Ziggy: Characterizing Query Results for Data Explorers

Sellam, T.; Kersten, M.

DOI
10.14778/3007263.3007287
Publication date
2016
Document Version
Final published version
Published in
Proceedings of the VLDB Endowment
License
CC BY-NC-ND

Link to publication

Citation for published version (APA):
Sellam, T., & Kersten, M. (2016). Ziggy: Characterizing Query Results for Data Explorers. *Proceedings of the VLDB Endowment*, *9*(13), 1473-1476.
https://doi.org/10.14778/3007263.3007287

Ziggy: Characterizing Query Results for Data Explorers

Thibault Sellam
CWI
the Netherlands
thibault.sellam@cwi.nl

Martin Kersten
CWI
the Netherlands
martin.kersten@cwi.nl

ABSTRACT

Data exploration has received much attention during the last few years. The aim is to learn interesting new facts from a possibly unfamiliar data set. Typically, explorers operate by trial and error: they write a query, inspect the results and refine their specifications accordingly. In this demo proposal, we present Ziggy, a system to help them understand their query results. Ziggy's aim is to complement an existing exploration system. It assumes that users already have a query in mind, but they do not know what is interesting about it. To assist them, it detects *characteristic views*, that is, small sets of columns on which the tuples in the results are different from those in the rest of the database. Thanks to these views, our explorers can understand why their selection is unique and make more informed exploration decisions.

1. INTRODUCTION

Data exploration is the act of querying a database to discover its content. Ultimately, the aim is to discover *nuggets*, i.e., interesting queries that expose unexpected phenomena. Because this task is important and omnipresent, engineers have devised *exploration systems* to support it. These systems let users "play" with their tuples. They provide facilities to write queries and visualize the results quickly. They engage their users in a loop of trial and errors, through which they discover their data. Examples of such systems are Tableau [4], based on visualizations, BlinkDB [1], based on sampling, or Blaeu [3] based on clustering.

Exploration systems assume that users can evaluate the usefulness of a set of tuples simply by looking at it - or at least, by looking at a sample. Typically, they present the results of the queries with tables, visualizations, or statistical summaries, in the hope that users will know what to inspect and where to go next. And indeed, this assumption holds with low dimension data. If a set of tuples involves less than a dozen columns, the users can plot it, build a mental picture of what it contains and judge whether they found a nugget or not. But this assumption breaks down in high dimension spaces. If the query results contain dozens or hundreds of columns, where should our users look? The resulting tuples are overwhelmingly large. Our explorers may not be able to recognize the nugget they had been wishing for even if it appeared on their screen.

One approach to solve this problem is to use dedicated multidimensional visualization methods, such as scatter-plot matrices or parallel coordinates. But these methods cannot scale to the volumes we envision. For example, a scatter plot matrix would require at least 190 distinct plots to represent a simple dataset with only 20 columns - this is hardly an improvement compared to tables.

An alternative approach is to reduce the dimensionality of the user's selection with techniques such as Principal Component Analysis. But these methods transform the data: they rescale it, project it and rotate it. Therefore, the tuples that the users visualize are not those that they requested in the first place. Furthermore dimensionality reduction techniques ignore the exploration context: they compress the user's selection, but they do not show how it compares to the rest of the database. Hence, they may miss interesting aspects of the selection.

In this demo proposal, we introduce Ziggy, a system to help explorers understand the results of their queries. Ziggy detects and plots *characteristic views*, that is, small sets of columns on which the user's tuples are different from those in the rest of the database. By consulting these views, our explorers can understand what makes their selection "special" and steer the exploration accordingly.

Let us present Ziggy with an example. An analyst wants to understand what causes violent crimes in US cities. She has access to a large table, containing 130 economic, social and demographic indicators for a few thousand communities. To seed the exploration, she selects the cities with the highest rates of criminality. Her database front-end returns a large table with more than a hundred columns. Which ones should she inspect?

Figure 1 depicts four examples of characteristic views. On all four plots, we observe that the selection has an "unusual" statistical distribution compared to the other tuples. The first plot shows that cities with a high crime index tend to have particularly high densities and large populations. In the second view, we see that these cities correspond to low levels of education. The third view reveals that dangerous neighbourhoods tend to have lower rents and a lower percentage of home ownership. The last one reveals that they are generally younger, with more mono-parental families. In effect, Ziggy identifies critical variables. Through those,

Proceedings of the VLDB Endowment, Vol. 9, No. 13
Copyright 2016 VLDB Endowment 2150-8097/16/09.

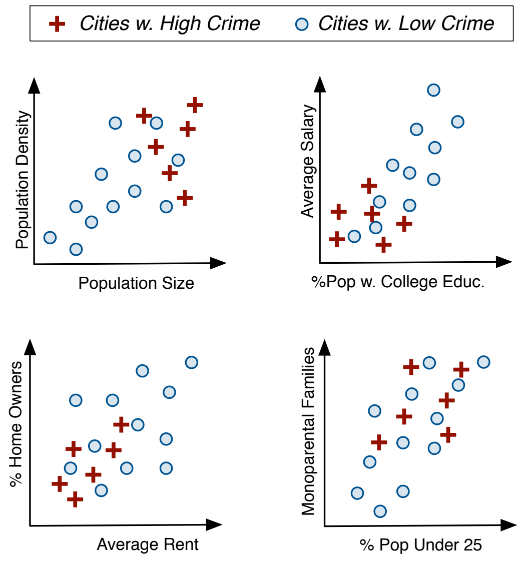


Figure 1: Four examples of characteristic views.

our analyst becomes quickly familiar with the results of her query. Ziggy's views have a purposely low dimensionality, such that our user can plot and inspect them. Furthermore, they are diverse, to show many different aspects of the data.

This demonstration will introduce the tuple characterization problem, previously described in a full research paper[1]. The visitors will discover real-life datasets and experiment with Ziggy's query description engine. Eventually, the demo will show that our prototype can extract surprising insights from datasets of all levels of complexity.

2. PROBLEM FORMULATION

We described the intuition behind characteristic views. We now formalize the problem. We present Ziggy's objective function, and we describe how it scores the views.

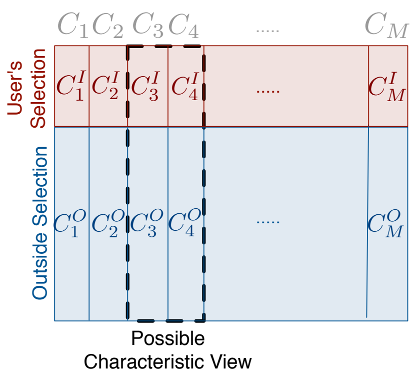
2.1 General Statement

When seeking views, Ziggy must consider three aspects. It must first find columns on which the users' selection have an "unusual" statistical distribution. Thereafter, it must enforce that its views are coherent (i.e., they describe the same aspect of the data) and that they are diverse. Let us formalize those objectives.

Let the random variables $C_1, \dots, C_M$ represent the columns in the database. We assume that the variable C_k^I represent the tuples covered by the user's query and that C_k^O represents the tuples outside the selection, as shown in Figure 2. Ziggy's aim is to find a set of at most D columns such that the distribution of $C_1^O, \dots, C_D^O$ is as different as possible from that of $C_1^I, \dots, C_D^I$. If $\mathfrak{D}$ represents a measure of distribution divergence from the statistics literature, our aim is to find the views $\mathcal{V}_i = \{C_1, \dots, C_D\}$ that maximize the following quantity:

$$\textsc{score}(\mathcal{V}_i) = \mathfrak{D}\big(C_1^O, \dots, C_D^O \,;\, C_1^I, \dots, C_D^I\big) \tag{1}$$

[1]At the time of writing, the paper was accepted for publication at SSDBM 2016, under the name *Fast, Explainable View Detection to Characterize Exploration Queries.*



$C_1 C_2 C_3 C_4$ ..... C_M
User's Selection
C_1^I C_2^I C_3^I C_4^I ..... C_M^I
Outside Selection
C_1^O C_2^O C_3^O C_4^O ..... C_M^O
Possible Characteristic View

Figure 2: Our problem setting.

Common examples of divergence functions $\mathfrak{D}$ are the distance between the centroids and the Kullback-Leibler divergence [5]. We will present our own in the next subsection.

Equation 1 is not complete because it favors large, heterogeneous subspaces. For many functions $\mathfrak{D}$, $\textsc{score}(\mathcal{V}_i)$ reaches its maximal when $\mathcal{V}_i$ contains all the columns in the database. Furthermore, maximizing $\textsc{score}(\mathcal{V}_i)$ alone does not guarantee that the variables are thematically related. To obtain smaller, more homogeneous subspaces, we introduce a constraint. Let $\mathfrak{S}$ describe a measure of statistical dependency, such as the correlation or the mutual information [5]. The *tightness* of a view measures how interdependent its variables are:

$$\textsc{tightness}(\mathcal{V}_i) = \min_{C_k, C_l \in \mathcal{V}_i} \mathfrak{S}(C_k, C_l) \tag{2}$$

We introduce a constraint MIN_{tight} on this quantity:

$$\textsc{tightness}(\mathcal{V}_i) \geq MIN_{tight} \tag{3}$$

Another shortcoming of the objective function exposed in Equation 1 is that it leads to redundancy. Typically, the results will contain every possible subsets of a few dominant variables. To keep the output short and diverse, we enforce that the view are disjoint. Assume that we have already discovered $i-1$ views $\mathcal{V}_1, \dots, \mathcal{V}_{i-1}$, and that we are seeking a new view $\mathcal{V}_i$. Let the notation $\mathcal{V}_{1\dots i-1}$ describe the union $\mathcal{V}_{1\dots i-1} = \bigcup_{v\in[1,i-1]} \mathcal{V}_v$. We introduce the following constraint:

$$\textsc{overlap}(\mathcal{V}_{1\dots i-1}, \mathcal{V}_i) = |\mathcal{V}_{1\dots i-1} \cap \mathcal{V}_i| = 0 \tag{4}$$

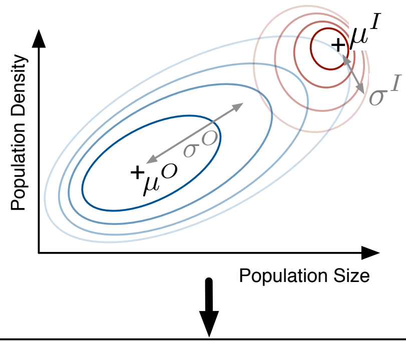
We can now present our full problem.

Problem *Assume that we have already discovered $i-1$ views. If MIN_{tight} describes a user-defined parameter, our aim is to find a view $\mathcal{V}_i$ that solves the following system:*

$$\begin{aligned} &Argmax_{\mathcal{V}_i}\ \textsc{score}(\mathcal{V}_i) \\ &\quad s.t.\ \textsc{tightness}(\mathcal{V}_i) \geq MIN_{tight} \\ &\qquad\ \ \textsc{overlap}(\mathcal{V}_{1\dots i-1}, \mathcal{V}_i) = 0 \end{aligned} \tag{5}$$

2.2 Dissimilarity Measure

The statistics literature presents many options to instantiate the statistical divergence measure $\mathfrak{D}$ [5]. But most of these operate in a "black box" fashion: they indicate how much two distributions differ, but they do not explain why. We now introduce our own function, the *Zig-Dissimilarity*, that overcomes this problem.



Zig-Components

Difference between the means	$\frac{\mu^I - \mu^O}{\sigma^O}$
Difference between the std. deviations	$\frac{\sigma^I - \sigma^O}{\sigma^O}$
Difference betw. the correlation coefficients	$r^I - r^O$

Figure 3: Examples of Zig-Components.

The idea behind the Zig-Dissimilarity is to compute several simple indicators of dissimilarity, the *Zig-Components*, and aggregate them into one synthetic score. Figure 3 presents three such indicators: the difference between the means, the difference between the standard deviations and the difference between the correlation coefficients. We see that each Zig-Component highlights one particular aspect of the difference between the distributions. Also, these functions are verifiable: the users can inspect the charts and check whether they hold. Most of our Zig-Components come from the statistics literature, where they are referred to as *effect sizes* [2]. Observe that we test dissimilarities in spaces with one but also two dimensions. For instance, the difference between the correlation coefficients involves two columns. In principle, we could design Zig-Components for higher dimensionalities. Nevertheless, those only add marginal accuracy gains in practice, at the cost of significant processing times. We refer the interested reader to our full paper for other examples of Zig-Components (e.g., involving categorical data).

To aggregate the Zig-Components, we normalize them and compute a weighted sum. The normalization enforces that the indicators have comparable scale. The weights in the final sum are defined by the user. Thanks to this mechanism, our explorers can express their preference for one type of difference over the others.

The advantage of our divergence measure is that it lets Ziggy *explain* its choices. To illustrate, let us return to the first view of Figure 1. A classic subspace search algorithm would present the chart without further explanations. In contrast, our system can motivate its decisions. In this case, it comments the view as follows:

> "On the columns `Population` and `Density`, your selection has particularly high values and a low variance"

This short sentence describes why Ziggy chose the columns `Population` and `Density`. The users can interpret these explanations as hints for further exploration, to make sure that they have not missed any aspect of their query results.

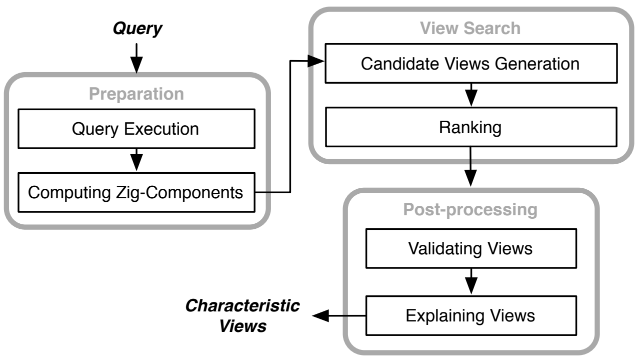


Figure 4: Ziggy's Tuples Description Pipeline.

3. ZIGGY'S ARCHITECTURE

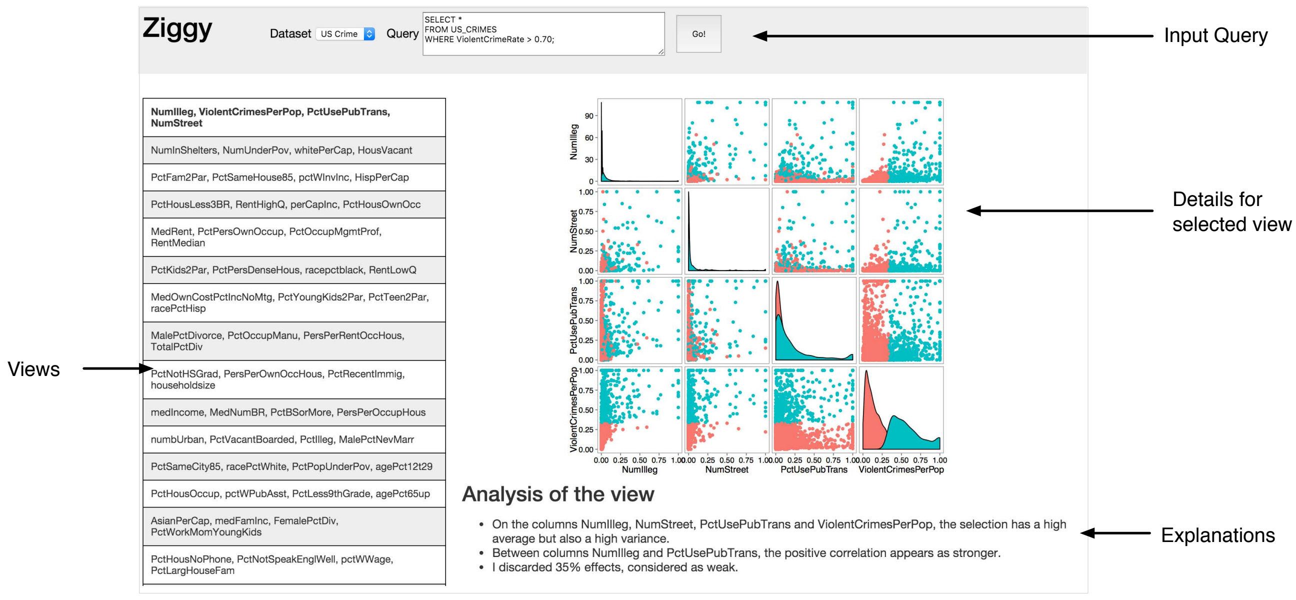
We now present how Ziggy solves the tuple characterization problem. Figure 4 presents Ziggy's tuples description pipeline. It includes three stages: preparation, view search and post-processing. During the preparation stage, Ziggy collects the statistics necessary to build the views. In the view search stage, it effectively forms the views. During the last step, it checks if those views are statistically robust and it generates the explanations.

Preparation. During the preparation step, Ziggy executes the user's query, loads the results, and computes the Zig-Components associated to each column and each couple of columns. This is often the most time consuming step. In our full paper, we present a strategy to share computations between queries, and therefore reduce the amount of data to read. The output of these operations is a table, which describes the Zig-Components associated to each variable and each pair of variables.

View Search. During this step, Ziggy builds the views. It operates as follows. First, it enumerates the groups of columns which satisfy the constraints of Equation 5. It does so with a graph-based algorithm: it materializes the graph formed by the column's pairwise dependencies, and partitions it with a clique search or clustering algorithm. In our implementation, we used complete linkage clustering [5]. This method is simple, well established, and it provides a dendrogram, i.e., visual support to help setting the parameter MAX_{tight}. From this step, Ziggy obtains a set of candidate views. It scores them using the Zig-Components obtained previously, and it ranks the set accordingly.

Post-Processing. During the final phase, Ziggy evaluates the statistical robustness of the views. The aim is to control spurious findings, that is, differences caused by chance. For each view, it tests the significance of the Zig-Component separately, using asymptotic bounds from the literature [2]. Then it aggregates the confidence scores associated with each component. Depending on the users' preferences, it retains the lowest value, or it uses more advanced aggregation schemes such as the Bonferroni correction [5].

During the last step, Ziggy also generates the explanations. Given the composite nature of the Zig-Dissimilarity, this step is straightforward: Ziggy choses the Zig-Components associated with the highest levels of confidence, and it describes them with text. We implemented the text generation functionalities with handwritten rules and regular expressions.



```
SELECT *
FROM US_CRIMES
WHERE ViolentCrimeRate > 0.70;
```

NumIlleg, ViolentCrimesPerPop, PctUsePubTrans, NumStreet

NumInShelters, NumUnderPov, whitePerCap, HousVacant

PctFam2Par, PctSameHouse85, pctWInvInc, HispPerCap

PctHousLess3BR, RentHighQ, perCapInc, PctHousOwnOcc

MedRent, PctPersOwnOccup, PctOccupMgmtProf, RentMedian

PctKids2Par, PctPersDenseHous, racepctblack, RentLowQ

MedOwnCostPctIncNoMtg, PctYoungKids2Par, PctTeen2Par, racePctHisp

MalePctDivorce, PctOccupManu, PersPerRentOccHous, TotalPctDiv

PctNotHSGrad, PersPerOwnOccHous, PctRecentImmig, householdsize

medIncome, MedNumBR, PctBSorMore, PersPerOccupHous

numbUrban, PctVacantBoarded, PctIlleg, MalePctNevMarr

PctSameCity85, racePctWhite, PctPopUnderPov, agePct12t29

PctHousOccup, pctWPubAsst, PctLess9thGrade, agePct65up

AsianPerCap, medFamInc, FemalePctDiv, PctWorkMomYoungKids

PctHousNoPhone, PctNotSpeakEnglWell, pctWWage, PctLargHouseFam

Analysis of the view

- On the columns NumIlleg, NumStreet, PctUsePubTrans and ViolentCrimesPerPop, the selection has a high average but also a high variance.
- Between columns NumIlleg and PctUsePubTrans, the positive correlation appears as stronger.
- I discarded 35% effects, considered as weak.

Figure 5: Snapshot of Ziggy's interface.

4. IMPLEMENTATION AND DEMO

4.1 Overview of the System

Our demonstration system comprises three components. In the bottom layer, a DBMS stores and delivers the data (we chose MonetDB). The middle layer comprises the query characterization engine and a Web server. We developed both components in R, except for a few critical performance operations written in C (those related to computing Zig-Components). The Web server relies on the package `Shiny`. The front-end is based on HTML and Javascript.

Figure 5 presents a snapshot of our demonstration system. Users specify their queries in the text box of the top panel. Ziggy returns the views on the left side, ranked by decreasing order of dissimilarity. It displays the explanations on the right side.

4.2 Use Cases

We will demonstrate Ziggy with three real-life datasets.

- The **Box Office** dataset describes Hollywood movies released between 2007 and 2013. We will use it to introduce the main concepts behind Ziggy: the query description problem, how Ziggy choses views and how to read them. The data contains 900 tuples and 12 columns.
- The **US Crime** database contains 128 crime and socio-economic indicators for 1994 US Cities. The dataset is freely available on the UCI Repository[2]. The use case is similar to the running example used throughout this paper. We hope to surprise our visitors by showing that seemingly superfluous variables can have a strong predictive power - such as the number of boarded windows in a given neighborhood.
- The **Countries and Innovation** dataset describes innovation and patents for different regions of the world. We obtained it by combining different tables from the Website of the OECD, an international economic organization[3]. It contains 6,823 rows and 519 columns. We will show that Ziggy can highlight complex phenomena, in effect generating hypotheses for future exploration.

During our presentation, we will use ready-made queries and encourage the visitors to suggest their own.

5. CONCLUSION

During the last few years, authors have introduced dozens of methods to discover new, interesting queries. In this paper, we tackled the complementary problem: once our users have a query, how do they know if it is a good one? Our short term objective is to demonstrate Ziggy and let visitors challenge our system. On the long term, we intend to distribute our tuple description engine as a library, to be included into external exploration systems.

6. ACKNOWLEDGMENTS

This work was supported by the Dutch national program COMMIT.

7. REFERENCES

[1] S. Agarwal, A. P. Iyer, A. Panda, S. Madden, B. Mozafari, and I. Stoica. Blink and it's done: interactive queries on very large data. *PVLDB*, 5(12):1902–1905, 2012.
[2] L. V. Hedges and I. Olkin. *Statistical method for meta-analysis*. Academic press, 1985.
[3] T. Sellam and M. Kersten. Cluster-driven navigation of the query space. *IEEE TKDE*, 28(5):1118 – 1131, 2016.
[4] C. Stolte, D. Tang, and P. Hanrahan. Polaris: a system for query, analysis, and visualization of multidimensional relational databases. *TVCG*, 2002.
[5] L. Wasserman. *All of statistics: a concise course in statistical inference*. Springer, 2013.

[2]`https://archive.ics.uci.edu/ml/datasets/Communities+and+Crime`

[3]`http://stats.oecd.org/`