(t|C_2)\|p(t)] \\
= \; & p(C_1) \sum_{j=1}^{k} p(t_j|C_1) \log \frac{p(t_j|C_1)}{p(t_j)} + p(C_2) \sum_{j=1}^{k} p(t_j|C_2) \log \frac{p(t_j|C_2)}{p(t_j)} \\
= \; & \sum_{j=1}^{k} p(t_j, C_1) \log \frac{p(t_j, C_1)}{p(t_j)p(C_1)} + \sum_{j=1}^{k} p(t_j, C_2) \log \frac{p(t_j, C_2)}{p(t_j)p(C_2)} \\
= \; & \sum_{j=1}^{k} \sum_{h=1,2} p(t_j, C_h) \log \frac{p(t_j, C_h)}{p(t_j)p(C_h)} \\
= \; & I(C;T)
\end{aligned}$$

□

Intuitively, $I(C;T)$ is larger when the distributions are more distinct, which is to say the two columns are less integratable. $I(C;T)$ has a maximum value of $\min(H(C), H(T))$. In fact, since $I(C;T) = H(C) - H(C|T)$, we can write a normalized expression for $I(C;T)$ as

$$\frac{I(C;T)}{H(C)} = 1 - \frac{H(C|T)}{H(C)} \tag{5.3}$$

It is more convenient for a measure of integratability to be larger when the two columns are *more* integratable, and so we will actually use the expression $\frac{H(C|T)}{H(C)}$ as the measure of integratability.

Computationally, $H(C|T)$ is easy to compute and interpret. Consider the probability $p(C_i|t)$ that denotes the fraction of mass of each column C_i present in the cluster t. We can define an entropy $H(C|t)$ as the entropy of this (two-atom) distribution. It is not hard to see that $H(C|T)$ is then the weighted average of this expression over all clusters t (weighted by the cluster weights $p(t)$).

Assume that both columns have the same size, and thus $p(C_1) = p(C_2)$. If the two columns were integratable, we would expect roughly half the mass of each cluster to belong to each column, and the entropy of this distribution would then be large. Averaging over all clusters, we get an estimate of how balanced the clusters are between the two columns, and that gives our measure of integratability.

We can compute this measure for the two instances depicted in Figure 5.1. In both examples, there are 15 crosses and 11 squares, and therefore $H(C) = 0.983$. For the left side, $H(C|T) = 0.98$, and therefore the integratability is 0.998. On the right side, $H(C|T) = 0.795$, and thus the integratability is 0.81, which is less than 0.998, as expected.

Our measure of integratability $\frac{H(C|T)}{H(C)}$ naturally extends to the multi-column scenario. Instead of having a two-atom distribution for $H(C|t)$, $H(C|t)$ is now of the same size as the set of columns, with each $p(C_h|t)$ representing the probability of a value x belongs to column C_h, given that x is of syntactic type t. Therefore, when columns are similar in syntactic types to one another, $H(C|t)$ is close to $H(C)$ for all t. Hence, $\frac{H(C|T)}{H(C)}$ is close to 1. Similarly, Equation 5.3 can be generalized to multi-column too. However, Theorem 5.2 is extensible only if JS divergence is generalized on a set of distributions $\{p_1, p_2, \ldots, p_n\}$ as follows:

$$D_{JS}(p_1, p_2, \ldots, p_n, \pi_1, \pi_2, \ldots, \pi_n) = H(\sum_{i=1}^{n} \pi_i p_i) - \sum_{i=1}^{n} \pi_i H(p_i)$$

where π_i is the weight for p_i with $\sum_i \pi_i = 1$.

5.3 Extracting a Match

A measure of integratability allows us to determine whether two columns of data have the same type. A more general kind of schema match occurs

when a single column (which we call the heterogeneous column) contains parts that have the same type with other columns. We came across one such example in the example from Section 5.1.2, when attempting to populate the attribute `propTitle`. In such cases, one would not expect a simple column-level comparison to detect type similarities; instead, what we need is a method for determining a *sub-string* of a column that matches types with another column.

As before, let C_1 and C_2 be the two columns under consideration. Let C_1 be the *heterogeneous* column i.e, the column from which we wish to extract a sub-string whose type matches that of C_2 (which we call the *corpus*). For each string $s \in C_1$, we therefore need to determine a sub-string $w(s)$ such that the set of strings $\{w(s)|s \in C_1\}$ integrates with C_2.

Suppose we have a procedure to compute a candidate $w(s)$. We can then test the efficacy of this procedure by integrating the resulting strings with C_2, using the method outlined in Section 5.2. A good procedure will generate a high integratability score.

One of generating a sub-string is by considering absolute positions in a string. For example, the sub-string extraction described by Warren and Tompa [WT06] calls for specific offsets into a string to generate the target string. We refer to such a sub-string as "syntactic", because the rule for constructing it is independent of the string contents.

It is possible (although inefficient) to search over all such syntactic rules to find the correct sub-string. However, in general such an approach will fail to capture many kinds of heterogeneous data, whose values may be derived semantically from different sources. This motivates our approach, which is based on using the *type* of the homogeneous column in order to extract relevant sub-strings.

5.3.1 Finding A Large Bump

We assume that a string in the column C_1 is of the form awb, where w is a string whose type matches that of C_2 and a, b are arbitrary strings. If this is the case, then we expect that a representation of w in the type space will match closely with a representation of strings in C_2, whereas the representations of a, b will not match.

We build a histogram on strings that characterizes the corpus C_2; in practice, this is done by fixing a parameter q and computing frequency counts for all q-grams that occur in any string of C_2.

Now for each string $s \in C_1$, we extract its q-grams and assign to each a weight that equals the relative frequency of this q-gram in the histogram constructed above. Let the relative range of these frequencies (the ratio of the largest to smallest) be denoted by r.

Treating these relative frequencies as likelihoods, we expect that if w indeed matches strings in C_2, then the likelihood of q-grams in w should be significantly higher than the likelihood of q-grams in the remainder of the string. In other words, as we move from left to right along s, we should see a jump in likelihood where w starts, and a corresponding dip where it ends.

The magnitude of a jump that constitutes a significant event is a subjective choice, depending on r. A good choice for the jump magnitude is $r/2$, for reasons we shall see later.

Consider a scenario where the corpus consists of different kinds of numeric sequences. A possible likelihood plot for the string `scro5127777` might look like Figure 5.2. Here, the jump (the shaded region) marks the beginning of the relevant sub-string.

The figure also illustrates how finding a significant transition is not sufficient. In the above example, there is no way to determine which of the

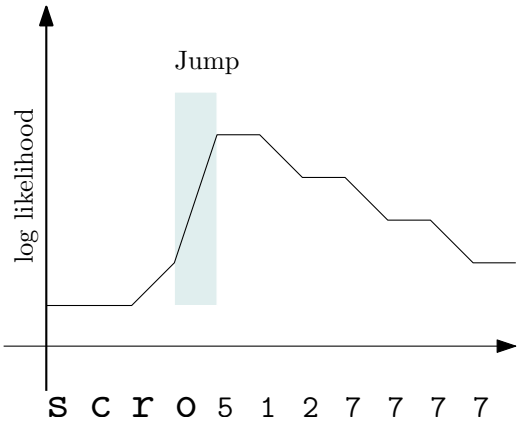


Figure 5.2: Using a jump in likelihood to indicate an interesting transition

strings `512`, `5127`, `51277` etc. are the appropriate strings to return, without making some kind of *ad hoc* choice.

Therefore, rather than defining a transition as a sufficiently large jump from one q-gram to the succeeding one in a string, we define a transition as a separation of likelihoods into two sets, *low* and *high*, such that the smallest likelihood in *high* is significantly larger than the highest likelihood in *low*.

If such a separation exists, then it is easy to identify the corresponding sub-string: we label all q-grams as being either *high* or *low*, and take the longest sub-string consisting of high q-grams.

Figure 5.3 depicts how this works, with the string `scro512arch`. Here, we can identify a clear gap of size $r/2$ that separates the q-grams into regions. The resulting sub-string that the algorithm returns is shaded in the figure.

The choice of $r/2$ as the gap width guarantees that we find at most one high sub-string. If we find no region, we deem the whole string to be high and return it as is.

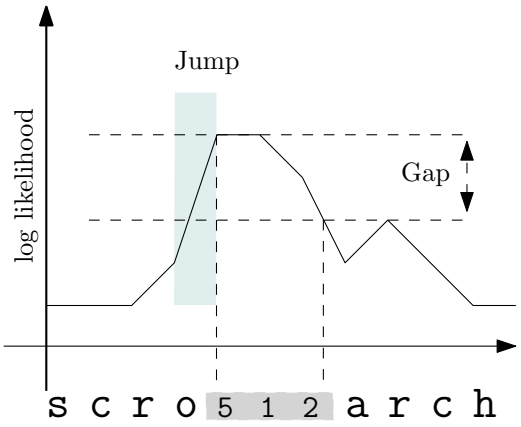


Figure 5.3: Using *low* and *high* regions to find an interesting sub-string

5.3.2 Combining Integratability with Sub-string Matches

In practise, we combine the measure of integratability and sub-string matches together, to predict the most likely relation between two columns. This procedure is illustrated as follows.

1. Compute the measure of integratability of two given columns. If the integratability is high, we say these two columns are integratable.

2. If the above integratability is low, we treat either column as a corpus to extract sub-strings from the other column. Compute the integratability between an original column and the sub-strings extracted from the other column. If one integratability is high, we say the column (where the sub-strings are extracted from) is sub-string integratable with the other column.

3. If the integratability computed in the above step is still low, we then compute the integratability of the two sets of sub-strings, and claim the two columns are dual-partially integratable if the integratability is high.

The latter two cases are both considered partially integratable, and it is useful when dealing with columns composed from a few semantic or syntactic types. This will be exemplified in Section 5.7.

5.4 (In)Validating Schema Matchings

Having developed tools for comparing and extracting types, we now apply them to different schema validation scenarios.

5.4.1 1-1 schema matching

The first scenario is a match between two columns, under the assumption that data in the first matches the type of data in the second, for example, the matching between `americaDB.coPiName` and `chinaDB.piName`. We compute integratability as described in Section 5.2. We have observed that integratability values close to 1.0 are reliable indicators of a valid match; in any case, comparing integratability values is a reliable way of deciding which matches are more accurate than others.

5.4.2 1-N or N-1 matchings

A schema mapping differs from a match in that the specific function relating elements of one schema to another is described, in addition to the element associations as well. A simple example of a multi-column schema mapping is the LOGIN-ID example of Warren and Tompa [WT06], where the LOGIN-ID field concatenates the first character of the `first-name` field, the first character of the `middle-name` field, and the entire `last-name` field. Such a map is called a *1-N mapping*, since one element (LOGIN-ID) is mapped to multiple elements. Another example is from our schema integration problem in

Section 5.1.2; here, an example of an M-1 matching is the matching between `concat (americaDB.piLN, americaDB.piFN)` and `chinaDB.coPiName`.

If the schema mapping is provided, then validating such a mapping can be done in the same way as we validated the 1-1 column match. The map is used to construct mapped entries from the destination schema, and then the original schema data and this new set of entries is checked for integratability.

5.4.3 Compositional matchings

In the introduction, we distinguished between ordered and unordered compositional matchings. As we argued there, unordered compositional matchings can be validated by using a simple matching strategy; the use of distributions over q-grams allows us to use the above method for validation.

Here we consider ordered compositional matchings. Note that even if the compositional matching consists of a conjunction of 1-1 matchings, we cannot use 1-1 matching validation to validate it. This comes from observing that a distribution (i.e type) on a composition cannot always be reduced to product of distributions (i.e composition of types) on the individual matchings.

Therefore, we use an extended representation of the data that encodes the positional information. The idea is as follows. Rather than treating data from each element separately, we construct one large tuple that concatenates all M schema elements in the appropriate order. When constructing q-grams for this tuple, we extend the representation with positional information. For example, if a tuple is formed by concatenating the strings `abcd` and `tybc`, then rather than extracting one 2-gram `bc` with a count of two, we extract *two* 2-grams of the form (`bc`, 1) and (`bc`, 2). We can now do an integratability calculation using this q-gram representation.

5.4.4 Sub-string Matches

Another kind of matching that has no analog in the value-based world is a sub-string matching. In this scenario, a schema matcher proposes a match between two columns of a table, but the actual matching types are parts of the values. Again, using the multinational research proposal example, we need to use sub-string matching to populate attributes like `propTitle` in the integrated database `chinamericaDB` from `americaDB.propTitle` and `sub-string(chinaDB.proposalInfo)`.

A simple example of this phenomenon occurs when matching a table that uses passport ID strings to identify people with another that uses birth date information (in combination with other fields). Many passport IDs have birth date information encoded within the string itself, and in order to construct a matching, we would need to extract relevant sub-strings from one or both columns and compare their types.

In general, the match might occur between sub-strings of both columns. Here, we consider the simpler case in which a sub-string of one column must match another column in its entirety. We use the procedure of Section 5.3 to extract sub-strings from the column data, and then perform an integratability check between the set of extracted sub-strings and the other column.

5.5 An Invalidation Certificate

The type-based validation methods described above all end with an integratability calculation, returning a value that ranges between 0 and 1 and is larger when the two schema elements being matched are believed to be more similar in type.

Our experiments have shown that for elements that are integratable, the

score returned is typically very close to 1, and as the score decreases, the integratability (as perceived by the user) decreases rapidly. Since the scale of the measurement is based on entropy calculations, this "log-scale" behavior is perhaps not surprising.

However, a deeper understanding of the score returned requires us to look beyond the number itself. The integratability measure is computed from a soft clustering of the data; in the process of doing this research, we discovered that the soft clustering contains information that enhances our understanding of the measure.

A hard clustering is easy to represent by listing the partitions. A soft clustering is harder to visualize because points spread their mass across multiple clusters. Moreover, as pointed out in Chapter 4, even the notion of a cluster is not well defined. Two distinct clusters in a solution are not really distinct if the membership probabilities $p(t|x)$ are identical for both of them.

We will visualize integratability using a "bar-code" idea borrowed from Chapter 4. Consider an image with one row for each item in either of the two columns, grouped so that rows for items in C_1 appear above rows for items in C_2. Each column of this image corresponds to a cluster t, and has a variable width proportional to $p(t)$ (variable width is implemented by using multiple pixels). The content of a column is the value $p(t|x)$, represented in log-scale with darker shades signifying higher probabilities.

Since a soft clustering might spread mass between clusters that are essentially identical, we need to group the columns of such clusters together, so as to make the representation more meaningful. Two clusters are similar if their cluster membership vectors $p(t|x)$ are similar. Using this observation, we can group clusters together, and reorder the columns so that all groups appear together.

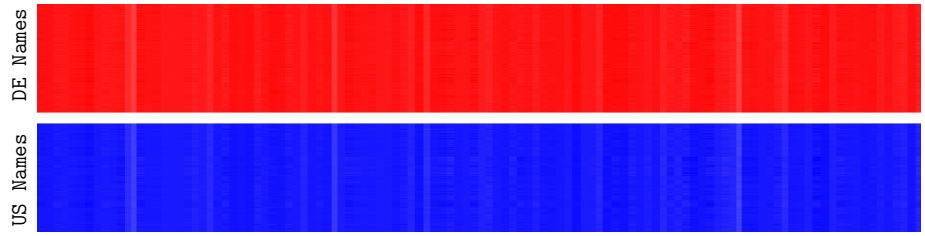
An Example

We illustrate the idea of a bar code with three simple examples, drawn from the `name` dataset (for more details on the dataset, see Section 5.7). One of the integratability tests described in Section 5.1 was a test to see whether two columns of names corresponded to the same nationality. Suppose we were to compare the lists of American and German names. Figure 5.4(a) illustrates what the bar-code looks like. As described above, rows are indexed by the data, and columns are indexed by the clusters. Each entry of the bar-code represents a particular cluster membership probability $p(t|x)$, with a darker color representing a higher value (in log-scale). All entries of the first data source appear above those of the second.

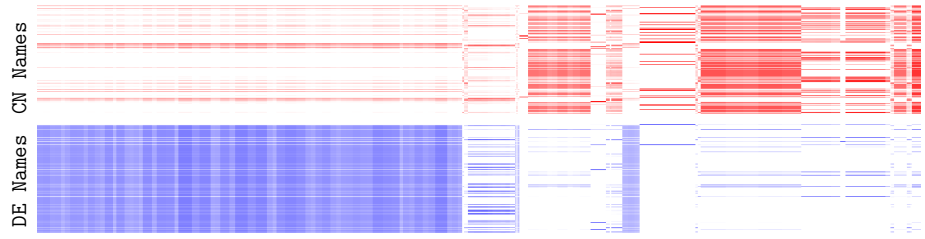
The integratability of these two sources is very high; notice that in the bar code, the two horizontal strips look almost identical. This means that each cluster contains the same mass from both data sets, implying integratability. Note that given the Germanic roots of many American names, this result is not particularly surprising.

At the other extreme, consider doing the same with Chinese and German names. The integratability score is 0.24, which is much lower, and predicts that the data sets should be separated into separate clusters. As Figure 5.4(b) shows, this is exactly what happens. The top and bottom strips are almost complementary to each other, indicating that most clusters contain mostly one kind or the other, and rarely both. The shades are also deep, indicating not only a difference, but a strong difference.

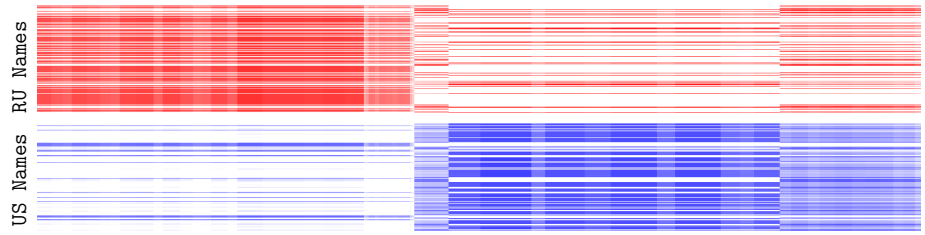
An intermediate example can be seen by integrating American and Russian names, which bear some, but not significant resemblance. The integratability score here is 0.61, and as Figure 5.4(c) shows, the top and bottom strips are not as sharply different as in the Chinese-German case, and neither are



(a) Integratability of German and American names = 0.999



(b) Integratability of Chinese and German names = 0.399



(c) Integratability of Russian and American names = 0.738

Figure 5.4: Certificates for 3 Pairs of `name` data

they as identical as in the American-German case. Remember that the intensity of the shading indicates the probability mass, and in this case the shading is weaker, indicating that even when there appear to be differences, it is not as significant.

5.6 Methodology

The algorithm to compute the measure of integratability is rather similar to the one estimating the heterogeneity. The differences are the sub-string

Algorithm 2 Computing Integratability.

1: Convert input strings to distribution of q-grams, and calculate $H(C)$
2: Extract sub-string with respect to each other as corpus if necessary
3: Compute the weight for each q-gram
4: Compute the weighted probabilistic vector $p(Y|x)$ in the space of q-grams for each value x
5: Compute $\beta^* = \frac{H(X)}{I(X;Y)}$
6: Compute soft clustering using `iIB` with β set to β^*
7: Compute $H(C|T)$ by $p(t|x)$
8: Compute the integratability by computing $\frac{H(C|T)}{H(C)}$

extraction, weighting the q-grams and computing $H(C|T)$. The first and the last differences are explained in great detail throughout earlier sections. We now explain the weighting scheme of the q-grams.

5.6.1 Weighting the q-grams

Again, in our setting, weighting schemes like i.d.f are not appropriate, as our goal is to determine whether two columns of data can be integrated. Therefore, q-grams that distinguish one set of data from another are more important than q-grams that are either rare or frequent.

In Chapter 4, we maximize the weights of q-grams whose probabilities appearing in the set of values are around 0.5, since those q-grams have the "highest" distinguishing power. However, in the setting of measuring integratability, such q-grams may not be powerful. Consider integrating columns of data with the same number of items. A q-gram with no distinguishing power will appear in roughly the same number of items in each column. A q-gram with high distinguishing power will occur mostly in one column or

the other. For example, a q-gram which appears in only one column with 0.1 probability is considered more powerful than another q-grams appearing uniformly with probability 0.5 in all columns.

There are different ways of constructing a measure that captures this idea. Our approach is as follows. For a q-gram q, we compute the number of times it appears in each column, and call these frequencies $f_1, f_2, \ldots, f_m$. Normalizing each f_h by n_h, the number of values in each column, and taking the largest difference among all $\frac{f_h}{n_h}$, i.e., $p' = \max \frac{f_h}{n_h} - \min \frac{f_h}{n_h}$ or $p' = \min \frac{f_h}{n_h} - \max \frac{f_h}{n_h}$, we get the quantity p' that ranges between -1 and 1. It will be convenient to work with the range $[0, 1]$, so we scale and shift p', obtaining $p = \frac{1+p'}{2}$.

This new variable has a range of $[0, 1]$, and is 0 or 1 when the q-gram appears exclusively in one or the other column. It takes the value 0.5 when the q-gram appears equally often in both. We desire a function that is symmetric around 0.5 and is maximized at the boundaries. Any appropriately chosen convex function suffices; we will use shifted negative entropy $W(p) = 1 - H(p)$, where $H(p) = -p \log p - (1 - p) \log(1 - p)$, where log is of base 2.

Once the frequency counts are weighted using the above method, we normalize the histograms, yielding a distribution for each string.

5.7 Experiments

5.7.1 Datasets

Besides the `many eyes` dataset, we will use the following dataset here to demonstrate the idea of using integratability better.

UVa Name Dataset

This dataset, called `name` dataset, is downloaded from the ACM Programming contest on-line judge, provided by Universidad de Valladolid[1]. There is a list of authors who submitted their solutions to the on-line judge[2]. We downloaded this list without accessing their profiles. From this list, we extracted their names, as well as their nationalities from the national flag attached to each user. Names are grouped by the nationalities, forming one column for each nationality.

5.7.2 Validation of Integratability Measure

In this section, we first apply our integratability measure on columns related to "car models" from the `many eyes` dataset.

Suppose someone thinks these 6 columns are related, `car@vehicles`, `car_line@fuel-economy-stats`, `car_name@cars-17`, `g_model@googlebase_vehicle`, `model@autosforsale-2` and `vehicle_name@car-data`. As she is not sure if these columns are integratable, she then use our proposed method to compute the mutual integratability among the 6 columns. Results are shown in Table 5.2.

First it is evident that the measure satisfies reflexivity; every column is integratable with itself. The columns are quite similar with each other since someone already thinks they are related by looking at their names, so this is reflected in the high integratability scores for pairs. By observing that column `vehicle_name@car-data` has relatively low integratabilities compared to other columns: except for `car_name@cars-17`, all other integratabilities are below 0.7. This is a strong signal for column `vehicle_name@car-data` to

[1]http://acm.uva.es

[2]http://acm.uva.es/problemset/rankauthjudge.php

Column A	Column B	Integratability
car@vehicles	car	1.000000
	car_line	0.819051
	car_name	0.862825
	g_model	0.856623
	model	0.804910
	vehicle_name	0.647728
car_line@fuel-economy-stats	car_line	1.000000
	car_name	0.781879
	g_model	0.765946
	model	0.789608
	vehicle_name	0.661030
car_name@cars-17	car_name	1.000000
	g_model	0.729814
	model	0.792743
	vehicle_name	0.792847
g_model@googlebase_vehicle	g_model	1.000000
	model	0.738048
	vehicle_name	0.576021
model@autosforsale-2	model	1.00000
	vehicle_name	0.649472
vehicle_name@car-data	vehicle_name	1.000000

Table 5.2: Integratability of Columns on car models. The table names for column B are omitted due to the space constraint.

Names from Different Region	Integratability with CN Names	Integratability with US Names
CN Names	1.000000	0.545518
HK Names	0.816060	0.732209
SG Names	0.870063	0.840161
JP Names	0.350561	0.580060
US Names	0.545518	1.000000
DE Names	0.399300	0.999930
ES Names	0.384223	0.887577
RU Names	0.267962	0.737690

Table 5.3: The Integratability of the Names of People from Different Regions

have different syntactic types than others. Further checking on this column discovered that this column describes a lot more than just car models, e.g., one of its values "`Audi A4 3.0 Quattro 4dr auto`" includes car manufacturer, model, engine size, car class and transmission. Therefore, it is less integratable with other columns which only record car models.

In the `name` dataset, we test the integratabilities between names of people from different regions in the world. For example, mainland Chinese usually use the romanization of their Chinese names as their names in the Latin system. Chinese people from Hong Kong or Singapore will probably use the anglicization of their Chinese names, with another English name as a prefix or suffix, like Simon Cheung. Therefore, we do expect that Hong Kong or Singapore names are more similar to Western names in syntactic types.

From Table 5.3, we discovered that both Hong Kong names and Singapore names are indeed more likely to integrate with either Chinese names

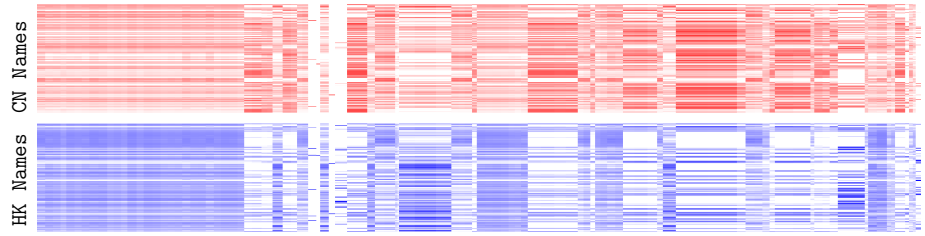
or American names than the other. At the same time, American names are more likely to integrate with names from Hong Kong or Singapore rather than names from mainland China. This matches with our intuition that Hong Kong and Singapore names are a combination of Chinese names and English names, and it is noteworthy that a purely information-theoretic approach is able to capture this. The other interesting remark is that, as Germanic languages are most similar to themselves, but less similar to Latin languages, and least similar to Slavic languages, the integratabilities between names from such languages show the consistent ranking, i.e., American names are most similar to German names, then followed by Spanish names, next followed by Russian names.

Investigating further the properties of this measure, the example shown in Figure 5.5 is instructive. In this example, we showed the certificates obtained for the integratability of pairs of name sets between Chinese, Hong Kong and American names. Figure 5.5(c) shows the overlap between Hong Kong and American names is more significant than that between Chinese and American names (Figure 5.5(b)), but not as much as that between Chinese and Hong Kong names (Figure 5.5(a)), which matches the integratability measure we listed in Table 5.3. Note that transitivity does not hold; although names from China and America both integrate well with names from Hong Kong, they do not integrate well with each other.

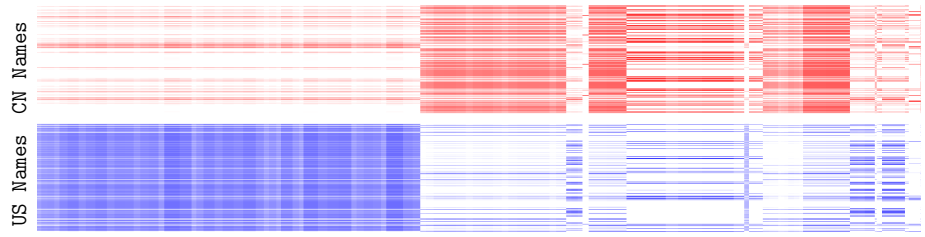
5.7.3 Validation of Sub-string Extraction

In this section, we will first validate our method of sub-string extraction by getting sub-strings from a potentially *heterogeneous* column against one homogeneous column (the *corpus*).

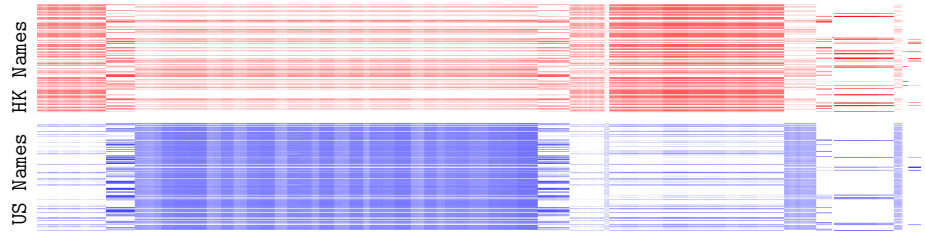
From Table 5.2, we noticed that column `vehicle_name@car-data` does



(a) Mainland Chinese Names v.s. Hong Kong Names



(b) Mainland Chinese Name v.s. American Names



(c) Hong Kong Names v.s. American Names

Figure 5.5: Certificates for 3 Pairs of `name` data

not integrate well with others because it is of a composition of several fields, which are treated differently in other tables. Therefore, we need to perform the sub-string extraction procedure to generate another set of string values, which are of better integratability.

As previously we discovered, column `vehicle_name@car-data.tsv` is a composite column, including several fields that are recorded separately in most other files. Thus we use one column which contains only an atomic field to extract the similar sub-strings by the method proposed in Section 5.3. There are also false positives like “ord” extracted from “honda accord ex

Strings in `vehicle_name@car-data.tsv`	Sub-string Extracted with Corpus `g_make@googlebase_vehicle`
⋮	⋮
cadillac xlr convertible 2dr	cadillac
chevrolet aveo ls 4dr hatch	chevrolet
acura tsx 4dr	acura
chrysler concorde lxi 4dr	chrysler
infiniti g35 sport coupe 2dr	infiniti
jaguar x-type 2.5 4dr	jaguar
nissan murano sl	nissan
toyota avalon xls 4dr	toyota
volvo s80 t6 4dr	volvo
⋮	⋮

Table 5.4: Exact sub-strings which are similar to `g_make@googlebase_vehicle` from composite column `vehicle_name@car-data.tsv`

2dr". This is probably due to the high frequency of the word "ford". Higher precision methods are more complicated and have to be tuned according to the specific set of values. Our method proposed in Section 5.3 provides a generic way to extract sub-strings with simple heuristics.

In the second experiment, we attempt to extract Chinese names from strings concatenated from Chinese names and American names. The result is shown in Table 5.5, and the right column contains sub-strings extracted from the left column. The text in *italics* in the left column of Table 5.5 is the original Chinese names which we used to construct the concatenated string.

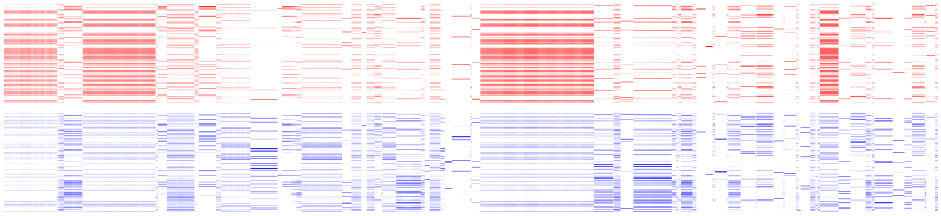
Concatenated String	Sub-string Extracted
⋮	⋮
*chen yang*amit singh	chen yang
*chen li*andrew conner	chen li<u>and</u>
*lai wen*david musicant	lai wen<u>da</u>
*li cheng*david strawn	li cheng
*li zhenting*denny vandenberg	li zhenting
*liu song*evan davis	liu song<u>e</u>
*ma xiao*ilya kozavchinsky	ma xiao
*niu chuanxin*jack hudson	niu chuan
*ren xiao yin*jason winokur	ren xiao yin<u>j</u>
*wang ting*jiwon kim	wang ting
⋮	⋮

Table 5.5: Chinese Names Extraction from a Heterogeneous Column of Chinese Names and American Names

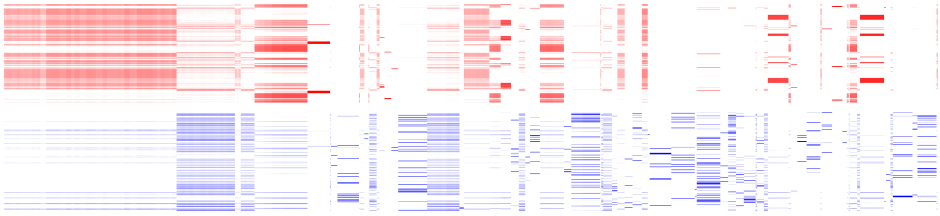
We underline the incorrect portions of the extracted strings.

5.7.4 Validation of Compositional Matchings

In this section, we will show how the positional encoding described in Section 5.4 allows us to distinguish between correctly and incorrectly ordered compositions. Given a matching that indicates source field "A" in table 1 maps to target field "C" in table 3, source field "D" in tables 2 maps to target field "B" in table 1, and "F" in table 3 maps to "G" in table 2. We construct two columns from the mapping: the first column contains fields "A", "D" and "F", separated by "|" (suppose "|" does not appear in any of the values).



(a) `make|model` vs `g_make|g_model`: 0.749511



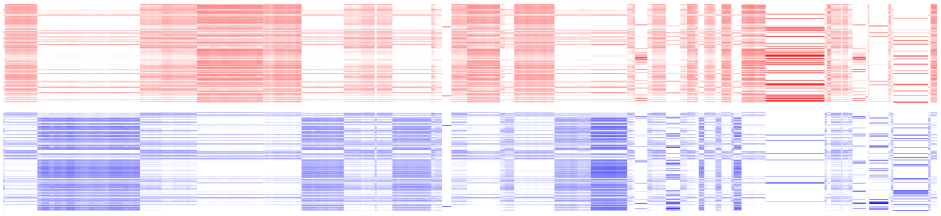
(b) `make|model` vs `g_model|g_make`: 0.309306

Figure 5.6: Integratability helps identifying potential incorrect mappings between two tables.

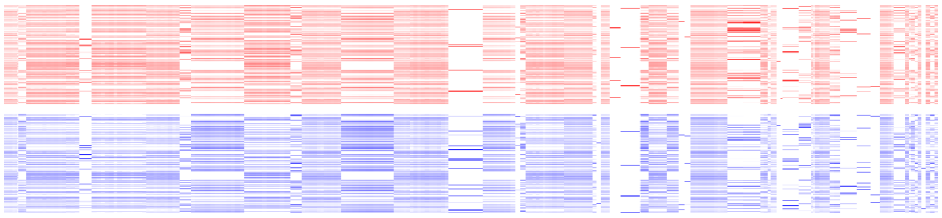
For example, if string "a", "d" and "f" are the instances of fields "A", "D" and "F" respectively, the first column will have a row "a|d|f". Similarly, we construct the second column from fields "C", "B" and "G", separated by "|".

If this mapping is valid, the two columns will have the same syntactic type. We illustrate this by constructing a correct and incorrect matching from two pairs of columns and allowing the positional encoding approach to distinguish the two.

We first select columns `make`, `model` from table `mpg.tsv`, and columns `g_make`, `g_model` from table `googlebase_vehicle.tsv`. Figure 5.6(a) shows the integratability when `make` is matched with `g_make` and `model` is matched with `g_model`, where the certificate shows the strings constructed with such mapping are similar. However, in Figure 5.6(b), when `make` is matched with `g_model` while `model` is matched with `g_make`, both low integratability and the certificate say this mapping is not valid.



(a) CN|US Names vs US|CN Names: 0.682351



(b) CN|US Names Set A vs CN|US Names Set B: 0.948135

Figure 5.7: Positional information affects integratability.

Figure 5.7 shows that when two columns contain values from the same set of single-type columns but with different order, positional encodings allow us to distinguish between them using our measure of integratability. The certificate shows that a mapping which accidentally made a wrong match between Chinese names and American names can be discovered, since the integratability between the two columns constructed by the wrong mapping is low.

5.7.5 Integratability between two Heterogeneous Columns

An interesting question that we did not fully address was how extraction of sub-strings will work if both columns under consideration were heterogeneous themselves. For example, we could concatenate Chinese names and Japanese names to obtain a new column which consists of strings concatenated from Chinese names followed by Japanese names. Such columns obtained by such

Column A	Column B	Integratability
CN Name	US Name	0.545518
CN Name-JP Name	US Name-JP Name	0.932994
CN Name-RU Name	US Name-RU Name	0.939777

Table 5.6: Integratability of sub-strings between two horizontally heterogeneous columns

concatenation are horizontally heterogeneous.

One approach to deal with this case is to run two instances of sub-string extraction. We use the first column as a corpus to extract the sub-strings from the second column. This will give us a set of sub-strings which are most similar to the first column. We then do the same thing again, but using the second column as corpus to extract sub-strings from the first column. Lastly, we put the two sets of sub-strings together and compute their integrability.

Table 5.6 shows the integratability between two pairs of columns: `CN Name-JP Name` against `US Name-JP Name`, and `CN Name-RU Name` against `US Name-RU Name`. The first row, the integratability between CN Name and US Name, provides a reference point for the latter rows. We see that in both cases, sub-string extraction from both columns is moderately successful at extracting the relevant common portions from both strings; the integratability scores are much higher than the scores for the base (non-integrable) pair of Chinese and American Names.

5.7.6 Performance

The performance is largely dependent on the clustering algorithm developed in Chapter 4. However, as the column heterogeneity measure for one column is not a concern here, the background addition is no longer necessary. The

computation of integratability is actually faster than the computation of column heterogeneity given the same amount of data values and the same representation of those values.

Sub-string extraction is faster than the computation of integratability as no iterative computation is involved, and the computation of the probabilities of the 2-grams is just a single scan over all data values.

5.8 Summary

In this chapter, we discussed mainly on validation of schema matchings through the types of values across multiple columns. We first proposed a measure of integratability based on conditional entropy of the columns given the type information. Intuitively, the higher the measure is, the more integratable the columns are. However, low integratability is a signal to suggest the columns are not to be integrated, else the integrated column is likely to be more heterogeneous than the original columns. Our measure of integratability can be plugged into any data integration tool, to provide a validation after an integration plan is drafted. Database administrator may look at the integratabilities and re-examine some of the column matchings or mappings.

Together with the measure of integratability, we have also proposed the procedure to extract sub-strings. This made the integratability avaible on the sub-string level, which is more applicable when handling horizontally heterogenous columns.

In Chapter 4 and Chapter 5, we prefer homogeneous columns than heterogeneous ones, and we prevent databases from becoming more heterogeneous. However, we can also make column heterogeneity useful, which will be demonstrated in Chapter 6.

Chapter 6

Inter-column Heterogeneity (II): Capturing the Semantics

In Chapter 1, we have motivated the need of having a user-friendly platform for easy access, manipulation and contribution to databases. However, this will be costly if such a platform operates directly on the databases as most users do not have the basic database knowledge. It is therefore desired to have a separation between such platform and databases, which is able to interpret users' intentions, and chooses only the "right" actions to operate the databases. Semantics then come into the picture since users' simple instructions have to be understood from the semantics perspective, as discussed in Example 1.5.

In Chapters 4 and 5, we elaborated the syntactic types of values within a column and across multiple columns. However, the syntactic types are not the same as the semantic types. Section 4.1.1 illustrated the relationship between syntactics and semantics, i.e., semantics are represented, delivered through syntactics. It also pointed out the key differences between syntactic types and semantics types:

- Syntactics are objective, while semantics are subjective; and
- There is no direct mapping from syntactics to semantics

The first difference tells us it is more challenging to capture the semantics than to study the syntactics, as we need to learn how people interpret. The second difference says the mappings from syntactics to semantics are usually done manually. For example, when filling up the credit card information at a payment web-page, everyone is required to fill in the expiry date in a certain format, like yyyy–mm or mm/yy, otherwise the system will reject. By doing this, the syntactics for the semantics "`date`" is made explicit. So is the mapping from the syntactics back to the semantics. Explicitly defining the syntactics for each of the semantics is tedious and very unlikely, as every word in a dictionary has its own semantics. Such manual work can be only done for some important semantics, e.g, the semantics in a semantic hierarchy in a certain domain. As users are free to choose any semantics, it is necessary to have a mechanism that interprets users semantics.

In this chapter, we will propose a solution that interprets users' semantics, along with a procedure that makes users' data available to databases and other users. We next look at what are the desiderata for such a mechanism in order to capture semantics.

6.1 Semantics: Desiderata

As our general solution is a separation layer that separates users' actions from database operations, community users are then isolated from accessing the database directly. But this requires the separation layer to play two roles. First it plays the role of interpreter, which understands a user in her own language. For example, one may issue a command like `insert (make:`

`toyota, model: prius, price: 20000, mileage: 10000)`. This separation layer then interprets this command, and understands that the user wants to insert a quadruple `(toyota, prius, 20000, 10000)`, which is labeled by `(make, model, price, mileage)` respectively. On the other hand, with this quadruple, the separation layer should act like a database expert to insert it properly into the database. In conventional databases, we expect the insert operation to be always specified with a table name and column names. However, given that users have no knowledge about the database, the separation layer needs to figure out which table and what are the columns to be inserted. Similarly, for users who issue database queries, this separation layer ought to interpret the queries, perform the query operations in the database like an expert, and then present the results back to the user.

In order to establish such an intelligent separation layer, we have to overcome at least three major challenges. The first challenge is on the interpretation of users' actions.

6.1.1 Interpreting Users

Example 6.1. *Alice inserted 100 records with the make of cars named as* ***`make`****, while Bob inserted another 100 car records with the same attribute named as* ***`manufacturer`****. If our separation layer blindly inserted the data into the column whose name is either* ***`make`*** *or* ***`manufacturer`****, we would have encountered the following problem. When Carol retrieves these data, she can only get half of the records no matter whether she uses* ***`make`*** *or* ***`manufacturer`****. This is because, by blind insertion, there is no connection between* ***`make`*** *and* ***`manufacturer`****, but human beings consider them highly related as they carry the same semantics and expect getting all results by querying just one of them.*

Hence our separation layer must be intelligent enough to associate attribute names with similar semantics. In fact, users are not aware of what attributes are used in the database. What they want is just to indicate the semantics of the values. Inspired by the method of tagging multimedia objects to facilitate searching, like in Flickr[1] and YouTube[2], we allow users to describe the semantics of data values by *tags*. Tags merely represent the semantics of values, not to match elements in the schemas like what attribute names do. Note that the problem in Example 6.1 exists because one data value is either associated to column `make`, or column `manufacturer`, but not both. When the semantics of values are represented by tags, there is no such constraint, i.e., one value can be associated with multiple tags. In Example 6.1, if a value is inserted with tag `make`, the separation layer should then associate another tag `manufacturer` to this value since the separation layer is intelligent enough to know the semantics of `make` and `manufacturer` are similar.

In the following, a tag is called *the original tag* of a value if the tag is used when this value is contributed. So our problem now is how to associate the original tag of one value to other tags that other users may use. One solution is to set up a mapping from one tag to a set of tags with similar semantics. The following example explains why this does not work.

Example 6.2. *(i) retrieve the contact of an agent with email js@a.com; (ii) retrieve the contact of an agent with phone 567-1234. The two* `contact` *refer to two different things. In the first query, it requires a result that is not an email address, while for the second query, the required result is not a phone number. The semantics of* `contact` *is similar to both* `email` *and* `phone`*,*

[1]http://www.flickr.com

[2]http://www.youtube.com

therefore, simply tying tags with similar semantics is not feasible since the semantics do depend on the query context.

Therefore, establishing mappings from one tag to another is not an appropriate way of associating tags, as the semantics of the tags are content-related. We need a better way to associate one tag with another.

6.1.2 Data Storage

Here comes our second problem–how do we store the data? Since users do not know anything about databases, the separation layer must be able to store users' data properly. As relations can be arbitrary, we allow data to be stored without specifying a schema. Structures like Google's BigTable [CDG+06, CDG+08] and wide-table [CBN07, LHLG09] suit our need. Such structures contain one single table with thousands of columns. Each row can be a relation of any type, which can relate to only a few or a few dozens of columns. Thus such tables are extremely sparse, with many empty cells. In this chapter, we implement our ideas on wide-table, as it makes our idea easier to understand. In fact, our idea is a generic approach as long as database operations are carefully designed in accordance to the underlying data model. Our data model will be elaborated more in Section 6.2.

6.1.3 Uncertainty in Querying

Unlike querying data from a well structured database where it is deterministic if a tuple matches the query, there are three types of uncertainties as follows:

- Uncertainty in interpreting a query specified by a user, which is exemplified by Example 6.2.

- Uncertainty in matching tuples to a query.
- Uncertainty in matching different values from one tuple to a query.

We first illustrate the uncertainty in matching different tuples to a given query by Example 6.3, which is presented below.

Example 6.3. *(i) Retrieve the* `manufacturer` *of model* `X5`*; (ii) Retrieve the* `brand` *of model* `X5`*. Coincidentally, there is a car model* `X5` *by* `BMW`*, and there is also a cell phone model* `X5` *by* `Sony Ericsson`*. When retrieving the result, both* `BMW` *and* `Sony Ericsson` *are candidates. There is however some differences between asking about* `manufacturer` *and* `brand`*. Most of the time, the make of a car is referred as* `manufacturer` *more than* `brand`*, but the make of a cell phone is referred as* `brand` *more than* `manufacturer`*. Therefore the answer for (i) is more likely to be* `BMW` *while the answer for (ii) is more likely to be* `Sony Ericsson`*. When matching tuples to the query, as there is no definite answer, tuples are matched with likelihoods. The result should be ranked according to such likelihoods so that most likely results are to be presented first.*

Even for one tuple, there are multiple values that could be the answer to the query. Example 6.4 then explains the last type of uncertainty: the uncertainty in matching different values within the same tuple.

Example 6.4. *Retrieve the* `contact` *of an agent with phone 234-7890. For the tuple which matches with phone number 234-7890, there are two values which can be mapped to contact,* `abc@def.com` *and* `123, Main St`*. When this happens, there should be a likelihood that associates* `abc@def.com` *to* `contact`*, and another likelihood that associates* `123, Main St` *to* `contact`*. Therefore, besides calculating likelihoods for tuples, likelihoods should also associate values to tags.*

Considering the three challenges introduced in the above three sections, we propose a novel data model, called *probabilistic tagging*, which tackles the first and the third challenges, whereas the second challenge is taken care by wide-table, as we shall see in the next section.

6.2 Probabilistic Tagging: an Overview

In this section, we discuss how probabilistic tagging can be used as a separation layer as illustrated in Section 6.1.

We first examine the command "`insert (make: toyota, model: prius, price: 20000, mileage: 10000)`". The user, who issues the above command, may not know whether columns `make`, `model`, `price` and `mileage` exist in the database. These terms can be considered as tags for the separation layer to interpret the semantics of the values. Relational databases use column names to describe the semantics of the values in each column. A database expert will not use "a" as a column name, but something interpretable like "account". In Table 6.1, the semantics of a value in the wide-table are consistent with the name of the column it is in. From this perspective, column names serve as tags in relational model. By the homogeneity property of relational model, the relationship between data values and data columns is definite, and each value is associated to one and only one column name.

For our probabilistic tagging, we allow the flexibility of associating one value to multiple tags, as tags are meaningful on the semantics level, rather than structural level. Therefore, users may tag `toyota` as `make`, `manufacturer`, or even {`make,manufacturer`}. By doing this, if tuples in Example 6.1 are tagged as {`make,manufacturer`}, no matter whether Carol retrieves data

make	model	price	mileage	year	name	lastname	firstname	contact	email	phone	handphone
BMW	3	31900	190500			Smith	John		js@a.com	222-4567	222-5678
Ford	F150	15900	198405	2002	John			j@b.com			234-8765
Toyota	Camry		191278	2004		Yamada	Taro			345-6789	
Honda	Civic	16000		2004		Brown	John	9,1st St	jb@c.com	567-1234	
Ford	Focus		189038		Tom	Davidson			td@d.com		678-2345
Volvo	XC90							123-4567			

Table 6.1: An example of wide table on car database, including agents' details

through `make` or `manufacturer`, she will get all 100 records. We leave it to Section 6.2.1 to discuss how to build such associations.

Another advantage of using multiple tags is to solve ambiguity. In Example 6.2, `contact` is a more general concept than `email` or `phone`. Both types can appear in a column called `contact` in a relational model, which makes emails and phone numbers indistinguishable on their semantics. However, if we associate {`email,contact`} to emails, and {`phone,contact`} to phone numbers, their semantics are easy to tell apart.

Table 6.2 gives a logic view on tagging tuples in Table 6.1, where values from the same column in Table 6.1 are tagged differently, and some values are associated by multiple tags which are semantically close.

However, nothing comes free. Once we relax such an association to multiple tags, we have to consider what is the appropriate set of tags that best describes the semantics of a value. By saying appropriate, it means tags are closely related to their values and no other irrelevant tags are expected to be associated. For example, value `toyota` can be associated with both tags `make` and `manufacturer`, but not with `model`.

Even if we can associate values with an appropriate sets of tags, one may ask if the tags being used are equally likely to describe the semantics of a data value, or if there is an order among the tags. This is what we encountered in Example 6.3, `BMW` is associated to `manufacturer` better than `brand` whereas `Sony Ericsson` is associated to `brand` better than `manufacturer`. To deal with this kind of associativity, a measure is proposed to indicate how close a tag is from a given data value. This measure should be higher when a tag is closer to the given value. Thus, the direct interpretation of this measure is the likelihood of a tag being used to tag a given value.

Definition 6.1 (Probabilistic Tagging)**.** *Let $\mathcal{G}$ be the set of tags, for a value*

TID	Data
026	BMW:{make,brand} | 3:{model} | Smith:{lastname} | John:{firstname} | 222-4567:{phone} | 222-5678:{handphone}
289	Ford:{make} | F150:{model,name} | John:{name,firstname} | j@b.com:{contact,email} | 234-8765:{handphone,contact}
354	Toyota:{make,manufacturer} | Camry:{name} | Yamada:{lastname} | Taro:{firstname} | 345-6789:{phone,contact}
571	Honda:{brand} | Civic:{model,name} | Brown:{lastname} | John:{firstname,name} | jb@c.com:{email,contact}
702	Ford:{manufacturer} | Focus:{model} | Tom:{name,firstname} | Davidson:{lastname,name} | td@d.com:{email,contact}
886	Volvo:{make} | XC90:{model} | 123-4567:{contact,phone,handphone}

Table 6.2: Values tagged with multiple tags

v and a tag $g \in \mathcal{G}$, probabilistic tagging *refers to the act of associating a probability $\zeta_{v,g}$ for v and g, such that the value v is tagged by the tag g with probability $\zeta_{v,g}$.*

The probability $\zeta_{v,g}$ is called associated probability*.*

For every tag in Table 6.2, there is a probability associated to each tag, representing the likelihood of this tag being used to tag the value. This is presented in Table 6.3. Each value is identified with tuple-ID (TID) and location-ID (LID) to indicate which tuple it comes from and its location in the tuple. The type indicates the type of the value, if it is string (S) or numerical (N). The last column is a set of tag-probability pairs. For example, `John` in the tuple with TID 026 is tagged by `firstname` with associated probability 1.0, and `name` with associated probability 0.7. We do not have the constraint that the associated probability sums up to be 1, as there is no mutual exclusion among the tags, and a value is absolutely possible to be tagged by more than one tag. There are three tuples containing value `John` in Table 6.3, it is because tuples come from different data sources with possibly different tags, and associated probabilities with these tags may also be different.

So far we have described a big picture about probabilistic tagging. The challenges, introduced in Section 6.1, have been transferred to the major challenge of associating values and tags with probabilities. We now list a few possible ways to establish such an association.

6.2.1 Association between Tags and Values

Since our model is tag-centric, the first issue that needs to be addressed is how tags can be associated correctly with the values in each tuple. Based on our analysis, there are at least four ways to achieve this:

Value	TID	LID	Type	Probabilistic Tagging
BMW	026	1	S	⟨make, 1.0⟩, ⟨brand, 0.4⟩
3	026	2	S	⟨model, 1.0⟩, ⟨name, 0.3⟩
31900	026	3	N	⟨price, 1.0⟩
Smith	026	5	S	⟨lastname, 1.0⟩
John	026	6	S	⟨firstname, 1.0⟩, ⟨name, 0.7⟩
Ford	289	1	S	⟨make, 1.0⟩, ⟨manufacturer, 0.8⟩
F150	289	2	S	⟨model, 1.0⟩, ⟨name, 0.4⟩
15900	289	3	N	⟨price, 1.0⟩
John	289	6	S	⟨name, 1.0⟩, ⟨firstname, 0.9⟩
Toyota	354	1	S	⟨make, 1.0⟩,⟨manufacturer, 0.9⟩
Camry	354	2	S	⟨model, 1.0⟩
Yamada	354	5	S	⟨lastname, 1.0⟩
Taro	354	6	S	⟨firstname, 1.0⟩
Honda	571	1	S	⟨manufacturer, 1.0⟩, ⟨brand, 0.5⟩
Civic	571	2	S	⟨model, 1.0⟩, ⟨name, 0.3⟩
16000	571	3	N	⟨price, 1.0⟩
Brown	571	5	S	⟨lastname, 1.0⟩, ⟨name, 0.6⟩
John	571	6	S	⟨firstname, 1.0⟩, ⟨name, 0.8⟩
Ford	702	1	S	⟨make, 1.0⟩, ⟨brand, 0.4⟩
Focus	702	2	S	⟨model, 1.0⟩, ⟨name, 0.2⟩
Tom	702	4	S	⟨firstname, 1.0⟩
Davidson	702	5	S	⟨lastname, 1.0⟩
Volvo	886	1	S	⟨make, 1.0⟩
XC90	886	2	S	⟨name, 1.0⟩

Table 6.3: An illustration of probabilistic tagging

a. Attribute Names

Since tuples in a wide-table are typically inserted by users with a schema in mind, each of these values in the tuple will thus be associated with an attribute name. Thus, a trivial way to associate at least one tag with each of the values is to assign the attribute name as a tag.

b. User Specified

Alternatively, a user might want to simply specify a tag for a group of values. For example, he/she might issue a command `tag(BMW,manufacturer,0.9)` which will tag all values `BMW` with tag `manufacturer`. Obviously, there exists many alternative approaches for users to specify associations between tags and values. Furthermore, checks should be in place to ensure that users do not add in spurious tags which make no meaning. However, this approach requires a lot of work to make sure all values are tagged properly.

c. Tag Inference from Private Tables

In the case where a user already has her private tables and hopes to append more tuples from a wide-table into her private tables, it is possible to adopt machine learning techniques like Markov random field [SC04] in order to match columns in the wide-table against the columns in the private table. If we regard columns in her private table as tags, a Markov model will then be trained for each tag in the private table and each value will be matched probabilistically against these tags.

d. Tag Inference without Private Tables

When no private table is available, we propose to infer alternative tags for values. For instance, if `BMW` is tagged by `manufacturer`, a tag-value pair

`(BMW,make)` can be inferred. The problem of tag inference without private tables are defined as follows.

Definition 6.2 (Tag Inference)**.** *Given (i) a value v, (ii) a set of tags $\mathcal{G}_v$ used for tagging v, and (iii) the associated probabilities $\zeta_{v,g_0}, \forall g_0 \in \mathcal{G}_v$; determine $\zeta_{v,g}$, the associated probability of v being tagged by g, $\forall g \in \mathcal{G}$, where $\mathcal{G}$ is the global set of tags.*

However, it is not trivial to do such tag inference as there are two major difficulties to find tags which are closely related:

- Data values are heterogeneous. If we were to compare whether two tags are similar by the two sets of values each of them is associated with, the similarity between the two tags depends on the similarity of the two sets of values. However, the two sets of values may be similar with no or little overlaps, which leads to underestimation of the tag similarity. Thus describing the tags by their associated data values already poses a problem.

- More importantly, even if there exists such a description, tag inference is not only simply decided by how close the tags are, but also determined by the "scope" of the tags. The "scope" of a more specific tag is relatively small, thus the distance may look larger under such a scenario. Therefore, the "scope" of the tags must be also taken care of when doing inferences among tags.

In Section 6.3, we will look at how tag inference can be done with some of the techniques we developed in Chapter 4 and 5.

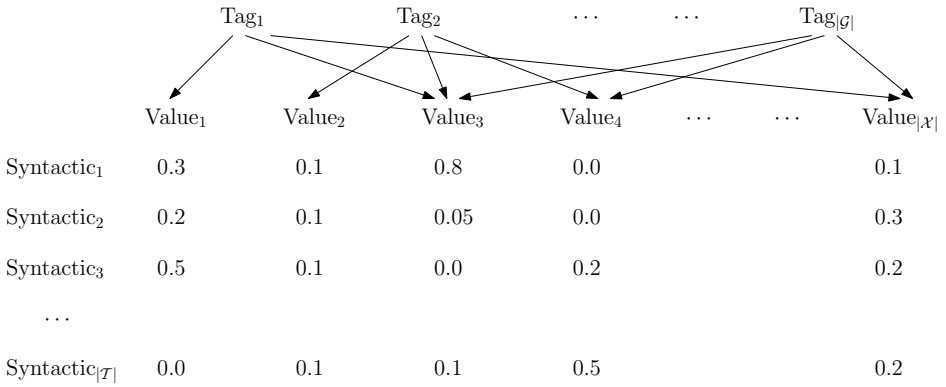


Figure 6.1: Representing the semantics of the tags by the syntactic types

6.3 Semantic Inference

Let us first understand how human beings judge whether two tags being similar or not. As when human beings look at the values associated with one tag, they will not memorize the values. Instead, they glance at the values, and get an idea of "how the values look like", i.e., the syntactic types which we have discussed in Chapter 4 and 5. For example, tag `price` and tag `cost` are semantically similar, but there may not be any overlap between the two sets of values associated with `price` and `cost` respectively. Thus they are far apart as the majority of the values in the two sets are different. When a human being reads these values, he/she tells that they are similar since the *syntactic type* of the values are similar. Therefore, the similarities between tags have to be elevated from the "value" level to the "type" level.

Therefore, our central idea of describing semantics is to

> *Represent the Semantics of the Tags in the space of Syntactic Types.*

However, unlike a data value, which can be directly mapped to a probabilistic distribution over the syntactic types through its representation in a

feature space Y, e.g., the space of q-grams; the representations of tags over the syntactic types are not so straightforward as there are no representations in the corresponding feature space Y for the tags. Therefore, the only way to obtain the representations for the tags in the space of syntactic types is through the representations of values they are associated with.

As shown in Figure 6.1, each value is represented by a probability distribution over the syntactic types. Each tag is associated with a set of values, thus the syntactic type representations for the tags can be inferred from the representations of the values. Note the feature space $\mathcal{Y}$ is not present in this figure. The details are as follows.

6.3.1 The representation of the Semantics

In Chapter 4, the syntactic types for each value is identified by a clustering process. Given value $x \in \mathcal{X}$, represented as a probabilistic distribution in feature space Y, i.e., $p(y|x)$ with the constraint that $\sum_y p(y|x) = 1$, the heterogeneity algorithm, i.e., Algorithm 1 in Section 4.3.3, computes a set of syntactic types $\mathcal{T}$ where each x is represented by a probabilistic distribution $p(t|x)$ in the space of $\mathcal{T}$.

As introduced in Section 6.2, since each value is from a different source or inserted by a different user, it is associated with only one or a small set of tags, which are often regarded as column names. We wish to set up associations between values and tags from a different source, which are originally unknown. For instance, `toyota` is tagged by `make` with associated probability 1.0, given tag `manufacturer` and `model` from another data source, a high associated probability $\zeta_{\texttt{toyota,manufacturer}}$ but a low associated probability $\zeta_{\texttt{toyota,model}}$ are expected because `make` is intuitively closer to `manufacturer` than to `model`.

Given a tag $g \in \mathcal{G}$, the representation of g in $\mathcal{T}$ is computed by the values which are initially associated with g as follows.

For all $t \in \mathcal{T}$,

$$\begin{aligned} p(t|g) &= \frac{p(t,g)}{p(g)} \\ &= \frac{\sum_{x\in\mathcal{X}} p(t,x,g)}{\sum_{x\in\mathcal{X}} p(x,g)} \\ &= \frac{\sum_{x\in\mathcal{X}} p(t|x)p(x,g)}{\sum_{x\in\mathcal{X}} p(x,g)} \\ &= \frac{\sum_{x\in\mathcal{X}} p(t|x)\zeta_{x,g}}{\sum_{x\in\mathcal{X}} \zeta_{x,g}} \end{aligned} \tag{6.1}$$

Note that before doing tag inference, $\zeta_{x,g}$ is proportional to the number of times observing value x with tag g, i.e., $p(x,g)$, thus $p(x,g)$ is replaced by $\zeta_{x,g}$ in Equation 6.1. As $\{p(t|x)\colon t \in \mathcal{T}\}$ is a probability distribution,

$$\sum_{t\in\mathcal{T}} p(t|g) = \frac{\sum_{x\in\mathcal{X}}\sum_{t\in\mathcal{T}} p(t|x)\zeta_{x,g}}{\sum_{x\in\mathcal{X}} \zeta_{x,g}} = 1$$

This implies Equation 6.1 yields a probability distribution, which is the probability distribution of tag g in the space of syntactic types T. There are two factors that determine the representation of tag g. The first factor is the set of type distributions of values originally associated with g, i.e. $p(t|x)$, and the other factor is how these values are distributed with g, i.e., $\zeta_{x,g}$. Generally speaking, two tags g_1 and g_2 are close if their associated values are similar in their syntactic types, and the distribution of the values are similar.

We can also think $\{p(t|g)\colon t \in \mathcal{T}\}$ as the weighted mean of $\{p(t|x)\colon t \in \mathcal{T}\}$ for those originally associated values. So $\{p(t|g)\colon t \in \mathcal{T}\}$ represents the overall syntactic types of those values.

6.3.2 The Scope of the Semantics

In order to make inferences from one tag to another, the distances between tags need to be quantified. With the representation of tags as a probabilistic distribution over syntactic types $\mathcal{T}$, the distance between tags can be determined by the distance between the two probabilistic distributions. However, such point-to-point distance function ignores the fact that the type distribution of the tag is determined by other type distributions, and does not capture the "scope" of the values associated with each tag in the space of syntactic types.

We therefore need to take the syntactic types of the associated values into consideration. Let T_g be the random variable taking values in the syntactic type space $\mathcal{T}$ with the probability distribution $\{p(t|g) \colon t \in \mathcal{T}\}$, i.e., $p(t) = p(T_g = t \in \mathcal{T}) = p(t|g)$. Now we consider the mutual information $I(X; T_g)$.

$$\begin{aligned} I(X; T_g) &= \sum_{x \in \mathcal{X}} p(x) D_{KL}[p(t|x) \| p(t)] \\ &= \sum_{x \in \mathcal{X}} p(x) D_{KL}[p(t|x) \| p(t|g)] \end{aligned} \tag{6.2}$$

Hence, $I(X; T_g)$ is now the weighted KL divergence from all $x \in \mathcal{X}$ to g in the space of $\mathcal{T}$.

For two tags g_1 and g_2, let the two sets of values associated to g_1 and g_2 are $\mathcal{X}_{g_1}$ and $\mathcal{X}_{g_2}$ respectively. If g_1 and g_2 are similar in their semantics, the overall distance from either $\mathcal{X}_{g_1}$ or $\mathcal{X}_{g_2}$ to either g_1 or g_2 should be similar, we therefore consider this quantity

$$\frac{I(X_{g_1}; T_{g_2})}{I(X_{g_1}; T_{g_1})} + \frac{I(X_{g_2}; T_{g_1})}{I(X_{g_2}; T_{g_2})}$$

We give the following lemma before we explain the quantity above.

Lemma 6.3 (Bregman Information Equality)**.** *Let X be a random variable that takes values in $\mathcal{X} = \{x_i\}_{i=1}^n \subseteq \mathbb{R}^d$ following the probability measure $\pi = \{\pi(x_i)\}_{i=1}^n$, given a Bregman divergence ϕ and another point y,*

$$\mathbb{E}_\pi[d_\phi(\mathcal{X}, y)] = \mathbb{E}_\pi[d_\phi(\mathcal{X}, \mu)] + d_\phi(\mu, y)$$

where $\mu = \mathbb{E}_\pi[\mathcal{X}]$.

Proof.

$$\begin{aligned}
& \mathbb{E}_\pi[d_\phi(\mathcal{X}, y)] = \sum_{i=1}^n \pi(x_i) d_\phi(x_i, y) \\
= & \sum_{i=1}^n \pi(x_i)\{\phi(x_i) - \phi(y) - \langle x_i - y, \nabla\phi(y)\rangle\} \\
= & \sum_{i=1}^n \pi(x_i)\{\phi(x_i) - \phi(\mu) - \langle x_i - \mu, \nabla\phi(\mu)\rangle \\
& + \langle x_i - \mu, \nabla\phi(\mu)\rangle + \phi(\mu) - \phi(y) - \langle x_i - \mu + \mu - y, \nabla\phi(y)\rangle\}
\end{aligned}$$

As

$$\sum_{i=1}^n \pi(x_i)(x_i - \mu) = 0$$

We then have both

$$\sum_{i=1}^n \pi(x_i)\langle x_i - \mu, \nabla\phi(\mu)\rangle = \sum_{i=1}^n \pi(x_i)\langle x_i - \mu, \nabla\phi(y)\rangle = 0$$

Hence,

$$\begin{aligned}
& \mathbb{E}_\pi[d_\phi(x, y)] \\
= & \sum_{i=1}^n \pi(x_i)\{\phi(x_i) - \phi(\mu) - \langle x_i - \mu, \nabla\phi(\mu)\rangle \\
& + \langle x_i - \mu, \nabla\phi(\mu)\rangle + \phi(\mu) - \phi(y) - \langle x_i - \mu + \mu - y, \nabla\phi(y)\rangle\} \\
= & \sum_{i=1}^n \pi(x_i)\{\phi(x_i) - \phi(\mu) - \langle x_i - \mu, \nabla\phi(\mu)\rangle\} \\
& + \sum_{i=1}^n \pi(x_i)\{\phi(\mu) - \phi(y) - \langle \mu - y, \nabla\phi(y)\rangle\} \\
= & \mathbb{E}_\pi[d_\phi(x, \mu)] + d_\phi(\mu, y)
\end{aligned}$$

□

By specifying $\mathcal{X} = \mathcal{X}_{g_1}$, $\pi(x_i) = \frac{\zeta_{x_i,g_1}}{\sum_x \zeta_{x,g_1}}$ and x_i being the probabilistic distribution $p(t|x_i)$ in $\mathcal{T}$, then $\mu = \sum_{i=1}^{n} \pi(x_i)p(t|x_i) = p(t|g_1)$.

By [BMDG05], let $\phi(p) = \sum p_j \log p_j$ be a strictly convex function on probability distributions, and thus $d_\phi(x, y) = D_{KL}[p(t|x)\|p(t|y)]$. Taking $y = p(t|g_2)$, Lemma 6.3 shows

$$\begin{aligned} I(X_{g_1}; T_{g_2}) &= \sum_{i=1}^{n} \pi(x_i) D_{KL}[p(t|x_i)\|p(t|g_2)] \\ &= \mathbb{E}_\pi[D_{KL}[p(t|x)\|p(t|g_2)]] \\ &= \mathbb{E}_\pi[D_{KL}[p(t|x)\|p(t|g_1)]] + D_{KL}[p(t|g_1)\|p(t|g_2)] \\ &= \sum_{i=1}^{n} \pi(x_i) D_{KL}[p(t|x_i)\|p(t|g_1)] + D_{KL}[p(t|g_1)\|p(t|g_2)] \\ &= I(X_{g_1}; T_{g_1}) + D_{KL}[p(t|g_1)\|p(t|g_2)] \end{aligned} \tag{6.3}$$

Similarly, we have

$$I(X_{g_2}; T_{g_1}) = I(X_{g_2}; T_{g_2}) + D_{KL}[p(t|g_2)\|p(t|g_1)]$$

Therefore,

$$\begin{aligned} & \frac{I(X_{g_1}; T_{g_2})}{I(X_{g_1}; T_{g_1})} + \frac{I(X_{g_2}; T_{g_1})}{I(X_{g_2}; T_{g_2})} \\ = \; & \frac{I(X_{g_1}; T_{g_1}) + D_{KL}[p(t|g_1)\|p(t|g_2)]}{I(X_{g_1}; T_{g_1})} + \frac{I(X_{g_2}; T_{g_2}) + D_{KL}[p(t|g_2)\|p(t|g_1)]}{I(X_{g_2}; T_{g_2})} \\ = \; & 2 + \frac{D_{KL}[p(t|g_1)\|p(t|g_2)]}{I(X_{g_1}; T_{g_1})} + \frac{D_{KL}[p(t|g_2)\|p(t|g_1)]}{I(X_{g_2}; T_{g_2})} \end{aligned}$$

We now define the distance function between tag g_1 and g_2 as

$$d(g_1, g_2) = \frac{D_{KL}[p(t|g_1)\|p(t|g_2)]}{I(X_{g_1}; T_{g_1})} + \frac{D_{KL}[p(t|g_2)\|p(t|g_1)]}{I(X_{g_2}; T_{g_2})} \tag{6.4}$$

The definition implies the following two corollaries directly.

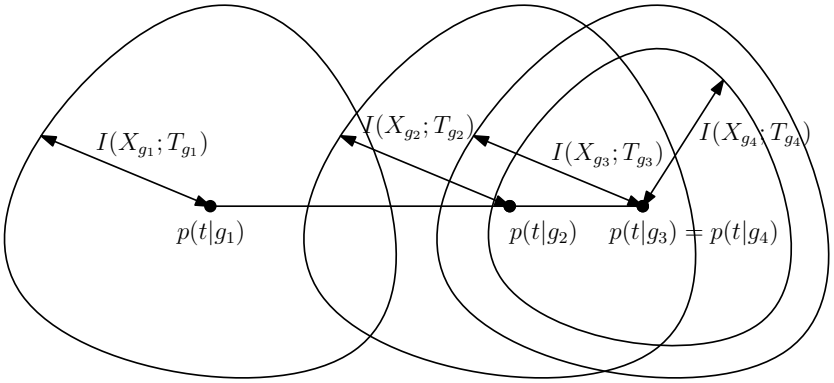


Figure 6.2: An illustration of the tag distance function

Corollary 6.4.

$$d(g_1, g_2) \geq 0$$

$$d(g_1, g_2) = 0 \iff p(t|g_1) = p(t|g_2)$$

Corollary 6.5.

$$d(g_1, g_2) = d(g_2, g_1)$$

Equation 6.4 captures both the direct distance between the syntactic type representations of g_1 and g_2, i.e., $p(t|g_1)$ and $p(t|g_2)$, as well as the scope of g_1 and g_2, i.e., $I(X_{g_1};T_{g_1})$ and $I(X_{g_2};T_{g_2})$. Thus the direct distance is "normalized" by the scope of the two tags.

We now give a few examples as shown in Figure 6.2. The closures are the contour with equal KL divergence to their centers. This distance is the mutual information between X_g and T_g as defined in Lemma 6.3, so $I(X_{g_1};T_{g_1})$, $I(X_{g_2};T_{g_2})$ and $I(X_{g_3};T_{g_3})$ are all about the same. Since $p(t|g_3)$ is closer to $p(t|g_2)$ than $p(t|g_1)$, we have $d(g_1, g_2) > d(g_2, g_3)$. Second, when the KL divergences are the same, the distance increases as mutual information decreases, as it makes distance relatively larger. In Figure 6.2, $p(t|g_3)$ and

$p(t|g_4)$ have the same distance to $p(t|g_2)$. However, $d(g_2, g_3) < d(g_2, g_4)$ as $I(X_{g_3}; T_{g_3}) > I(X_{g_4}; T_{g_4})$.

Note that with the help of Lemma 6.3, the computation of this distance function only involves the mutual information about each tag and pairwise KL divergences between tags, instead of computing the mutual information between each value set $\mathcal{X}$ and each tag $p(t|g)$.

6.3.3 Tag Inference

With the tag distance defined above, we can now perform our tag inference by taking the power of the negative distance, i.e. $e^{-\alpha \cdot d(g_1, g_2)}$. α is a parameter which controls the power of inference. When α is set to be 0, every value is associated with every tag; when α is set to be infinity, there is no inference among tags.

Suppose value x is associated to tag g_1, g_2, ..., g_m with probability $\zeta_{x,g_1}, \zeta_{x,g_2}, \ldots, \zeta_{x,g_m}$ respectively, given a new tag g, the probability of x being tagged by g is thus

$$\max_{i=1}^{m} \zeta_{x,g_i} \cdot e^{-\alpha \cdot d(g, g_i)} \tag{6.5}$$

In practice, only tags with probability above a certain threshold is worth considering. Therefore, when $d(g, g_i)$ is large for all g_i, the probability $\zeta_{x,g}$ will be very small, the association between x and g will thus not be recorded.

We now summarize the algorithm to compute tag inferences as follows.

The details about mapping values into a feature space Y will be discussed next.

Algorithm 3 Computing Tag Inference

1: Map all values into a feature space Y
2: Compute soft clustering $p(t|x)$ using `iIB` for all x
3: **for all** tag $g \in \mathcal{G}$ **do**
4: Compute $p(t|g)$ according to Equation 6.1
5: Compute $I(X_g; T_g)$ according to Equation 6.2
6: **end for**
7: **for all** pair $(g_1, g_2) \in \mathcal{G} \times \mathcal{G}$ **do**
8: Compute tag distance $d(g_1, g_2)$ according to Equation 6.4
9: **end for**
10: **for all** value $x \in \mathcal{X}$ **do**
11: **for all** tag $g \in \mathcal{G}$ **do**
12: **if** x was not originally tagged by g **then**
13: Inference g to x by Equation 6.5
14: **end if**
15: **end for**
16: **end for**

6.4 Methodology

In this section, we will propose the idea of using the "signatures" of values for clustering. Values are considered as either string values, or numerical values, and two different kinds of signatures are developed for these two types respectively.

6.4.1 Sigram: Signatures for String Values

Observing that many tags are associated with categorical data values, e.g., values associated with tag `make` include `ford`, `toyota`, `audi`, etc. Using q-gram with small q to represent such values results in many q-grams having the same frequency. For example, when $q = 2$, 2-grams `oy` and `ot` only appear in `toyota` among all categorical values associated with tag `make`. These two 2-grams are highly correlated since the two always appear together when the value is `toyota`. Therefore, the dimensionality of the feature space can be reduced if such correlated dimensions are combined together, i.e., instead of having all the 2-grams `to`, `oy`,`yo`,`ot` and `ta` from `toyota`, we combine them in one single dimension, which is `toyota`.

This problem has been discussed by papers including Lee et al.[LNS07] and Li et al.[LWY07], where [LNS07] introduced a wild-card to the alphabets and [LWY07] proposed *VGRAM*, which is a set of variable length q-grams of high quality. In this thesis, we would like to capture those significant q-grams which are long and frequent, which are called *sigram*.

Definition 6.6 (Immediate Sub-string)**.** *We define string A as an immediate sub-string of string B if*

(i) A is a sub-string of B; and

(ii) $|A| = |B| - 1$.

From this definition, it is natural to derive the next related definition.

Definition 6.7 (Immediate Super-string)**.** *B is an immediate super-string of A if A is an immediate sub-string of B.*

Considering the case where a q-gram A is as frequent as its immediate super-string B, we can say B is more significant than A as all strings that contain A also contain B. Therefore, a q-gram is significant if it is maximal, i.e., if it does not have any immediate super-string which is as frequent as itself. We thus define *sigram* as

Definition 6.8 (Sigram)**.** *A q-gram is called a sigram with respect to a set of string values if*

(i) it is frequent; and

(ii) No immediate super-string is as frequent as itself

A significant q-gram behaves much like a maximal frequent itemset [HK00]. If we consider each character as an "item", a string is then a sequential "itemset", the sigram is thus an analog to maximal frequent itemset.

Figure 6.3 shows a lattice from a column containing three strings: pink, ping and sung. Each arrow represents an immediate sub-string relationship from a string to one of its immediate sub-strings. The frequency of each string is written on the right of the string. Although q-gram "in" appears twice out of three times, it is not considered as a sigram as one of its immediate super-string, which is "pin" appears the same number of times as itself. Therefore, this q-gram is not a sigram according to our second criterion. "pin" itself is a sigram as there is no immediate super-string that is as frequent.

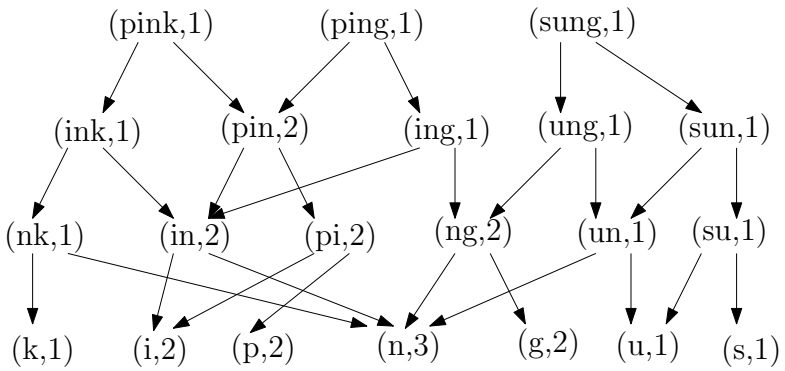


Figure 6.3: Illustration of Immediate sub-string

Note that the frequency of any immediate super-string of a q-gram is always upper bounded by the frequency of the q-gram itself. A q-gram will be filtered by the second criterion only when it has only one immediate super-string. But this seldom happens, and most q-grams can pass the second criterion, which makes the set of sigrams merely a set of frequent q-grams by the first criterion. In order to select significant ones out of the frequent q-grams, we need to relax the second criterion by parameter η. We redefine the second criterion as "No immediate super-string with frequency greater than η of the frequency of itself".

Definition 6.9 (Sigram with Parameters ϵ and η)**.** *Given a pair of parameters ϵ and η, a q-gram A is a sigram if*

(i) $Freq_A > \epsilon$

(ii) $\nexists B$ *s.t.* $Freq_B > \eta \cdot Freq_A$ *and B is an immediate super-string of A*

Take Figure 6.3 for example, if we set $\epsilon = \eta = 0.5$, the q-grams that still remain after the first criterion are {pin, in, pi, ng, i, p, n, g} as they appear at least twice out of 3 times. By the second criterion, the only q-grams left are {pin, ng}, which is the set of sigrams.

Apriori Property In the definition above, we call string A frequent if $Freq_A > \epsilon$. This definition implies if A is frequent, then both its immediate sub-strings are frequent. The contrapositive of this statement leads to the *Anti-Monotone* property [HK00], which says if a set fails a test, all of its supersets will fail as well. In our content, if a string is not frequent, all of its immediate super-strings are not frequent as well. As a result, we can use an Apriori algorithm to reduce the search space for sigrams.

In our problem setting, instead of having one string completely "belong" to a set, e.g., a column, strings are associated to tags with probabilities. The frequency of a string is then the accumulated associated probabilities. For the pair of parameters ϵ and η, the algorithm to select all sigrams is presented in Algorithm 4. Each q-gram Q is associated with a pair of integers: i) $Q.w_i$: the accumulated associated probabilities of itself, and ii) $Q.w_s$: the maximum of the accumulated associated probabilities of its immediate super-string.

6.4.2 Signatures for Numerical Values

The second type of tag, i.e., numerical tag, is associated with numerical values that cannot be treated as strings to make sigrams as their "signature". As numerical values are often continuous, which makes it difficult to map into a set of dimensions, thus, we take histograms as the "signature" of a numeric tag. Given a parameter m, we can represent a range of values by $2m$ dimensions, as presented in Algorithm 5

Algorithm 5 builds a set of dimensions ND_g from a set of values associated with numerical tag g. Each dimension is of width w on the numerical domain, spreading from the median for at most m stripes towards both sides. Tag g is now represented by a probability histogram over ND_g. Given two tags g_1 and g_2 whose values are distributed similarly, the histograms obtained from

Algorithm 4 Selection of Sigrams from string values associated with tag g

Input: A set of string values $x \in \mathcal{X}$ associated to tag g with $\zeta_{x,g}$

A pair of parameters ϵ and η

Output: A set of sigram SQ_g as a set of dimensions

$\zeta_g \leftarrow \sum_{x \in \mathcal{X}} \zeta_{x,g}$
$k \leftarrow 1$
while $k = 1$ or $A_{k-1} \neq \emptyset$ **do**
  $A_k \leftarrow \emptyset$
  for all string value $x \in \mathcal{X}$ **do**
    for all k-gram Q appears in x **do**
      $A_k \leftarrow A_k \cup \{Q\}$
      $Q.w_i$ += $\zeta_{x,g}$;
    end for
  end for
  for all $Q \in A_k$ **do**
    for all Q', an immediate sub-string of Q, where $k > 2$ **do**
      $Q'.w_s = \max\{Q'.w_s, Q.w_i\}$
    end for
    Remove Q from A_k **if** $k > 1$ and $Q.w_i \leq \epsilon \cdot \zeta_t$
  end for
  for all $Q \in A_{k-1}$ where $k > 2$ **do**
    Remove Q from A_{k-1} **if** $Q.w_s > \eta \cdot Q.w_i$
  end for
  k ++;
end while
Return $SQ_g = \bigcup_k A_k$

Algorithm 5 Construction of numerical dimensions from numerical values associated with tag g

Input: A set of numerical values $x \in \mathcal{X}$ associated to tag g with $\zeta_{x,g}$
A granularity parameter m

Output: A set of numerical dimensions ND_g

1: $\zeta_g \leftarrow \sum_{x \in \mathcal{X}} \zeta_{x,g}$
2: Get median $x_m = \arg\max_{x_0} \sum_{x<x_0} \zeta_{x,g} \leq \frac{1}{2}\zeta_g$, lower quartile $x_l = \arg\max_{x_0} \sum_{x<x_0} \zeta_{x,g} \leq \frac{1}{4}\zeta_g$, upper quartile $x_u = \arg\max_{x_0} \sum_{x<x_0} \zeta_{x,g} = \frac{3}{4}\zeta_g$, minimum value x_{min} and maximum value x_{max} from $\mathcal{X}$
3: Let $w \leftarrow \frac{x_u - x_l}{m}$
4: Let $ND_g \leftarrow \emptyset$: the set of numerical dimensions constructed by g
5: Let $n_l \leftarrow \min(m, ceil(\frac{x_m - x_{min}}{w}))$
6: **for all** $i \in \{1, 2, \ldots, n_l\}$ **do**
7: Construct numerical dimension on interval $[x_m - i \cdot w, x_m - (i-1) \cdot w)$ (the most left dimension has no lower bound)
8: insert this dimension into ND_g
9: **end for**
10: Let $n_r \leftarrow \min(m, ceil(\frac{x_{max} - x_m}{w}))$
11: **for all** $i \in \{1, 2, \ldots, n_r\}$ **do**
12: Construct numerical dimension on interval $[x_m + (i-1) \cdot w, x_m + i \cdot w)$ (the most right dimension has no upper bound)
13: insert this dimension into ND_g
14: **end for**
15: Return ND_g

the numerical dimensions ND_{g_1} and ND_{g_2} will be similar too.

There may be overlapping dimensions between numerical dimensions constructed by different tags. So when numerical dimensions are put together, it is a partition on the numerical domain, thus this results in certain degree of redundancy.

Together with the set of sigrams that are obtained from string tags, we now have a set of features/dimensions $\mathcal{Y}$. Note that space $\mathcal{Y}$ in Chapter 4 and 5 are the set of q-grams since only string values are dealt with. For a value $x \in \mathcal{X}$, $p(Y|x)$ represents the probabilistic distribution of x over $\mathcal{Y}$, obtained as follows.

For a string value x and a sigram y, $p(y|x) \propto k \cdot w(y)$ if the sigram y appears in value v for k times; for numerical value x and a numerical dimension y, $p(y|x) \propto w(y)$ if $y.\min \leq x < y.\max$, where $w(y)$ is the weighting function on dimension y. $p(Y|x)$ is then normalized so that $\sum_{y \in \mathcal{Y}} p(y|x) = 1$.

The clustering part of the algorithm is similar to what we described in Chapter 4. We now proceed to the experimental study to validate the measure of tag distance we proposed.

6.5 Experiments

6.5.1 The Effectiveness of the Tag Distance

We first validate our distance function by finding out which tags are near to each other. 40 tables in the `manyeyes` car database, introduced in Section 4.5.1, and one table from `GoogleBase` are selected. The column names are taken as tags, where tags starting with `g_` are attributes from `GoogleBase`. Values are sampled at the rate of 10%, the upper bound for the number of clusters, K, is set at 200. The result is shown in Table 6.4. Most pairs of tags

identified by very small distances are semantically similar, which is suggested by the names of the tags. We have also examined values from the column, and the human judgments on whether two tags are similar are indicated in the last column in Table 6.4. Note that our method is a pure syntactic type based method, it does not include some simple external heuristic, e.g., by looking at the tag names. Thus there are tag pairs with similar values on their syntactic types, but with different semantics. This is the case for `g_price` and `mileage` since they are both of numerical type and ranging from couples of thousands to several tens of thousands. But the majority of the tag pairs identified are consistent in their semantics. Our tag inference is thus effective in general.

6.5.2 The Robustness of the Tag Distance

We further validate our approach by some semi-real data with controlled distributions to show our measure between tag pairs is robust. As `GoogleBase` provides options on attribute names for users to upload data, the entire database is in fact quite well organized, and is unlikely that there are two attributes carrying the similar semantics. There are 15 attributes and about 2 million data values. We pre-processed `GoogleBase` dataset as follows.

For each of the attribute, we created a number of different tags with the same prefix. For example, we created another 9 tags `make_a`, `make_b`, ... `make_i` from the original attribute name `make`. These tags are used to simulate different tags from heterogeneous data sources, but with the same semantics. As we noticed that the number of databases decreases as the size of database increases, these 10 tags are divided into four groups with different weighting schemes.

Given a positive parameter p, the first group contains 4 tags, `make`,

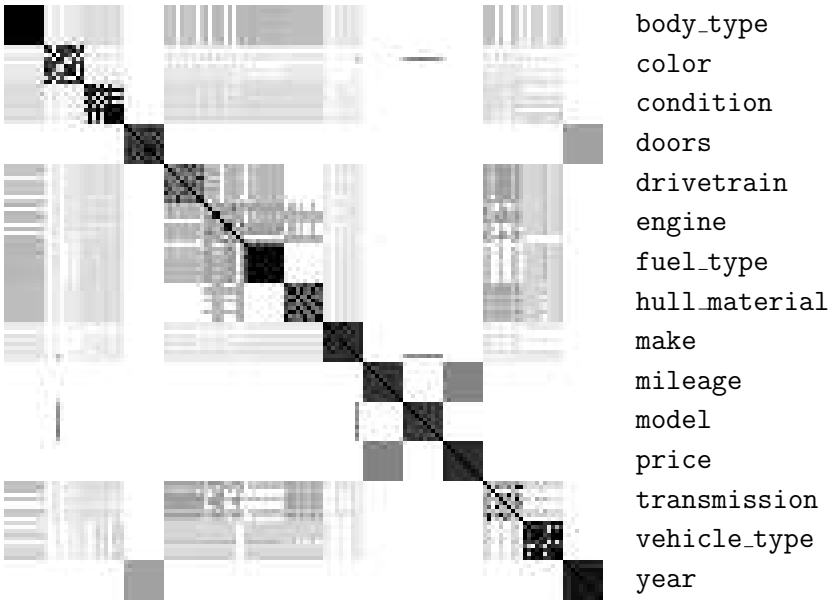


Figure 6.4: Tag distance bitmap: the closer two tags are, the darker the pixel is

`make_a`, `make_b` and `make_c`, which are assigned with weight 1^p, the second group contains 3 tags, `make_d`, `make_e` and `make_f`, which are assigned with weight 2^p, the third group has only two tags, `make_g` and `make_h`, having weight 3^p, the last group, with the sole attribute `make_i`, is assigned with weight 4^p.

With different weights assigned to each tag, we can calculate a probability distribution with probabilities proportional to their weights. The next is to assign every value to a tag according to this probability distribution. The assignment will give associated probability 1, but associated probabilities with other tags are to be inferred. For example, if $p = 0$, the tags are uniformly randomly assigned, if $p = 1$, the probability for each tag in the first group is 5%, and becomes 10%, 15% and 20% for each tag in the second, third and last group respectively. In the experiment, we consider each tag as a tag,

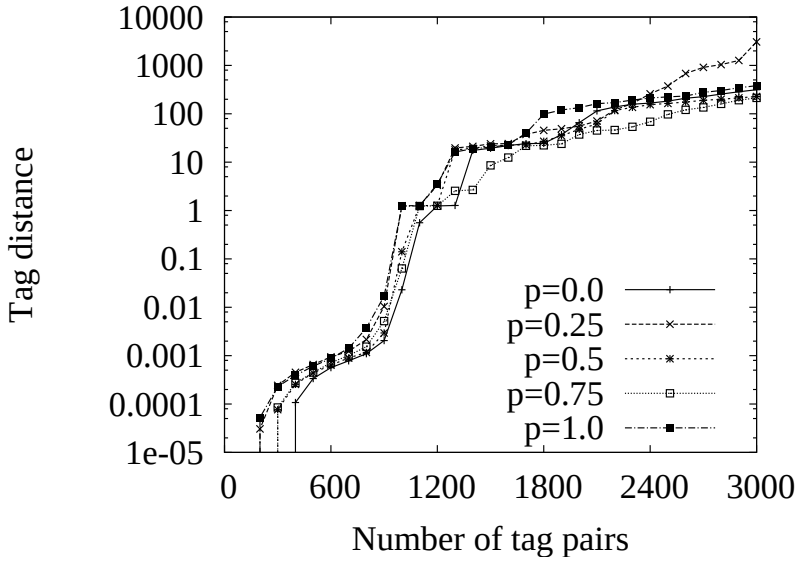


Figure 6.5: Tag distance steady increases regardless of the parameter p

and then obtain the tag inference after clustering the values. The clustering ran 3 times with a different initialization on a 2% sample (∼40 thousand data values), while tags generated about 800 dimensions. We expect tags with the same original attribute name to be high.

Figure 6.4 visualizes the pairwise distance between tags. Each row and column represents a tag, the distance between the row tag and the column tag at a given pixel is inversely related to the darkness of the pixel. Note that the tags are arranged in the same order in both the row and column orientation, and tags with the same original attribute name are grouped together. Therefore, the small squares on the diagonal are dark as the distances between tags with the same original attribute name are small. The outlier is the fourth last attribute `price` and the sixth last attribute `mileage`. Their off diagonal intersections still give a moderate greyscale, meaning that these tags, although not having the same original attribute name, are closer to one

another.

Figure 6.5 shows how the maximum of the smallest-k all pair distances change as varying parameter p. Firstly, tag distance is very small when k is less than 1000. As shown in Figure 6.4, these small distances come from tags with the same original attribute name. Secondly, the increment of the tag distance is stable regardless of the parameter p, i.e. regardless of how the distributions are. Therefore, our distance measure is robust.

6.5.3 Performance

As there are about 2 million data values in the `GoogleBase` dataset, we only sampled 10% to compute the clusters of types. This clustering of these 200 thousand values took about 1 hour. With these clusters, we then compute the cluster distribution for each of the 2 million values, which took about half an hour. The computation of tag distance is then fast once the cluster distributions are computed.

Although it spent more than an hour to compute the pair-wise tag distances for 150 tags, the re-computation of tag distances is not carried often. Only when there is a major change in the type distribution for one tag, the tag distances involving this particular tag ought to be updated, which incurs partial computation of the cluster distributions.

6.6 Summary

In this chapter, we actually made use of column heterogeneity, i.e., the type distribution for each tag, to infer the semantics of the tags. Hence the tag distances and tag similarities are computed based on the type distribution of the tags. The tag distance between two tags considered both the type

distributions and their scopes. We have also proposed signatures for string values, i.e., sigrams, as well as signatures for numerical values, to improve the clustering efficiency. The tag distance is validated on both semi-synthetic dataset and real dataset, and we concluded that the proposed tag distance is sound and robust.

Tag 1	Tag 2	Distance	Similar?
drive_wheels	drive	0.000	√
hwy_mpg	unadj_cmb_mpg	0.010	√
highway_mpg	hwy_mpg	0.036	√
retail_price	dealer_cost	0.038	√
g_color	color	0.040	√
power	hp	0.044	√
g_make	manufacturer	0.071	√
highway_mpg	unadj_cmb_mpg	0.077	√
bore	stroke	0.127	√
highway_mpg	epa_hwy_mpg	0.165	√
city_mpg	mpg	0.233	√
highway_mpg	mpg	0.264	√
epa_hwy_mpg	hwy_mpg	0.272	√
epa_hwy_mpg	unadj_cmb_mpg	0.304	√
cyl	cylinders	0.306	√
city_mpg	epa_city_mpg	0.309	√
make	manufacturer	0.388	√
hwy_mpg	mpg	0.424	√
epa_hwy_mpg	mpg	0.442	√
mpg	unadj_cmb_mpg	0.468	√
g_price	mileage	0.516	×
cyl	air_pollution_score	0.635	×
city_mpg	highway_mpg	0.732	√
epa_city_mpg	mpg	0.741	√
myear	year	0.758	√
g_price	dealer_cost	0.801	√
g_year	myear	0.989	√

Table 6.4: Tag pairs with distance less than 1

Chapter 7

Conclusion

In this thesis, we have studied the problem of column heterogeneity from three aspects. The column heterogeneity measure is first addressed in Chapter 4, the schema matching validator is then proposed in Chapter 5 and the probabilistic tagging is finally discussed in Chapter 6.

- In Chapter 4, we built a foundation for the whole thesis, i.e., identification of the syntactic types from one single column, and the identified syntactic types give the measure of the column heterogeneity.
- This is followed by Chapter 5, where syntactic types identified from multiple columns are put together to tell how well these columns can be integrated. Additional techniques are developed for partial matches as well.
- In Chapter 6, the syntactic types are further extended to semantics by considering semantics as distributions in the space of syntactic types, so as to infer probabilistic association between values in one column to those similar columns.

These chapters together give an overall framework from intra-column heterogeneity to inter-column heterogeneity, and also extend from the syntactic perspective to the semantic perspective.

We claim that columns are important to databases, no matter in the conventional RDBMS, or in any kind of emerging data model. Instead of enforcing every column in a database to be homogeneous, with the series of techniques we developed in this thesis, heterogeneous columns in databases can be well managed.

With our management techniques on heterogeneous columns discussed in this thesis, we hope our methodologies can be utilized for future database technologies.

Bibliography

[Aka74] H. Akaike. A new look at the statistical model identification. *IEEE Transactions on Automatic Control*, 19:716–723, 1974.

[BCS04] C. Batini, T. Catarci, and M. Scannapieco. A survey of data quality issues in cooperative information systems. In *ER pre-conference tutorial*, 2004.

[BM01] Jacob Berlin and Amihai Motro. Autoplex: Automated discovery of content for virtual databases. In *Proceedings of the 9th International Conference on Cooperative Information Systems*, CooplS '01, pages 108–122, London, UK, 2001. Springer-Verlag.

[BM02] Jacob Berlin and Amihai Motro. Database schema matching using machine learning with feature selection. In *Proceedings of the 14th International Conference on Advanced Information Systems Engineering*, CAiSE '02, pages 452–466, London, UK, UK, 2002. Springer-Verlag.

[BMDG05] Arindam Banerjee, Srujana Merugu, Inderjit S. Dhillon, and Joydeep Ghosh. Clustering with bregman divergences. *J. Mach. Learn. Res.*, 6:1705–1749, December 2005.

[BV04] Stephen Boyd and Lieven Vandenberghe. *Convex Optimization*. Cambridge University Press, 2004.

[CBN07] Eric Chu, Jennifer Beckmann, and Jeffrey Naughton. The case for a wide-table approach to manage sparse relational data sets. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 821–832, New York, NY, USA, 2007. ACM.

[CDG+06] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: a distributed storage system for structured data. In *Proceedings of the 7th USENIX Symposium on Operating Systems Design and Implementation - Volume 7*, OSDI '06, pages 15–15, Berkeley, CA, USA, 2006. USENIX Association.

[CDG+08] Fay Chang, Jeffrey Dean, Sanjay Ghemawat, Wilson C. Hsieh, Deborah A. Wallach, Mike Burrows, Tushar Chandra, Andrew Fikes, and Robert E. Gruber. Bigtable: A distributed storage system for structured data. *ACM Trans. Comput. Syst.*, 26:4:1–4:26, June 2008.

[CGGM03] Surajit Chaudhuri, Kris Ganjam, Venkatesh Ganti, and Rajeev Motwani. Robust and efficient fuzzy match for online data cleaning. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 313–324, New York, NY, USA, 2003. ACM.

[Cod70] E. F. Codd. A relational model of data for large shared data banks. *Commun. ACM*, 13:377–387, June 1970.

[CT06a] Laura Chiticariu and Wang-Chiew Tan. Debugging schema mappings with routes. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 79–90. VLDB Endowment, 2006.

[CT06b] Thomas M. Cover and Joy A. Thomas. *Elements of Information Theory (Wiley Series in Telecommunications and Signal Processing)*. Wiley, 2006.

[DDH01] AnHai Doan, Pedro Domingos, and Alon Y. Halevy. Reconciling schemas of disparate data sources: a machine-learning approach. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, SIGMOD '01, pages 509–520, New York, NY, USA, 2001. ACM.

[Dev86] Luc Devroye. *Non-uniform random variate generation*. Springer-Verlag, 1986.

[DH05a] AnHai Doan and Alon Y. Halevy. Semantic-integration research in the database community. *AI Mag.*, 26:83–94, March 2005.

[DH05b] Xin Dong and Alon Y. Halevy. Malleable schemas: A preliminary report. In *WebDB*, pages 139–144, 2005.

[DHY07] Xin Dong, Alon Y. Halevy, and Cong Yu. Data integration with uncertainty. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 687–698. VLDB Endowment, 2007.

[DJ03] Tamraparni Dasu and Theodore Johnson. *Exploratory Data Mining and Data Cleaning*. Wiley, 2003.

[DJMS02] Tamraparni Dasu, Theodore Johnson, S. Muthukrishnan, and Vladislav Shkapenyuk. Mining database structure; or, how to build a data quality browser. In *Proceedings of the 2002 ACM SIGMOD international conference on Management of data*, SIGMOD '02, pages 240–251, New York, NY, USA, 2002. ACM.

[DKO+06] Bing Tian Dai, Nick Koudas, Beng Chin Ooi, Divesh Srivastava, and Suresh Venkatasubramanian. Rapid identification of column heterogeneity. In *Proceedings of the Sixth International Conference on Data Mining*, ICDM '06, pages 159–170, Washington, DC, USA, 2006. IEEE Computer Society.

[DKS+08] Bing Tian Dai, Nick Koudas, Divesh Srivastava, Anthony K. H. Tung, and Suresh Venkatasubramanian. Validating multi-column schema matchings by type. In *ICDE*, pages 120–129, 2008.

[DLR77] A. P. Dempster, N. M. Laird, and D. B. Rubin. Maximum likelihood from incomplete data via the em algorithm. *Journal of the Royal Statistical Society, Series B*, 39(1):1–38, 1977.

[DRS09] Nilesh Dalvi, Christopher Ré, and Dan Suciu. Probabilistic databases: diamonds in the dirt. *Commun. ACM*, 52:86–94, July 2009.

[DSDH08] Anish Das Sarma, Xin Dong, and Alon Halevy. Bootstrapping pay-as-you-go data integration systems. In *Proceedings of the 2008 ACM SIGMOD international conference on Management*

of data, SIGMOD '08, pages 861–874, New York, NY, USA, 2008. ACM.

[EXD04] David W. Embley, Li Xu, and Yihong Ding. Automatic direct and indirect schema mapping: experiences and lessons learned. *SIGMOD Rec.*, 33:14–19, December 2004.

[GS06] Rahul Gupta and Sunita Sarawagi. Creating probabilistic databases from information extraction models. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 965–976. VLDB Endowment, 2006.

[GZM09] Tingjian Ge, Stan Zdonik, and Samuel Madden. Top-k queries on uncertain data: on score distribution and typical answers. In *Proceedings of the 35th SIGMOD international conference on Management of data*, SIGMOD '09, pages 375–388, New York, NY, USA, 2009. ACM.

[HCH04] Bin He, Kevin Chen-Chuan Chang, and Jiawei Han. Discovering complex matchings across web query interfaces: a correlation mining approach. In *KDD '04: Proceedings of the tenth ACM SIGKDD international conference on Knowledge discovery and data mining*, pages 148–157, New York, NY, USA, 2004. ACM Press.

[HK00] Jiawei Han and Micheline Kamber. *Data Mining: Concepts and Techniques*. Morgan Kaufmann, 2000.

[JD03] Theodore Johnson and Tamraparni Dasu. Data quality and data cleaning: an overview. In *Proceedings of the 2003 ACM SIGMOD*

international conference on Management of data, SIGMOD '03, pages 681–681, New York, NY, USA, 2003. ACM.

[KMS04] Nick Koudas, Amit Marathe, and Divesh Srivastava. Flexible string matching against large databases in practice. In *Proceedings of the Thirtieth international conference on Very large data bases - Volume 30*, VLDB '04, pages 1078–1086. VLDB Endowment, 2004.

[KN03] Jaewoo Kang and Jeffrey F. Naughton. On schema matching with opaque column names and data values. In *Proceedings of the 2003 ACM SIGMOD international conference on Management of data*, SIGMOD '03, pages 205–216, New York, NY, USA, 2003. ACM.

[KS05] Nick Koudas and Divesh Srivastava. Approximate joins: concepts and techniques. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 1363–1363. VLDB Endowment, 2005.

[LADT11] Meiyu Lu, Divyakant Agrawal, Bing Tian Dai, and Anthony K.H. Tung. Schema-as-you-go: on probabilistic tagging and querying of wide tables. In *Proceedings of the 2011 international conference on Management of data*, SIGMOD '11, pages 181–192, New York, NY, USA, 2011. ACM.

[LC00] Wen-Syan Li and Chris Clifton. Semint: a tool for identifying attribute correspondences in heterogeneous databases using neural networks. *Data Knowl. Eng.*, 33:49–84, April 2000.

[LD09] Jian Li and Amol Deshpande. Consensus answers for queries over probabilistic databases. In *Proceedings of the twenty-eighth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, PODS '09, pages 259–268, New York, NY, USA, 2009. ACM.

[LHLG09] Boduo Li, Mei Hui, Jianzhong Li, and Hong Gao. iva-file: Efficiently indexing sparse wide tables in community systems. In *Proceedings of the 2009 IEEE International Conference on Data Engineering*, pages 210–221, Washington, DC, USA, 2009. IEEE Computer Society.

[Lin91] Jianhua Lin. Divergence measures based on the shannon entropy. *IEEE Transactions on Information Theory*, 37(1):145–151, 1991.

[LNS07] Hongrae Lee, Raymond T. Ng, and Kyuseok Shim. Extending q-grams to estimate selectivity of string matching with low edit distance. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 195–206. VLDB Endowment, 2007.

[LWY07] Chen Li, Bin Wang, and Xiaochun Yang. Vgram: improving performance of approximate queries on string collections using variable-length grams. In *Proceedings of the 33rd international conference on Very large data bases*, VLDB '07, pages 303–314. VLDB Endowment, 2007.

[MBR01] Jayant Madhavan, Philip A. Bernstein, and Erhard Rahm. Generic schema matching with cupid. In *Proceedings of the 27th International Conference on Very Large Data Bases*, VLDB '01,

pages 49–58, San Francisco, CA, USA, 2001. Morgan Kaufmann Publishers Inc.

[MRV99] George A. Mihaila, Louiqa Raschid, and María-Esther Vidal. Querying "quality of data" metadata. In *IEEE META-DATA Conference (In Proceedings)*, 1999.

[RB01] Erhard Rahm and Philip A. Bernstein. A survey of approaches to automatic schema matching. *The VLDB Journal*, 10:334–350, December 2001.

[RS08] Christopher Ré and Dan Suciu. Approximate lineage for probabilistic databases. *Proc. VLDB Endow.*, 1:797–808, August 2008.

[SAB+05] Mike Stonebraker, Daniel J. Abadi, Adam Batkin, Xuedong Chen, Mitch Cherniack, Miguel Ferreira, Edmond Lau, Amerson Lin, Sam Madden, Elizabeth O'Neil, Pat O'Neil, Alex Rasin, Nga Tran, and Stan Zdonik. C-store: a column-oriented dbms. In *Proceedings of the 31st international conference on Very large data bases*, VLDB '05, pages 553–564. VLDB Endowment, 2005.

[SC04] Sunita Sarawagi and William W. Cohen. Semi-markov conditional random fields for information extraction. In *Neural Information Processing Systems*, 2004.

[Sch78] Gideon Schwarz. Estimating the dimension of a model. *the Annals of Statistics*, 6(2):461–464, 1978.

[SIC07] Mohamed A. Soliman, Ihab F. Ilyas, and Kevin Chen-Chuan Chang. Top-k query processing in uncertain databases. In *ICDE*, pages 896–905, 2007.

[Slo03] Noam Slonim. *The Information Bottleneck: Theory and Applications*. PhD thesis, The Hebrew University, 2003.

[TPB99] Naftali Tishby, Fernando C. Pereira, and William Bialek. The information bottleneck method. In *Proceedings of the 37-th Annual Allerton Conference on Communication, Control and Computing*, pages 368–377, 1999.

[Wid05] Jennifer Widom. Trio: A system for integrated management of data, accuracy, and lineage. In *CIDR*, pages 262–276, 2005.

[WT06] Robert H. Warren and Frank Wm. Tompa. Multi-column substring matching for database schema translation. In *Proceedings of the 32nd international conference on Very large data bases*, VLDB '06, pages 331–342. VLDB Endowment, 2006.

[YMHF01] Ling Ling Yan, Renée J. Miller, Laura M. Haas, and Ronald Fagin. Data-driven understanding and refinement of schema mappings. In *Proceedings of the 2001 ACM SIGMOD international conference on Management of data*, SIGMOD '01, pages 485–496, New York, NY, USA, 2001. ACM.

[ZGBN07] Xuan Zhou, Julien Gaugaz, Wolf-Tilo Balke, and Wolfgang Nejdl. Query relaxation using malleable schemas. In *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, SIGMOD '07, pages 545–556, New York, NY, USA, 2007. ACM.