Fast, Explainable View Detection to Characterize Exploration Queries

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

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

ABSTRACT

The aim of data exploration is to get acquainted with an unfamiliar database. Typically, explorers operate by trial and error: they submit a query, study the result, and refine their query subsequently. In this paper, we investigate how to help them understand their query results. In particular, we focus on medium to high dimension spaces: if the database contains dozens or hundreds of columns, which variables should they inspect? We propose to detect subspaces in which the users' selection is different from the rest of the database. From this idea, we built Ziggy, a tuple description engine. Ziggy can detect informative subspaces, and it can explain why it recommends them, with visualizations and natural language. It can cope with mixed data, missing values, and it penalizes redundancy. Our experiments reveal that it is up to an order of magnitude faster than state-of-the-art feature selection algorithms, at minimal accuracy costs.

CCS Concepts

•**Information systems** → **Data mining;** *Data analytics;* •**Human-centered computing** → *Visual analytics;* •**Computing methodologies** → *Feature selection;*

Keywords

Data exploration, subspace search, data description

1. INTRODUCTION

Data exploration has gained lots of attention over the last few years. The main challenge is to support users with little prior knowledge, and no clear objective. Some explorers need a preliminary impression of the data before engaging into more structured tasks. Others are browsing in the hope to discover new insights. Several authors have proposed tools to help data exploration [1, 5, 11, 17]. These systems offer some textual or graphical interface, through which users can create, visualize and modify selections of tuples quickly. Thus, explorers are engaged in a tight trial-and-error loop, through which they discover their datasets.

Data exploration systems rely on a crucial assumption: they suppose that if users see an interesting set of tuples (e.g., through tables or visualizations), they will recognize it immediately, and think "aha, this is interesting". This assumption may hold with small datasets, but it collapses in higher dimensions. If the dataset contains dozens, or hundreds of columns, then which variable should the users inspect? Harder still, which *combination* of variables should they inspect? We cannot assume that our data explorers know where to look. Yet, studying each possibility in turn can turn out to be a slow, tedious process. This beats the purpose of a data exploration system. **How can we efficiently choose columns to characterize our explorers' tuples?**

One way to solve this problem is *subspace search* [2, 14], which detects sets of columns on which the data has an "interesting" distribution (e.g., exploiting correlations, clusters and outliers). Also, we could use *dimensionality reduction* techniques such as PCA. However, these algorithms are unsupervised: they completely ignore the user's input, hence they may fail to capture its characteristics. Another approach is to map our problem to a *feature selection* setting [8]: we materialize the user's selection into a vector of binary variables to be predicted, and we detect which combination of variables from the database has the best predictive power. But feature selection algorithms focus on accuracy, not on exploration. They often neglect speed, and they are totally oblivious to interpretation and variety.

In this paper we introduce our approach, *multi-view subset characterization*. The main idea is to show how the selected tuples differ from the rest of the database. To do so, we identify several sets of columns for which the statistical distribution of the tuples is "unusual". We implement this idea in Ziggy, our subset characterization engine. Experiments with classifiers show that Ziggy generates *informative* views. But it is also very fast: thanks to aggressive optimizations, it can process 10,000s tuples on dozens of variables within a second - an order of magnitude faster than state-of-the-art algorithms, at minimal accuracy costs. More importantly, Ziggy is a white box: every choice it makes is fully *explainable*. As a proof of concept, we will present a method to convert its findings to natural language, i.e., *plain English*. Few data mining algorithms provide this functionality, and none of those that do tackle view search.

We summarize our contributions as follows:

SSDBM '16, July 18 - 20, 2016, Budapest, Hungary

ISBN 978-1-4503-4215-5/16/07…$15.00

DOI: http://dx.doi.org/10.1145/2949689.2949692

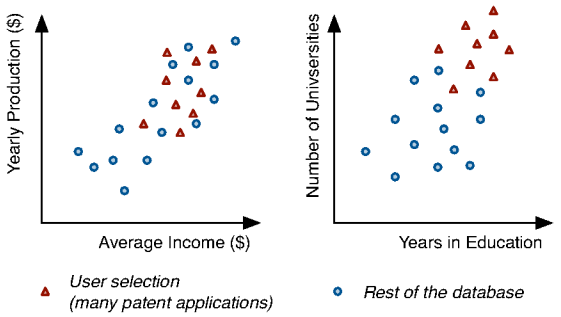


Figure 1: Two examples of subspaces for which the user's selection has an unusual distribution (using fictive data).

- We formalize the multi-view characterization problem.
- We describe Ziggy, a fast, transparent, and robust system to detect characteristic views of tuples.
- We present methods to validate Ziggy's findings and describe them with charts and natural language.
- We apply Ziggy to real-life situations and benchmark it against state-of-the-art algorithms.

The rest of this paper is organized as follows. In Sections 2 and 3, we present our general problem. We instantiate this problem in Section 4 and describe our algorithms in Section 5. We discuss how to validate our results and report them in Sections 6 and 7. We describe how to set parameters in Section 8. In Sections 9 and 10, we apply our solution to real and synthetic data. Finally, we present related work and conclude in Sections 11 and 12.

2. OVERVIEW

Let us introduce our problem through an example. A government analyst tries to understand which demographic, social and economic factors lead to innovation. More specifically, she is interested in patents: in which parts of the world do individuals submit patents? What lessons can she learn? She collects several hundred regional statistics, and loads them in her favorite Business Intelligence tool (possibly based on SQL or spreadsheets). She selects the top 10% regions for which many patents are submitted, and ends up with an overwhelmingly large table, comprising a few dozen tuples and hundreds of columns. How can we help her?

Our idea to detect sets of columns for which our analyst's tuples are "special", that is, have an unusual distribution compared to the rest of the database. A fundamental assumption behind our work is that the user is interested in the differences between her selection and the remaining tuples, not the similarities. This is a strong hypothesis, but recent work has shown that it covers a wide range of use cases [19].

Figure 1 depicts two of those sets: `{Yearly Production, Average Income}`, and `{Years in Education, Number of Universities}`. We refer to them as *views*, or *subspaces*. We see that on all the variables, the selection has unusually high values. This shows that regions with many patent applications tend to be richer and have solid education systems. Observe that these views have low dimensionalities: each of them illustrates one "characteristic" of the selection. In fact, our aim is not to show *all* the variables on which differences appear. Instead, we seek a few small, non redundant subspaces. We call this approach *multi-view characte-rization*.

To visualize the subspaces, our analyst may use tables or charts, as in Figure 1. But, she may also wonder *why* the system chose these views. For instance, she may wish to check the sanity of the results, or she may seek guidance. To help her, we provide justifications in natural language. Here is an example:

> "On the columns `Average Income` and `Yearly Production`, your selection has a high average and low variance. The effect is similar and stronger on `Years in Education` and `Number of Universities`."

We also couple these descriptions with visualizations, for further inspection.

3. GENERAL PROBLEM STATEMENT

Let us now formalize the multi-view characterization problem. We represent our database with a matrix $\mathbf{D}$, with M columns and N rows. We model each tuples by an iid. random vector $\mathbf{x}_n = (x_n^1, \ldots, x_n^M)^\top$, and each column by $\mathbf{x}^m = (x_1^m, \ldots, x_N^m)^\top$.

Let the submatrix $\mathbf{V} = [\mathbf{x}^1, \ldots, \mathbf{x}^V]$ describe a view (i.e., subspace, or set of columns). We split this view in two: $\mathbf{V}_t$ contains the tuples in the selection, $\mathbf{V}_{\bar{t}}$ the remainder, as shown below:

$$\begin{array}{c} \\ \mathbf{x}_1 \\ \mathbf{x}_2 \\ \\ \\ \mathbf{x}_N \end{array} \begin{array}{c} \begin{array}{cc} \mathbf{x}^1 & \mathbf{x}^2 \end{array} \\ \begin{bmatrix} x_1^1 & x_1^2 & \cdots \\ x_2^1 & x_2^2 & \\ \vdots & & \ddots \\ & & \\ & & \end{bmatrix} \end{array} \begin{array}{c} \\ \\ \mathbf{V}_t \\ \\ \mathbf{V}_{\bar{t}} \\ \\ \end{array} \begin{array}{c} \mathbf{x}^M \\ \\ \\ \\ \\ x_N^M \end{array}$$

To measure how much $\mathbf{V}_t$ and $\mathbf{V}_{\bar{t}}$ differ, we compare the empirical probability distribution of their rows. If these distributions differ, then the view exhibits some "peculiarity" of the data. Therefore, $\mathbf{V}$ is likely to be informative. We measure this difference with a **mass dissimilarity measure** $\mathfrak{D}(\mathbf{V}_t, \mathbf{V}_{\bar{t}})$. This function returns 0 if the tuples come from the same distribution, it grows otherwise. The statistics literature contains several candidates, for example estimators the Kullback-Leibler Divergence [21]. We will present our own function in the following section.

We can already propose a first, naive, formulation of our problem:

PROBLEM 1. *Consider a distribution dissimilarity function $\mathfrak{D}$ and two integers K and D. Find the top K distinct views $\mathbf{V}^i$ with at most D dimensions which maximize $\mathfrak{D}(\mathbf{V}_t^i; \mathbf{V}_{\bar{t}}^i)$ (ignoring column permutations).*

This approach is simple, but it yields redundancy: a small number of good columns may dominate the results and appear in all K views. In a data exploration scenario, users may value *diversity*, even if the views are sub-optimal. To enforce this requirement, we introduce a penalty factor in the objective function.

Property	Type 1	Type 2	Zig-Component	Comments
Mean	Contin.	-	$\mathfrak{z}_m^{\mathbf{d}} = (\bar{\mathbf{d}}_t - \bar{\mathbf{d}}_{\bar{t}})/s_{\bar{t}}$	Known as Glass' Δ [4]
Stand. Dev.	Contin.	-	$\mathfrak{z}_\sigma^{\mathbf{d}} = (s_t - s_{\bar{t}})/s_{\bar{t}}$	
Frequencies	Discrete	-	$\mathfrak{z}_\chi^{\mathbf{d}} = \sqrt{\chi^2}$	Based on Pearson's χ^2 goodness-of-fit test [21]
Dependence	Contin.	Contin.	$\mathfrak{z}_r^{\mathbf{d},\mathbf{d}'} = r_t - r_{\bar{t}}$	$r_t$is the correlation coefficient between $\mathbf{d_t}$ and $\mathbf{d'_t}$ $r_{\bar{t}}$ is the correlation coefficient between $\mathbf{d_{\bar{t}}}$ and $\mathbf{d'_{\bar{t}}}$ [21]
Dependence	Discrete	Both	$\mathfrak{z}_V^{\mathbf{d},\mathbf{d}'} = V_t - V_{\bar{t}}$	V_t is Cramér's V coefficient between $\mathbf{d_t}$ and $\mathbf{d'_t}$ $V_{\bar{t}}$ is Cramér's V coefficient between $\mathbf{d_{\bar{t}}}$ and $\mathbf{d'_{\bar{t}}}$ [4]

Table 1: Our choice of Zig-Components for different data types. Each component describes a difference, to be evaluated for either each column $\mathbf{d}$ or each couple of columns $\mathbf{d}, \mathbf{d}'$ in the view. The notations $\bar{\mathbf{d}}$ and s respectively represent the sample mean and sample standard deviation. In the mixed-types case, we discretize the continuous column with equi-width binning.

Measuring redundancy between columns is not straightforward, as this notion is partially subjective. In our model, we exploit statistical dependency: if two sets of columns are tightly correlated, then there is a high chance that they describe the same property of the "real world". Oppositely, if they are independent, then they probably convey different types of information. From this observation, we introduce a new version of our problem: we seek views which maximize the dissimilarity measure, while minimizing inter-view dependency. We define this problem in a recursive way:

PROBLEM 2. *Suppose that we have already detected $i-1$ views ($i > 1$, $\mathbf{V}^0 = \emptyset$). We obtain $\mathbf{V}^{1..i-1} = [\mathbf{V}^1, \ldots, \mathbf{V}^{i-1}]$ by concatenating these views. Let $\mathfrak{S}$ describe a statistical dependency measure. Given a positive real λ, find the view $\mathbf{V}^i$ with at most D columns which maximizes:*

$$\mathfrak{D}\big(\mathbf{V}_t^i; \mathbf{V}_{\bar{t}}^i\big) - \lambda \cdot \mathfrak{S}(\mathbf{V}^i; \mathbf{V}^{1..i-1}) \tag{1}$$

Statistics textbooks offer many options to instantiate the dependency measure $\mathfrak{S}$. Well established examples are multivariate variants of the correlation coefficient, or the mutual information [21]. Here again, we will present our own function in Section 4.2.

In Equation 1, the parameter λ controls the trade-off between dissimilarity and diversity: a high value enforces that the view are diverse, while a low value expresses our preference for maximizing $\mathfrak{D}(\mathbf{V}_t^i; \mathbf{V}_{\bar{t}}^i)$. In practice, this parameter is not convenient because it has no intuitive scale. For some L, an equivalent way to express our problem is the following:

$$\begin{aligned} &\text{Argmax}_{\mathbf{V}^i} && \mathfrak{D}\big(\mathbf{V}_t^i; \mathbf{V}_{\bar{t}}^i\big) \\ &\text{s.t.} && \mathfrak{S}(\mathbf{V}^i; \mathbf{V}^{1..i-1}) < L \end{aligned} \tag{2}$$

Equation 1 in the Lagrangian of Equation 2, up to a negligible additive constant. We prefer this form because the trade-off parameter L has the same scale as $\mathfrak{S}(\mathbf{V}^i; \mathbf{V}^{1..i-1})$. For example, if we instantiate $\mathfrak{S}$ with a measure of correlation, then L will simply describe the maximal acceptable correlation between $\mathbf{V}^i$ and $\mathbf{V}^{1..i-1}$.

4. INSTANTIATION: MEET ZIGGY

We now instantiate the dissimilarity measure $\mathfrak{D}$ and redundancy measure $\mathfrak{S}$.

4.1 Explainable Mass Dissimilarity

Let us begin with the dissimilarity $\mathfrak{D}$. As mentioned previously, we could borrow any general divergence measure from the statistics literature, such as the KL-divergence. However, these measurements operate as "black boxes": they describe the intensity of the differences, but they do not explain how the tuples differ. Our approach is diametrically opposed: we introduce the **Zig-Dissimilarity** $\mathfrak{D}_{Zig}$, a completely *explainable* mass dissimilarity measure.

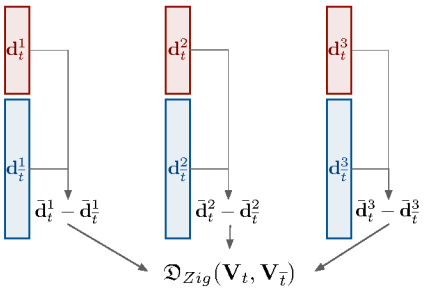


Figure 2: Illustration of Ziggy's Dissimilarity.

The idea behind our dissimilarity measure is to compute several simple, interpretable indicators of difference, called **Zig-Components**, and aggregate them in a single composite score, the Zig-Dissimilarity. Consider for instance two sets of tuples $\mathbf{V}_t$ and $\mathbf{V}_{\bar{t}}$ with V columns. If these subsets have the same average for every column, then they probably come from the same distribution. Oppositely, if the averages are different, then they probably come from different distributions. From this observation, we can build an elementary dissimilarity measure: for each column $\mathbf{d}$, we compute the differences between the means $\bar{\mathbf{d}}_t - \bar{\mathbf{d}}_{\bar{t}}$, as shown in Figure 2. These are our Zig-Components. We then aggregate these scores, by averaging their absolute values: we obtain the Zig-Dissimilarity.

Let us now formalize these ideas. We define Zig-Components as follows:

DEFINITION 1. *A Zig-Component is a function $\mathfrak{z} : \mathbf{V} \times \mathbf{V} \to \mathbb{R}$ which describes one difference between two sets of tuples. If the tuples are similar, then $\mathfrak{z}(\mathbf{V}_t, \mathbf{V}_{\bar{t}}) = 0$. If not, the magnitude of $\mathfrak{z}(\mathbf{V}_t, \mathbf{V}_{\bar{t}})$ varies with the strength of the difference.*

For the sake of presentation, we will abbreviate $\mathfrak{z}(\mathbf{V}_t, \mathbf{V}_{\bar{t}})$ as $\mathfrak{z}$. Once we computed the Zig-Components $\mathfrak{z}_1, \ldots, \mathfrak{z}_Z$, we obtain the Zig-Dissimilarity as follows. First, we compute the absolute values $|\mathfrak{z}_1|, \ldots, |\mathfrak{z}_Z|$ for each component. We then normalize the values across components of same type. We obtain a set of intermediary scores $z_1, \ldots, z_Z$. We aggregate these scores with a weighted sum. These operations give us one scalar indicator, which summarizes the magnitude of all the differences.

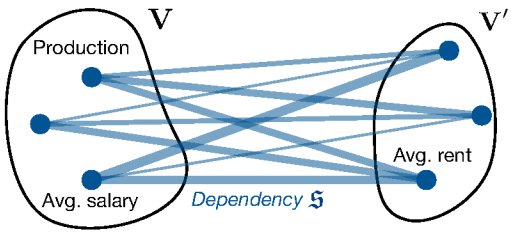


Figure 3: Illustration of Ziggy's redundancy measure. Vertices represent columns, edges represent pairwise dependencies.

Definition 2. *Let $z_1, \ldots, z_Z$ represent a set of normalized absolute values of zig-components, and $w_1, \ldots, w_Z$ a set of weights. We define the Zig-Dissimilarity as follows:*

$$\mathfrak{D}_{Zig}(\mathbf{V}_t, \mathbf{V}_{\bar{t}}) \equiv \sum_{k \in [1,Z]} w_k \cdot z_k(\mathbf{V_t}, \mathbf{V}_{\bar{t}}) \tag{3}$$

In our implementation, we combined five types of Zig-Components, summarized in Table 1. A few of them come from the statistics literature, where they are referred to as *effect sizes* [4]. We chose these indicators because they have intuitive interpretations, as well as known asymptotic properties (we will use those Section 6). A majority, but not all of them are normalized and vary between -1 and 1. None of these functions are symmetric, and most of them change sign according to the direction of the underlying effect.

Observe that Ziggy computes univariate, but also bivariate components: in this case it compares columns by pairs. Our motivation is to check which correlations are present in $\mathbf{V_t}$ and not in $\mathbf{V}_{\bar{t}}$, or more generally which correlations are either strengthened or weakened across the subsets. In the discrete case, we substitute the correlation coefficient by Cramér' s V. This function also measures the associations between two variables, and it varies similarly between 0 and 1 [4]. Its value is $\sqrt{\chi^2/(N(\min(|\mathbf{d_t}|, |\mathbf{d_{\bar{t}}}|) - \mathbf{1}))}$, where χ^2 is Pearson's χ^2 statistic, and $|\mathbf{d_t}|, |\mathbf{d_{\bar{t}}}|$ are the number of distinct values in $\mathbf{d_t}$ and $\mathbf{d_{\bar{t}}}$ respectively.

In principle, we could test differences in spaces with more than two dimensions. We chose not to do so, for two reasons. First, the number of relationships to be inspected grows exponentially with the dimensionality of the Zig-Component. This hurts Ziggy's runtime, and leads to much longer outputs. Second, relationships in three dimensions or more are harder to convey and understand. We will show in Section 10 that this restriction has surprisingly little effect on Ziggy's accuracy in practice.

Finally, the aim of the weights w_k is to balance the effects across column types. For instance, we measure two components for the numerical columns, and only one for categorical data. Thus, we set $w_k = 1/2$ for the former and $w_k = 1$ for the latter. These parameters also let users express their preferences: for example, a novice user may value one-dimension Zig-Components over those based on correlation.

4.2 Dependency Measure

We now present two instantiations for the measure of redundancy, $\mathfrak{S}_{hard}$ and $\mathfrak{S}_{soft}$. Both measures are a variant of the same principle, and we will refer to them collectively as the Zig-Dissimilarity $\mathfrak{S}_{Zig}$. Here again, our motivation is to aggregate several simple interpretable indicators.

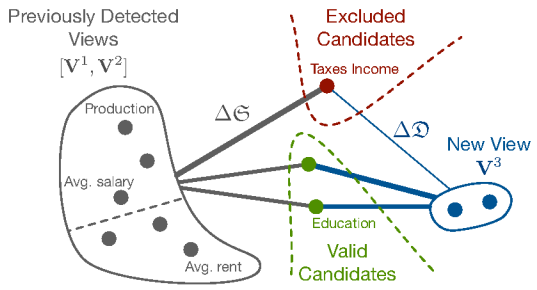


Figure 4: Ziggy's greedy view composition algorithm, with $i = 3$ and $D = 3$. The grey edges represent the amount of redundancy added by each candidate. The blue edges represent the amount of dissimilarity added by each candidate.

Let $\mathbf{V}$ and $\mathbf{V}'$ represent two views. We compute the Zig-Dependency in two steps. First, we compute the pairwise dependency $\mathfrak{s}(\mathbf{d}, \mathbf{d}')$ between every column $\mathbf{d}$ of $\mathbf{V}$ and every column $\mathbf{d}'$ of $\mathbf{V}'$, as shown in Figure 3. The measure $\mathfrak{s}$ is a convenience function, which allows us to combine different types of variables:

$$\mathfrak{s}(\mathbf{d}, \mathbf{d}') = \begin{cases} r(\mathbf{d}, \mathbf{d}') \text{ if } \mathbf{d}, \mathbf{d}' \text{ are continuous (correlation)} \\ V(\mathbf{d}, \mathbf{d}') \text{ otherwise (Cramér's V, cf. Sec. 4.1)} \end{cases} \tag{4}$$

During the second step, we aggregate these dependencies. The function $\mathfrak{S}_{hard}$ utilizes the maximum, while $\mathfrak{S}_{soft}$ uses the mean:

$$\mathfrak{S}_{hard}(\mathbf{V}, \mathbf{V}') = \max_{d \in \mathbf{V}, d' \in \mathbf{V}'} |\mathfrak{s}(d, d')| \tag{5}$$

$$\mathfrak{S}_{soft}(\mathbf{V}, \mathbf{V}') = \frac{\sum_{d \in \mathbf{V}, d' \in \mathbf{V}'} |\mathfrak{s}(d, d')|}{V \cdot V'} \tag{6}$$

Both functions $\mathfrak{S}_{soft}$ and $\mathfrak{S}_{hard}$ vary between 0 and 1, 0 indicating no dependency. The crucial difference lies in how they treat overlap. If one column is present in both $\mathbf{V}$ and $\mathbf{V}'$, then $\mathfrak{S}_{hard}$ is at its maximal value 1, regardless of the other columns in the views. It is not necessarily so for $\mathfrak{S}_{soft}$. Thus $\mathfrak{S}_{hard}$ leads to less redundancy, while $\mathfrak{S}_{soft}$ is more flexible. We will demonstrate these effect in our Experiments section.

Finally, observe that our choice for $\mathfrak{S}$ is computationally efficient: we get the pairwise correlations "for free", because we need to compute them anyway to obtain the Zig-Components $\mathfrak{z}_\chi$ and $\mathfrak{z}_r$.

5. ALGORITHMS TO DETECT VIEWS

We introduced a mass dissimilarity measure $\mathfrak{D}_{Zig}$ and a view dependency measure $\mathfrak{S}_{Zig}$. We now discuss how to maximize the former while constraining the latter, as exposed in Equation 2.

5.1 Base algorithm

Once the user has provided a selection of tuples, Ziggy performs two steps. First, it computes the Zig-Components, for each column and each pair of columns in the database. Then, it composes the views greedily: it successively picks the columns associated with the highest components, while ensuring that the dependency threshold L is not breached.

Algorithm 1 View Construction

```
function DetectViews(K, D, L)
    for k ∈ [1, K] do
        Cand ← {x^1, ..., x^M}
        Red.Gains ← {ΔG(d, V^{1..k}), for d ∈ Cand}
        Cand ← {d where not Red.Gains(d) > L}
        Dis.Gains ← {ΔD(d, d'), for d, d' ∈ Cand}
        V^k ← V^k ∪ {argmax_{d,d'} Dis.Gains}
        for d ∈ [3, D] do
            Cand ← {x^1, ..., x^M} \ V^k
            Red.Gains ← {ΔG(d, V^{1..k}), for d ∈ Cand}
            Cand ← {d where not Red.Gains(d) > L}
            Dis.Gains ← {ΔD(d, V^k), for d ∈ Cand}
            V^k ← V^k ∪ {argmax_d Dis.Gains}
        end for
    end for
end function
```

The first step is straightforward, but it is also by far the most time consuming. We discuss in Section 5.2 how to optimize it. During the second step Ziggy creates the views by adding columns in a best-first fashion. Figure 4 illustrates this approach. Suppose that Ziggy has previously obtained two views, $\mathbf{V}^1$ and $\mathbf{V}^2$, and it is currently building a third one, $\mathbf{V}^3$. For each column in $\mathbf{V} \setminus \mathbf{V}^3$, our algorithm computes two scores: the gain of dissimilarity $\Delta\mathfrak{D}$ induced by adding the candidate to $\mathbf{V}^3$, and the gain of redundancy $\Delta\mathfrak{G}$ induced by the same. Ziggy excludes the columns which exceed the redundancy capacity, e.g., those for which $\mathfrak{G}([\mathbf{V}^1, \mathbf{V}^2], \mathbf{V}^3) + \Delta\mathfrak{G} > L$. It then detects the best candidate among those that remain and adds it to the view. It repeats the process until either the maximal number of columns D is met, or there are no more eligible columns. We present the pseudo-code in Algorithm 1.

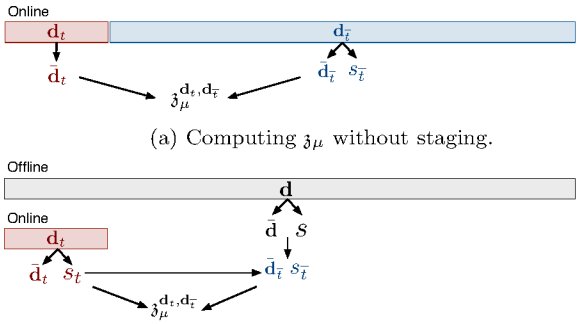
To compute $\Delta\mathfrak{G}$, we apply Equations 5 and 6 directly. Computing $\Delta\mathfrak{D}$ requires more care, because each column is involved in several Zig-Components, and the bivariate components depend on the current state of the algorithm. Suppose for instance that we are building a view $\mathbf{V}^i$, and we wish to compute $\Delta\mathfrak{D}$ for a given candidate $\mathbf{d}$. In our implementation, if $\mathbf{d}$ contains numeric values, then it is associated with at least two components: the difference between the means $\mathfrak{z}_\mu$, and the difference between the standard deviations $\mathfrak{z}_\sigma$. We obtain those from the first step of the algorithm. But we must also account for the difference between dependencies, for all the columns already included in $\mathbf{V}^i$. Thus, if z describes the normalized absolute value of a Zig-Component $\mathfrak{z}$, and if $\mathbf{V}^i_{num}$ and $\mathbf{V}^i_{cat}$ respectively represent the numerical and categorical variables of $\mathbf{V}^i$ the final score is:

$$\Delta\mathfrak{D} \;=\; z^{\mathbf{d}}_\mu \;+\; z^{\mathbf{d}}_\sigma \;+\; \sum_{\mathbf{d}' \in \mathbf{V}^i_{num}} z^{\mathbf{d},\mathbf{d}'}_r \;+\; \sum_{d' \in \mathbf{V}^i_{cat}} z^{\mathbf{d},\mathbf{d}'}_V \qquad (7)$$

The last two terms of the equation depends on the current state of $\mathbf{V}^i$. Therefore we must update $\Delta\mathfrak{D}$ after each step.

If we use the redundancy measure $\mathfrak{G}_{hard}$, then we can avoid computing $\Delta\mathfrak{G}$ altogether: Ziggy can discard the redundant candidates before it starts building the view. Consider a candidate column $\mathbf{d}$, and let $\mathbf{V}^{1..i-1}$ describe the union of all previously built views. If $\mathfrak{G}_{hard}(\mathbf{V}^{1..i-1}, \mathbf{d}) > \mathbf{L}$, then the column is not exploitable: adding it to the current view $\mathbf{V}^i$ will breach the threshold, regardless of $\mathbf{V}^i$'s current state. Conversely, if we have $\mathfrak{G}_{hard}(\mathbf{V}^{1..i-1}, \mathbf{d}) < \mathbf{L}$, then candidate is "safe", it will never breach the dependency threshold. Thus Ziggy, builds the view in two separate steps: first it eliminates the redundant columns (i.e., those for which $\mathfrak{G}_{hard}(\mathbf{V}^{1..i-1}, \mathbf{d}) > \mathbf{L}$), then it selects the top D candidates.

To conclude, our greedy algorithm is not exact, but it runs in $\mathcal{O}(KMD)$. Thanks to this heuristic, we avoid an exhaustive search of $\binom{M}{D}$ possible view combinations, which would lead to an unpractical $\mathcal{O}(KM^D)$ runtime.



(a) Computing $\mathfrak{z}_\mu$ without staging.

(b) Computing $\mathfrak{z}_\mu$ with staging.

Figure 5: Our staging strategy.

5.2 Staging Computations

We now discuss how to compute the Zig-Components for each column of the database. This task is critical: in the best case, it requires a full scan of the database. In the worst case it runs in $\mathcal{O}(NM^2)$, because we need to compute and compare every possible pair of correlations in $\mathbf{V}_t$ and $\mathbf{V}_{\bar{t}}$ for the scores $\mathfrak{z}_r$ and $\mathfrak{z}_V$.

To support this workload, we prepare some computations offline, before the user starts submitting queries. Let us focus on the Zig-Component $\mathfrak{z}_\mu$, which reports the difference $(\bar{\mathbf{d}}_t - \bar{\mathbf{d}}_{\bar{t}})/s_{\bar{t}}$, for a given column $\mathbf{d}$ of the database. Figure 5a presents the naive method to compute this component. As soon as our user submits a selection, we compute the average $\bar{\mathbf{d}}_t$ for those tuples, the values $\bar{\mathbf{d}}_{\bar{t}}$ and $s_{\bar{t}}$ for the rest of the database, and we apply the formula $(\bar{\mathbf{d}}_t - \bar{\mathbf{d}}_{\bar{t}})/s_{\bar{t}}$ directly. Thus, we read the whole column.

Figure 5b illustrates our staging strategy. Offline, we compute the mean $\bar{\mathbf{d}}$ and the standard deviation s for the whole column $\mathbf{d}$. Online, when the user submits a selection, we compute $\bar{\mathbf{d}}_t$ and s_t only - thus, we read the selection and ignore the rest of the database. We then reconstitute $\bar{\mathbf{d}}_{\bar{t}}$ and $s^2_{\bar{t}}$, as follows:

$$N_{\bar{t}} = N - N_t \qquad (8)$$

$$\bar{\mathbf{d}}_{\bar{t}} = \frac{N \cdot \bar{\mathbf{d}} - N_t \cdot \bar{\mathbf{d}}_t}{N - N_t} \qquad (9)$$

$$s^2_{\bar{t}} = \frac{N}{N_{\bar{t}}} \cdot s^2 - \frac{N_t}{N_{\bar{t}}} \cdot s^2_t - \frac{N_t}{N} \cdot (\bar{\mathbf{d}}_t - \bar{\mathbf{d}}_{\bar{t}})^2 \qquad (10)$$

We now have all the elements to compute the Zig-Component. To obtain these equations, we used formulas designed to compute the mean and variance incrementally [16], and we reversed them - in fact we compute $\bar{\mathbf{d}}_{\bar{t}}$ and $s_{\bar{t}}$ in a "decremental" fashion.

In effect, this method does not reduce the complexity of the algorithm, but it greatly reduces the amount of tuples to

Property	Type 1	Type 2	Test Statistic	Comment
Mean	Contin.	-	$(\bar{\mathbf{d}}_t - \bar{\mathbf{d}}_{\bar{t}})/s_{\bar{\mathbf{d}}_t - \bar{\mathbf{d}}_{\bar{t}}}$	Wald test [21]
Stand. Dev.	Contin.	-	$s_t/s_{\bar{t}}$	F-test [4]
Frequencies	Discrete	-	χ^2	Pearson's χ^2 test [21]
Dependence	Contin.	Contin	$(Z_t - Z_{\bar{t}})/s_{Z_t - Z_{\bar{t}}}$	Z is the Fisher Z-transformation of the correlation coefficient r between $\mathbf{d}$ and $\mathbf{d}'$ [7]
Dependence	Contin.	Both	$(V_t - V_{\bar{t}})/s_{V_t - V_{\bar{t}}}$	V is Cramér's V between $\mathbf{d}$ and $\mathbf{d}'$. We obtain its distribution with Fisher's Normal approximation of $\sqrt{\chi^2}$ [15].

Table 2: Our choice of tests, corresponding to each Zig-Component. We use the same notations as in Table 1.

read. The smaller the user's selection is, the greater is the performance gain. Fortunately, we managed to extend it for all the Zig-Components presented in Table 1. We can derive similar computations to update correlation coefficients. If q represents the covariance between $\mathbf{d}$ and $\mathbf{d}'$:

$$q_{\bar{t}} = \frac{N}{N_{\bar{t}}} \cdot q - \frac{N_t}{N_{\bar{t}}} \cdot q_t - \frac{N_t}{N} \cdot (\bar{\mathbf{d}}_t - \bar{\mathbf{d}}_{\bar{t}}) \cdot (\bar{\mathbf{d}}'_t - \bar{\mathbf{d}}'_{\bar{t}}) \qquad (11)$$

To cope with categorical data, our approach is slightly different. Offline, we compute a histogram for $\mathbf{d}$. Online we compute another histogram for $\mathbf{d_t}$. From those, we can infer the distribution of $\mathbf{d_{\bar{t}}}$'s values, and compute $\mathfrak{z}_\chi$ and $\mathfrak{z}_V$.

6. MODEL VALIDATION

We now focus on the following problem: for a given view $\mathbf{V}$, how *significant* is the Zig-Dissimilarity $S = \mathfrak{D}(\mathbf{V}_t; \mathbf{V}_{\bar{t}})$? A high value may indicate that $\mathbf{V}_t$ and $\mathbf{V}_{\bar{t}}$ come from two different distributions. But it could also be caused by chance. How confident are we of this result? Confidence scores help Ziggy decide which Zig-Components to show, and it helps users interpret its results.

The statistics literature proposes a completely generic way to solve this problem: permutation testing. This method works with the Zig-Dissimilarity, but it can also handle other divergence measures. It consists in repeatedly shuffling the rows of $\mathbf{V}$, such that tuples are randomly affected to $\mathbf{V}_t$ or $\mathbf{V}_{\bar{t}}$. We then observe how the dissimilarity S varies: if the permutations have no effect on S, then there is high chance that the dissimilarity was caused by chance. Oppositely, if S is very sensitive, then we have a high confidence in our result. We refer the interested reader to Wasserman [21] for more details.

Permutation testing offers plenty of advantages, but shuffling the rows is computationally heavy. In our implementation, we used an alternative approach: we exploit the composite nature of the Zig-Dissimilarity, and test each Zig-Component individually. We then aggregate these scores in a synthetic confidence indicator. Therefore we do not test the Zig-Dissimilarity directly - instead we focus on its underlying effects. This method is much lighter because we can use known asymptotic results, or at least approximations thereof. For instance, we know that under certain assumptions, we can test the difference between the means with a Wald test [21], which only requires an extra $\mathcal{O}(1)$ computation. Similarly, we can use a F-test for the ratio of the variances, or a χ^2 test for the differences in categorical distributions. Table 2 summarizes our choices. To aggregate the tests, we report the minimum observed confidence, a purposely conservative approach [21].

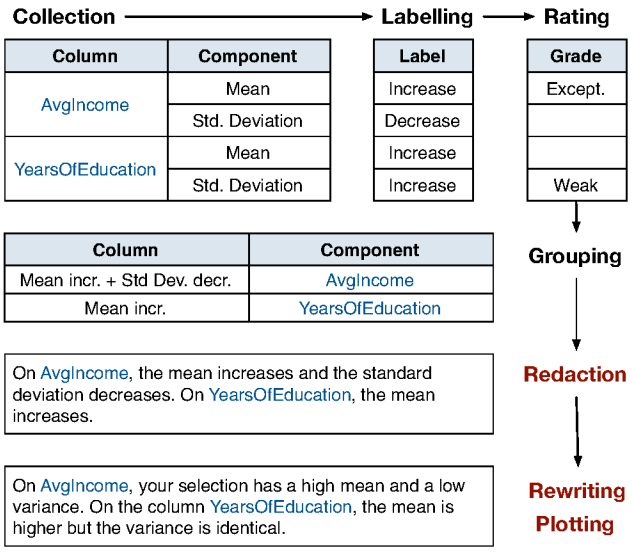


Figure 6: Ziggy's report generation pipeline.

7. REPORT GENERATION

We explained how our system detects views. We now detail Ziggy's report generation pipeline, through which it justifies its choices.

Ziggy describes its views one by one. For each view, it generates a report, comprising a few sentences and several charts. The aim of the text is to give a general presentation of how the tuples differ from the rest of the data. The charts let users inspect the details. Figure 6 illustrates how Ziggy proceeds. First, it gathers all the Zig-Components associated with each column of the database. It then generates a short description for each component using handwritten rules such as those presented in the first column of Table 3. It also rates each component, with three possible grades: "weak", "neutral", and "exceptional". These grades are based on both the values of the components and their confidence, as illustrated in Table 3. To set the thresholds, we took inspiration from classic statistics textbooks [4], and chose conservative options. Nevertheless, these quantities are inherently arbitrary. It is therefore important to communicate them to the end users, and motivate each decision with its corresponding rule.

Once Ziggy has collected, labeled and rated the Zig-Components, it groups the columns which have the same "profile", that is, the same variations on the same components. It then generates one sentence for each group, such that each sentence has the same structure: `on [column_names], the`

Description	Exceptional?	Weak?
$\mathfrak{z}_\mu \geq 0 \wedge \bar{\mathbf{d}}_{\bar{t}} \geq 0$: "increase" $\mathfrak{z}_\mu \geq 0 \wedge \bar{\mathbf{d}}_{\bar{t}} < 0$: "decrease" $\mathfrak{z}_\mu < 0 \wedge \bar{\mathbf{d}}_{\bar{t}} \geq 0$: "decrease" $\mathfrak{z}_\mu < 0 \wedge \bar{\mathbf{d}}_{\bar{t}} < 0$: "increase"	$z_\mu^{\mathbf{d}}$ in top 5%	$\lvert\mathfrak{z}_\mu^{\mathbf{d}}\rvert < 0.2$ or $p_\mu^{\mathbf{d}} < 0.01$

Table 3: Example of handwritten rules for the difference of means $\mathfrak{z}_\mu$. The variable p_μ describes the confidence (p-value), as described in Section 6.

`[zig-component] [label]`. If several components are involved, as in Figure 6, the Ziggy enumerates all the pairs `[zig-component] [label]`, separated by `and`. At this point, the text produced is understandable, but it is also highly redundant and possibly grammatically incorrect. During the last phase, Ziggy rewrites it a set of handwritten rules. Such rules include inserting connectors (e.g., "Additionally", "Finally"), using random synonyms (e.g., "the tuples", "the selection", "your data") or replacing wider chunks (e.g., "has a lower variance" by "spreads more widely").

By default, Ziggy only produces visualizations for the variables associated with exceptional components. It plots the remainder on demand. To determine which type of visualization to use, it checks the type of the columns, and applies usual techniques: it uses density plots and histograms for one dimension data, and scatterplots and heat maps for two-dimension data.

8. SETTING PARAMETERS

Ziggy's model relies on three parameters: the total number of views K to generate, the width of the views D, and the dependency threshold L.

Number of Views K. When should we stop producing views? We propose to generate as many views as possible, and delegate the final decision to the users - after all, we have little idea about what they are seeking. In practice, we do so lazily: we start with a small selection of views (e.g., as many as the display can contain), and we generate the remainder on demand, for instance with a "show me more" button. In our experience, the number of views stays manageable because the algorithm is blocked by the redundancy constraint: after a certain number of views, Ziggy finds no more columns to exploit.

Size of the Views D. In our implementation, we set this parameter in an adaptive fashion, using a method presented by Zhu and Ghodsi [25]. We summarize it as following. When building a view, we keep track of the gains $\Delta\mathfrak{D}_d$ induced by each column d. We then detect change points in the sequence (e.g., "jumps" or "elbows"). If we observe such a behavior, we truncate the current view and start a new one. The advantage of this method is that D can adapt to each subspace. However, it is only a heuristic. Fortunately, inaccuracies have little consequence in practice: if the dependency constraint is weak, then the excluded columns are simply pushed to the next view.

Dependency threshold L. Admittedly, there is no optimal way to set this parameter: it depends entirely on the user and the exploration context. By default, we use $\mathfrak{S}_{hard}$, and we limit to $L = 0.99$. This setting enforces that the views are non-overlapping, but it adds no other constraint - we see this as a safe option.

Zig-Components. Previously, we proposed several Zig-Components, in order to capture a wide range of effects. We do not have to use them all. For instance, a user interested in quick and concise results may prefer to ignore bivariate components.

Columns	Except. Comp. (%)	Weak Comp. (%)	$\mathfrak{D}_{Zig}$
Patent_applications_p_inhabitant PCT_patent_applications Personal_earnings	83.3	0	25.4
Dwellings_no_basic_facilities Educational_attainment Emp._work_very_long_hours Life_expectancy	64.2	7.1	18.9
Average_hours_worked Population_growth_rates Working_age_population Pop_under_the_age_of_15	50	42.2	12.4
Assault_rate, Homicide_rate Current_account_balance	77.7	11.1	12.1
Export_Pharmaceutical Incidence_part_time_emp Long_term_unemp Production_crude_oil	57.1	57.1	11.4
Air_pollution, Job_security Student_skills	77.7	11.1	10.1
Employment_rate Total_primary_energy_supply Trade_Balance._Pharmaceutical	66.6	33.3	7.1
Triadic_patent_year Time_devoted_to_leisure	28.5	75.5	6.2
Renewable_energy_supply Voter_turnout Total_tax_revenue	33.3	55.5	5.5
Consultation_on_rule_making Implicit_GDP_Price_Indices Years_in_education	7.1	78.5	5.3
Quality_of_support_network Taxes_on_income_and_profits	33.3	55.5	4.5
Value_Added_of_Industry Exchange_Rates Gross_Domestic_Product	0	100	2.9

Table 4: Detail of Ziggy's views, by decreasing order of dissimilarity. The two middle columns indicate the proportion of Zig-Components marked as Exceptional and Weak.

9. USE CASE

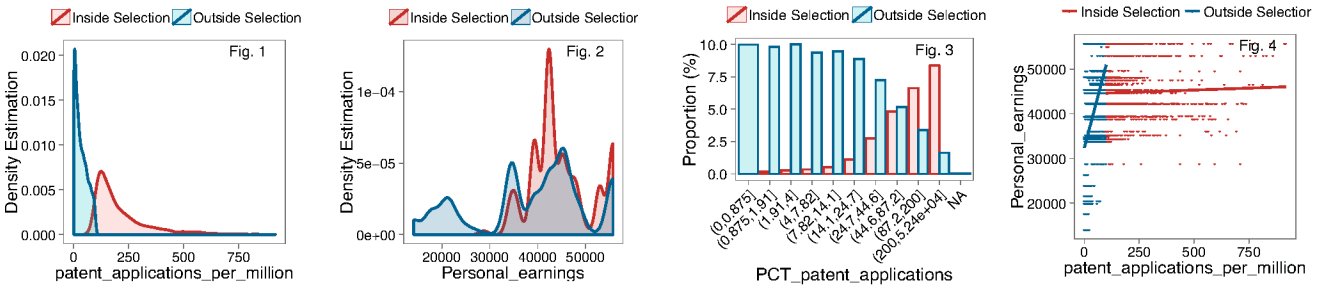
We now apply Ziggy on the data which inspired our running example. Our aim is to understand which factors lead to innovation, and more specifically patents. To answer this question, we aggregated several databases from the OECD, an international economic organization. All the data we used may be found online[1]. Our core dataset is the `Patents per Region` database, which contains 15 years of patent statistics for 2,180 regions in 31 countries. We augmented this set with several other region-level databases (`Demographics per Region` and `Labour per Region`) and country-level indicators (`Better Life`, `Well Being` and `Innovation Indicators`). We obtain a table with about 6,823 rows and 519 columns, including mixed types and missing values. We filtered out the categorical columns with more than 20 distinct values (e.g., primary keys, countries and region names).

Our selection of tuples contains the top 10% regions for

[1]http://stats.oecd.org/

Observe the following columns: patent_applications_per_million, PCT_patent_applications, and Personal_earnings.

On patent_applications_per_million, your selection has a high average but also a high variance (Fig.1). On Personal_earnings, your tuples are concentrated around a higher value (Fig. 2). On column PCT_patent_applications, the value (0,0.875] is underrepresented, while (200,5.24e+04] is overrepresented (Fig. 3).
Between columns patent_applications_per_million and Personal_earnings, the positive correlation is either weaker or reversed (Fig. 4).



Take a look at columns Assault_rate, Current_account_balance, and Homicide_rate

On Assault_rate, the average is similar, but the tuples are particularly concentrated (Fig. 1). Additionally, on column Current_account_balance, the selection has a high average but also a high variance (Fig. 2). On Homicide_rate, the data is concentrated around a low value.
Between columns Assault_rate and Current_account_balance, the negative correlation changes direction (Fig. 3). Also, between columns Assault_rate and Homicide_rate, the positive correlation is inverted (Fig. 4). Finally, between columns Current_account_balance and Homicide_rate, the negative correlation seems stronger.
I discarded 1 effect, considered as weak. Click here to see it.

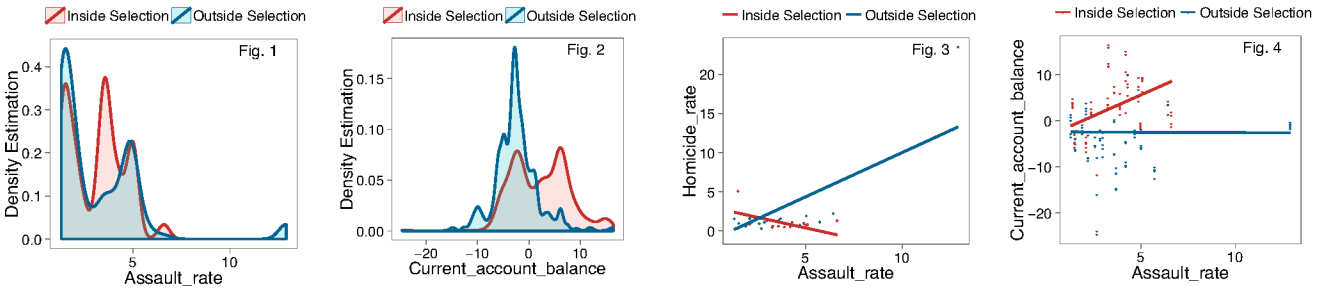


Figure 7: Ziggy's explanations for Views 1 and 4

the statistic `patent applications per inhabitant`. We set $D = 6$, with the adaptive stopping method described in Section 8. We use $\mathfrak{S}_{hard}$, with a maximum value of $L = 0.75$. Ziggy detects a total of 12 views, which columns are reported in Table 4.

Some of Ziggy's choices are not surprising. For instance, the first column mentioned in the Table is precisely the variable we used to define our selection. The second one, `PCT patent applications` is highly similar (the PCT is an international patent treaty, which allows transnational patents). Likewise, we expected some relationship between education and innovation (views 2, 6 and 10). However, some effects are less straightforward. How does `Employees working very long hours` impact innovation? Are patents produced at work, or during hobby time? Similarly, how do our regions behave on the variable `Job security`?

Figure 7 presents Ziggy's explanations for two views. To obtain this figure, we made only two edits: we removed some plots to save space, and we inserted references to the charts in the text. Consequently, the figure illustrates accurately what Ziggy's users would obtain. The first view reflects the fact that innovative regions usually offer high incomes. Ziggy expresses this through its comments about the variable `Personal earnings`, but also through the last chart. As it points out, there is a correlation between patent applications and income, but this correlation disappears when we focus on innovative regions. Then, the regression line is flat, with a high offset, which indicates that all these regions are relatively rich, regardless of their patent statistics.

The second view offers a different perspective. The first and third charts show that innovative regions tend to be safer, but the relationship is not straightforward. Ziggy shows that these regions have less extreme values on `Assault rate`, but the mean is similar. Also, the expected correlation between `Assault rate` and `Homicide rate` is inverted. This reflects the fact that assaults do exist in our innovative regions, but little of them actually lead to a homicide. The last chart is more puzzling: normally, there exists no relationship between `Assault rate` and `Current account balance`. And indeed, we expect these variables to be independent, because they describe completely different topics. Yet, in our regions, a clear correlation appears. How can we interpret this effect? If this dependence completely spurious? Are those variables cofounded by a third, hidden variable? Or maybe sane public accounts causes anger among inventors? As implausible as this last hypothesis may seem, the chart gives us no way to discard it. We leave the question open for future investigations.

10. EXPERIMENTS

Metrics. We now present our experimental results. We evaluate three aspects of our system: the *quality* of the views, their *diversity*, and Ziggy's *runtime*. To evaluate the quality of the views, we simulate users with statistical classifiers. We assume that if a classifier can learn from a view, then so can a real user. Technically, we materialize the user's selection into a vector $\mathbf{t} = (t_1, \ldots, t_n)^\top$: $t_i = 1$ if the tuple is chosen, 0 otherwise. We then train a classifier, using the view $\mathbf{V}$ as feature set and the user's selection $\mathbf{t}$ as target. To obtain our quality score, we check if the classifier can accurately model the user's selection $\mathbf{t}$ with the variables in $\mathbf{V}$. If so, we deduce that the selection has a "peculiar" structure in the subspace, and therefore the view contains exploitable information. Oppositely, if the classifier cannot reconstruct the user's selection, then either the classifier is poor, or the view contains no information about the selection. We use a 5-Nearest Neighbor (5-NN) classifier, and we report the F1 measure with 5 cross-validation (higher is better). We chose the 5-NN algorithm for convenience: it is fast, and it gave us good predictive performance.

To measure diversity, we report the number of distinct columns used by the views in the result set (higher is better). We measure runtime with the total wall clock time, including preprocessing in Ziggy's case (lower is better).

Baselines. We compare Ziggy to four state-of-the-art subspace detection methods from the data mining literature. Our first three methods are supervised: their aim is to detect informative columns for a classification or regression task. We adapt these methods to our problem by setting $\mathbf{t}$ as the target column. Here again, our rationale is the following: if a set of column is a good predictor for the user's selection, then it contains useful information.

Our first baseline is `Claude` [18], a recently published feature search algorithm. We chose this method because it uses a multi-view approach, and therefore its results are directly comparable to ours: like Ziggy, it returns a fixed number of subspaces, with a user-specified number of dimensions. Technically, Claude differs on two points. First, it uses a different notion of interestingness: it seeks groups of columns which are strongly dependent to the target, dependency being measured with the mutual information. Second, Claude builds subspaces with a level-wise, beam search algorithm. We used the authors' implementation. We expect this approach to be both fast and accurate.

Our second baseline is `Clique`, inspired by the pattern mining literature [23]. The algorithm builds a graph inside which the weight of each edge (i, j) represents the statistical dependency between the pair of columns i, j and the target variable (again, we measure dependency with the mutual information). To obtain K subspaces with at most N variables, we remove all the edges except the $K' > K$ strongest, and detect cliques of at most N nodes in the remaining graph. By default, we set $K' = 2 \cdot K$. To detect cliques, we used the `igraph` software package. We expect this algorithm to be fast but rather approximative.

Our third baseline, `Wrap-5NN` is a "wrapper" from the feature selection literature [8]. This algorithm trains 5-NN classifiers with increasingly large sets of variables. First, it tests each variable, and it keeps the top K. Then, it tests combinations of two columns. The process is repeated until the views reach D dimensions. We chose 5-NN because it is the same algorithm we use to evaluate the views. Therefore, we optimize exactly what we measure. We expect this approach to be accurate, but also very slow. Thus, we only use it as a "gold standard" for our experiments with real data.

Parameter	Value
Selection (tuples)	3,000
Tuples from Gauss. mixture	15,000
Tuples from Unif. distrib.	15,000
Sep. / Non-sep. variables	20 / 4
Num. dimensions subspaces	4
Num. components Gaussians	5
Mean / Variance Gaussians	Unif. in [-10,10] / [1, 20]
Uniform noise	Unif. in [-45,45]

Table 5: Default parameters for our data generator.

The last baseline, `4S`, detects subspaces in an unsupervised manner: it seeks subspaces which contain clusters and outliers, independently of the user's selection. To do so, it detects groups of variables which are strongly mutually dependent, using an multivariate correlation measure. We used the author's implementation written in Java. We expect `4S` to be fast, but also less accurate than its competitors because of it unsupervised nature.

Setup. We implemented Ziggy in R, exploiting its native primitives for critical operations (namely computing means, covariance matrices and cross-tabulation). We interrupted all the experiments which lasted more than 1 hour. Our test system is based on a 3.40 GHz Intel(R) Core(TM) i7-2600 processor. It is equipped with 16 GB RAM, but the Java heap space is limited to 8 GB. All the algorithms we present are single-threaded. The operating system is Fedora 16.

10.1 Synthetic Data

In this set of experiments, we benchmark our algorithms in a synthetic environment. Since we know the structure of the data, we can tune the competitors optimally. For instance, if we generate a dataset with 3 subspaces of 5 columns, then we set $K = 3$ and $D = 5$. We must however report that `4S` tunes itself: it computes how many subspaces to generate, and how large these should be. We use two versions of Ziggy: for `Ziggy-Soft` we use the dependency measure $\mathfrak{S}_{soft}$ and we limit it to 0.1. For `Ziggy-Hard`, we use $\mathfrak{S}_{hard}$ and we limit it to 0.9.

Our generator produces columns by groups. It yields two types of subspace: *non-separated* subspaces, and *separated* subspaces. In the non-separated case, the selection and the rest of the data are sampled from the same distribution, namely a mixture of multivariate Gaussians with random parameters. In the separated case, the selection is sampled from a separate Gaussian distribution, with its own random parameters. Additionally, our generator produces uniform noise on all the columns. Table 5 presents our parameters. For each experiment, we generate 4 random data sets and report the average F1 of all the views.

Quality of the Views. The first chart in Figure 8 presents the robustness of the algorithms with regards to the size of the selection. For all our baselines, raising this parameter ameliorates the views, but the quality converges fast (at around 15% of the database size). The algorithm `Claude` comes first, but it is very closely followed by the two instances of `Ziggy` and `4S`. The ranking is almost identical in the second chart, which displays the accuracy of the algorithms varying the dimensionality of the subspaces. All algorithms involved seem robust, but `Ziggy-Soft`, `Ziggy-Hard`

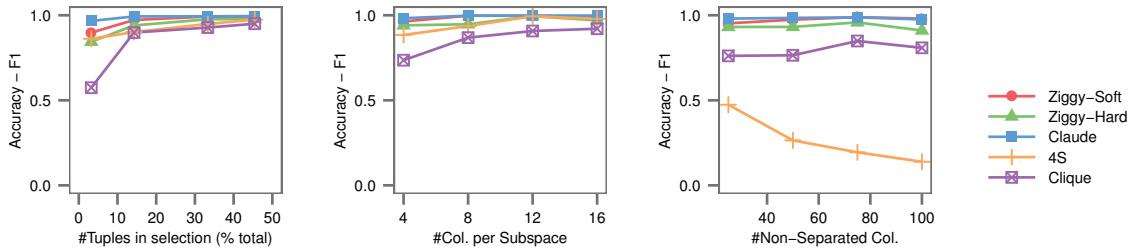
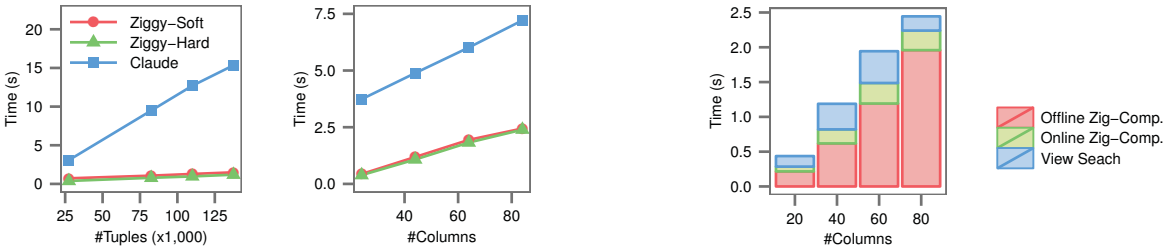


Figure 8: Average quality of the views varying the data generator parameters.



(a) Runtime for the three fastest algorithms, varying data parameters. The points for `Ziggy-Hard` and `Ziggy-Soft` overlap.

(b) Breakdown of the total runtime for `Ziggy-Soft`, varying the number of columns.

Figure 9: Runtime experiments with synthetic data.

and `Claude` are above, followed closely by `4S`, then `Clique`. The last graph illustrates the robustness of the views with regards to the number of non-separated columns. All our baselines achieve good scores, except `4S`. We interpret this effect by the fact that `4S` is unsupervised, it has therefore no way to detect which subspaces are interesting and which are not. In conclusion, despite its simple assumptions, Ziggy delivers high quality, robust views, largely comparable to state-of-the-art feature selection algorithms.

Runtime. Figure 9a illustrates the total runtime with regards to the number of rows and columns in the database. We ignored the algorithms `4S` and `Clique`, which were slower than the other candidates. We observe that Ziggy's performance is spectacular: it is an order of magnitude faster than `Claude`, which is itself faster than the other competitors. And yet, all the algorithms involved in our benchmark have the same $\mathcal{O}(ND^2)$ complexity. We explain the difference with Ziggy's simpler computations. Ziggy relies on means and correlations, which are much lighter than `Claude`'s information theoretic estimators. Besides, when evaluating its views, Ziggy considers at most two dimensions at a time, while its competitors test higher level dependencies.

Figure 9b shows where `Ziggy-Soft` spends its time. By far, the most expensive operations are the offline computations. This validates our staging strategy. Table 5 reports that the database contains 1 selected tuple for every 10 non selected tuple. Therefore we expected the online phase to be about 10 times faster, ignoring overhead costs. The figure confirms this estimation.

Diversity. Figure 10 presents the number of distinct columns mentioned in the views. In the leftmost chart, we compare four competitors, varying the width of the subspaces. We notice two distinct groups of algorithms. The approaches `Ziggy-Hard` and `4S` offer a very high diversity, because they enforce that no column is shared among subspaces. The approaches `Claude` and `Ziggy-Soft` offer much less variety. The rightmost figure illustrates the effect of the

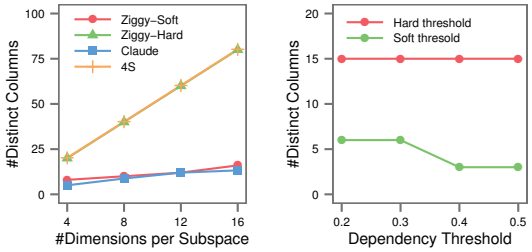


Figure 10: Variety of the views.

dependency threshold. In the Hard case, it has no apparent effect because the views are completely disjoint anyway (the threshold does decorrelate the views, but this does not influence our metric). In the Soft case, we see that our deduplication strategy works: a lower threshold forces Ziggy to introduce variety.

10.2 Real Data

We now presents our experiments on real data from the UCI Repository. We use the algorithm `Wrap-5NN` as "gold standard", since it optimizes precisely the metric we report. To set the number and width of the subspaces, we rely on `4S`. Table 6 describes the datasets and our settings. Because all the competitors have the same parameters, the comparison is fair.

Accuracy. Figure 11 illustrates the quality of the views for each algorithm. We observe that `Wrap 5-NN` always comes first. It is closely followed by `Ziggy-Soft`, `Claude` and `Ziggy-Hard`, in different orders (although `Crime` is a striking counter-example). The unsupervised `4S` follows in most cases, tailed by `Clique`. Here again, we conclude that Ziggy's performance is largely comparable to good feature selection algorithms. However, we observe that `Ziggy-Hard` is often below `Ziggy-Soft`, the most extreme case being the `Let-`

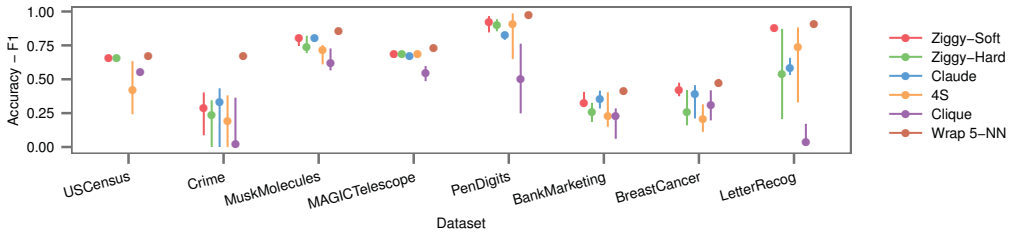


Figure 11: Quality of the views. The points represent median scores, the bars represent the lowest and greatest scores.

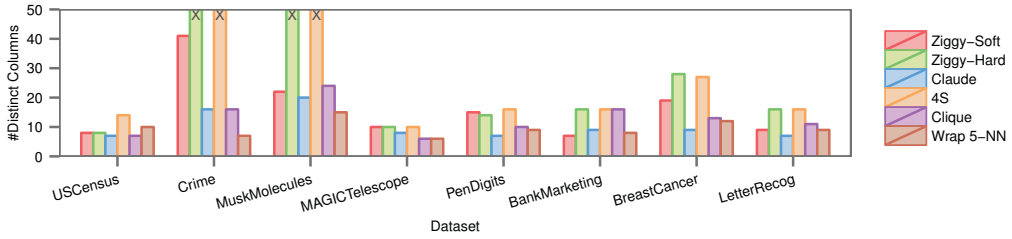


Figure 12: Diversity of the views. The `X` mark indicates that we truncated the column.

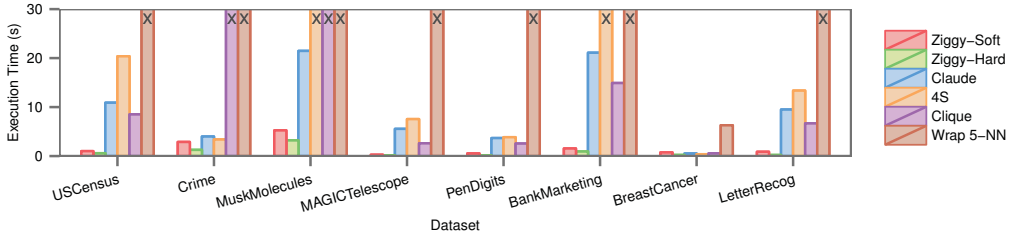


Figure 13: Runtime. The `X` mark indicates that we truncated the column.

`terRecog` set. This is a consequence of `Ziggy-Hard`'s diversity. In the synthetic case, it had several equally good subspaces to chose from. In the last three datasets, it seems that some subspaces are better than others, and therefore non-redundancy induces an accuracy loss (observe that `4S` suffers from the same problem).

Diversity. Figure 12 presents the diversity results. As with synthetic data, we observe that `4S` and `Ziggy-Hard` dominate the other algorithms, particularly with wide tables such as `Crime` and `MuskMolecules`. The algorithms `Ziggy-Soft` and `Clique` follow, then `Claude` and `Wrap-5NN` comes last - which we expected since they mostly target accuracy. This chart, in conjunction with Figure 11, shows that the algorithms which generate the best F1 rarely generate the best diversity, and reciprocally. This motivates our choice to offer both $\mathfrak{S}_{hard}$ and $\mathfrak{S}_{soft}$.

Runtime We present the runtime of our algorithms in Figure 13. The results are consistent with our previous conclusions: Ziggy outperforms all of its competitors, and the speedup gets more dramatic as the size of the datasets increase.

11. RELATED WORK

Outlier Description. Outlier description consists in finding subspaces inside which a particular tuple is an outlier [3, 6, 9, 24]. This task inspired our work, but its objectives are different. Outlier description describes single objects, while we describe sets of tuples. It focuses on distance-based outliers, while we focus on probability distributions,

Dataset	Columns	Rows	#Views	Dim Views
MuskMolecules	167	6,600	22	18
Crime	128	1,996	20	17
BreastCancer	34	234	10	13
PenDigits	17	7,496	9	10
BankMarketing	17	45,213	11	8
LetterRecog	16	20,000	10	12
USCensus	14	32,578	10	7
MAGICTelescope	11	19,022	1	10

Table 6: Characteristics of the datasets.

regardless of how central or isolated the user's selection is. Authors focus on specific data types (either numerical or categorical, but not both), and little of them mention redundancy (an exception is [6], which seeks closed sets). None of these works discuss how to report results.

Contrast Mining. Contrast mining is a similar task, but in a pattern mining context: the aim is to differentiate two or more populations by identifying patterns which are present (e.g., have a high support) in one population but not in the other [20, 22]. This line of work is related but orthogonal, because we deal with neither itemsets nor patterns. Notably, Loekito and Bailey introduced a method to identify discriminative variables [13]. Yet, their work focus on "contrast of contrasts" for categorical data, a close but different task

Feature Selection. As noted in Section 10, our work is intimately related to feature selection. Feature selection

seeks predictive variables for a classification or regression task [8]. We can map this task to our problem by treating the user's selection as a column to be predicted. The main difference is that feature selection targets statistical accuracy, while we target interpretation. Thus, most feature selection algorithms seek one optimal set of variables, while we seek several small sets of variables. Also, feature selection tends to optimize class separability, while we are interested in any difference in distribution. Nevertheless, we acknowledge the similarity between our work and some algorithms, and compare them directly in Section 10.

Data Exploration Recent works in data exploration have tackled different but related problems. Similarly to our work, SeeDB [19] recommends visualizations by seeking columns on which a set of tuples have an unusual behavior. However, it focuses on `GROUP BY` aggregates in data warehouses, while we focus on describing the general distribution of the selection, independently of any sub-grouping. Li and Jagadish [10] tackle the counterpart of our problem: they describe how to issue queries in natural language. Lloyd et al.'s automated statistician [12] describes regression models in natural language. These approaches are somewhat complementary to ours, and there would be much to gain by combining them.

12. CONCLUSION

We exposed the tuple characterization problem, and presented our solution, Ziggy. Our experiments show that Ziggy can produce informative, non-redundant results within a fraction of the time taken by state-of-the-art machine learning algorithms. In fact, we could apply our solution to a wide range of problems beyond the strict realm of data exploration, including supervised and unsupervised learning.

We are genuinely excited by the perspectives offered by our results. In the future, we will focus on human aspects: we will investigate how real users interact with Ziggy, and which graphical tools can enhance their experience. We will also enrich Ziggy with other measures of difference, possibly with Bayesian modeling. Finally, the problem of how to describe tuples with natural language is immense, but it has, to the best of our knowledge, only received little attention. We are convinced that this field is full of promises, for data explorers and data researchers alike.

13. REFERENCES

[1] A. Abouzied, J. Hellerstein, and A. Silberschatz. Dataplay: interactive tweaking and example-driven correction of graphical database queries. In *USIT*, pages 207–218. ACM, 2012.

[2] R. Agrawal, J. Gehrke, D. Gunopulos, and P. Raghavan. Automatic subspace clustering of high dimensional data for data mining applications. In *Proc. SIGMOD*, pages 94–105, 1998.

[3] F. Angiulli, F. Fassetti, and L. Palopoli. Detecting outlying properties of exceptional objects. *ACM TODS*, 2009.

[4] J. Cohen. *Statistical power analysis for the behavioral sciences*. Lawrence Erlbaum Associates, 1977.

[5] K. Dimitriadou, O. Papaemmanouil, and Y. Diao. Explore-by-example: An automatic query steering framework for interactive data exploration. In *Proc. SIGMOD*, pages 517–528, 2014.

[6] L. Duan, G. Tang, J. Pei, J. Bailey, A. Campbell, and C. Tang. Mining outlying aspects on numeric data. *Data Mining and Knowl. Discovery*, pages 1–36, 2014.

[7] R. A. Fisher. Frequency distribution of the values of the correlation coefficient in samples from an indefinitely large population. *Biometrika*, 1915.

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

[9] E. M. Knorr and R. T. Ng. Finding intensional knowledge of distance-based outliers. In *Proc. VLDB*, pages 211–222, 1999.

[10] F. Li and H. Jagadish. Constructing an interactive natural language interface for relational databases. In *Proc. VLDB*, pages 73–84, 2014.

[11] E. Liarou and S. Idreos. dbtouch in action database kernels for touch-based data exploration. In *Proc. ICDE*, pages 1262–1265, 2014.

[12] J. R. Lloyd, D. Duvenaud, R. Grosse, J. B. Tenenbaum, and Z. Ghahramani. Automatic construction and Natural-Language description of nonparametric regression models. In *AAAI*, 2014.

[13] E. Loekito and J. Bailey. Mining influential attributes that capture class and group contrast behaviour. In *Proc. CIKM*, pages 971–980, 2008.

[14] H. V. Nguyen, E. Muller, and K. Bohm. 4s: Scalable subspace search scheme overcoming traditional apriori processing. In *IEEE Big Data*, pages 359–367, 2013.

[15] J. K. Patel and C. B. Read. *Handbook of the normal distribution*, pages 204–205. CRC Press, 1996.

[16] P. Pébay. Formulas for robust, one-pass parallel computation of covariances and arbitrary-order statistical moments. *Tech. report, Sandia National Laboratories*, 2008.

[17] T. Sellam and M. L. Kersten. Meet charles, big data query advisor. In *CIDR*, 2013.

[18] T. Sellam, E. Müller, and M. Kersten. Semi-automated exploration of data warehouses. In *proc. CIKM*, pages 1321–1330, 2015.

[19] M. Vartak, S. Rahman, S. Madden, A. Parameswaran, and N. Polyzotis. Seedb: Efficient data-driven visualization recommendations to support visual analytics. *Proc. VLDB*, pages 2182–2193, 2015.

[20] J. Vreeken, M. Van Leeuwen, and A. Siebes. Characterising the difference. In *Proc. SIGKDD*, pages 765–774, 2007.

[21] L. Wasserman. *All of statistics: a concise course in statistical inference*. Springer, 2013.

[22] G. I. Webb, S. Butler, and D. Newlands. On detecting differences between groups. In *Proc. SIGKDD*, pages 256–265, 2003.

[23] Y. Xie and P. S. Yu. Max-clique: a top-down graph-based approach to frequent pattern mining. In *ICDM*, pages 1139–1144, 2010.

[24] J. Zhang and H. Wang. Detecting outlying subspaces for high-dimensional data: the new task, algorithms, and performance. *Knowledge and information systems*, pages 333–355, 2006.

[25] M. Zhu and A. Ghodsi. Automatic dimensionality selection from the scree plot via the use of profile likelihood. *Computational Statistics & Data Analysis*, pages 918–930, 2006.