Semi-Automated Exploration of Data Warehouses

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

Emmanuel Müller
KIT
Karlsruhe, Germany
emmanuel.mueller@kit.edu

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

ABSTRACT

Exploratory data analysis tries to discover novel dependencies and unexpected patterns in large databases. Traditionally, this process is manual and hypothesis-driven. However, analysts can come short of patience and imagination. In this paper, we introduce Claude, a hypothesis generator for data warehouses. Claude follows a 2-step approach: (1) It detects interesting views, by exploiting non-linear statistical dependencies between the dimensions and the measure. (2) To explain its findings, it detects local patterns in these views and describes them with SQL queries. Technically, we derive a model of interestingness from fundamental information theory. To exploit this model, we present aggressive approximations and heuristics, allowing Claude to be fast and more accurate than state-of-art view selection algorithms.

Categories and Subject Descriptors

H.2.8 [**Database Applications**]: Data mining

Keywords

Data exploration; Query recommendation; Feature selection; Subgroup discovery

1. INTRODUCTION

Businesses and scientists willing to extract knowledge from databases face a dilemma. On one hand, software editors have commercialized quantity of visual Business Intelligence tools, such as Tableau or Qlik. Yet, these rely almost entirely on users' expertise, intuition, and patience. In an exploratory scenario, when users know little about their data, such tools can involve time-consuming cycles of trial and error. On the other hand, automated, machine learning methods have gained much popularity, as shown by the recent success of industrial "data scientists". Nevertheless, at the time of writing this paper, these experts are still a rare and expensive resource. Can we find a middle way? Can we design a flexible, accessible method to automatically analyze data warehouses?

Consider the following scenario. A government analyst studies criminality in US cities. The database describes how a *measure*, the number of crimes, varies along several dozen *dimensions*, e.g., the population, unemployment rate, or location. Which cities are prone to crime? Can we identify causes, correlations, or patterns? For our analyst, inspecting how every possible combination of dimensions correlates with crime is close to impossible. But important observations could hide behind unexpected dependencies. Our aim is to automatically *generate hypotheses* about what influences the measure. We want to synthesize SQL queries which highlight what causes crime to vary, and how these variations occur. Given such queries, even lay users could explore their data and detect unexpected patterns.

We believe that this problem is extremely common in both business and science, where multi-dimensional data warehouses have been deployed for years. Addressing it with an automated method would allow analysts and researchers to make discoveries more quickly, focusing on interpretation rather than writing queries. Such a technique could also provoke serendipitous findings, e.g., highlighting a surprising correlation or confronting an ill-posed hypothesis. Besides, the problem is getting more pressing as data warehouses grow in size and complexity: manual exploration can turn out to be a painful exercise when dealing with hundreds of columns.

Automating data exploration is a difficult problem for two reasons. First, how do we recognize "interesting" queries? There is, to our knowledge, no universal measure of interestingness. The challenge is to devise a theoretically well-founded model, general enough to cover a wide range of use cases. The second problem is efficiency: given a measure of interest, how can we explore the space of all database queries quickly enough to support large datasets? We must design efficient methods to traverse all possible combinations of dimensions.

Several semi-automated exploration frameworks were proposed in the past, in the context of OLAP data cubes [17, 10]. They assumed that databases contained less than a dozen well-known dimensions. The whole challenge was to drill-in correctly, in order to locate "exceptions", e.g., local anomalies. We believe that many of these assumptions do not hold anymore. In our model, databases can contain more than a hundred dimensions with mixed types. Analysts have little to no knowledge about the effects of these dimensions, even less about their combination and their effect on a given

CIKM'15, October 19–23, 2015, Melbourne, Australia.
© 2015 ACM. ISBN 978-1-4503-3794-6/15/10 ...$15.00.
DOI: http://dx.doi.org/10.1145/2806416.2806538.

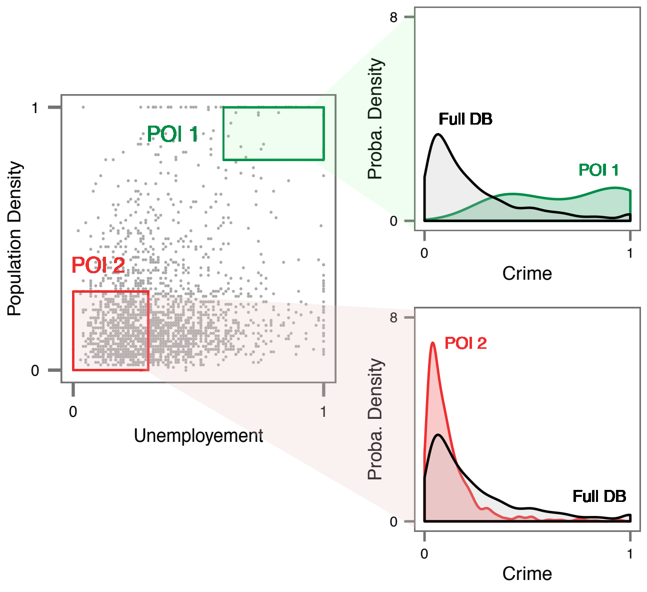

Figure 1: Example view, with two points of interest. All values are normalized.

measure. Therefore, our challenge is to depict a general picture of how the measure varies across any possible view on the database.

In this paper, we introduce *Claude*, a hypothesis generator for data warehouses. Claude's aim is to detect interesting views, i.e., interesting combination of dimensions, and highlight local patterns in these views. We make three contributions: (1) We derive a model of what makes a view "interesting" from fundamental information theory. (2) We present aggressive methods to detect good views in large databases, based on approximations and greedy search. (3) We apply our model to real-life situations, involving either humans (data exploration by subgroup discovery) or automatic predictors (classification algorithms).

The rest of this paper is organized as follows. Section 2 gives an overview of our model, which we refine in Section 3. Sections 4 describes how to compute the quality a view, Section 5 presents our view detection algorithm, and Section 6 introduces how to detect points of interest. We present our experiments in Section 7, related work in Section 8 and conclude in Section 9.

2. PROBLEM FORMULATION

Let us define our problem setting. We assume that the data is given in flat database table DB. This table contains two types of columns: the *dimensions* $X_0, \ldots, X_N$ and a *measure* (or *target*) T. This model is logical, we are oblivious to the physical structure of the data. Our aim is to describe how the measure varies across the dimensions. To do so, we want to detect **views**, i.e. *interesting* selections of dimensions. We also want to show *why* these views are interesting, by detecting **points of interests** (POI). A point of interest is a region in the view where the measure behaves unusually.

We demonstrate these notions with a running example. We want to understand what causes crime in US communities. We have a database of cities with several dozen socio-economic indicators (for instance, employment, age, or diplomas). For each city we also have the annual number of violent crimes: this is our target variable. Figure 1 presents an example view. The leftmost scatterplot pictures the dimensions `Unemployment` and `Population Density`. Why did Claude pick these two columns? The two POIs provide explanations. Cities with high densities and high unemployment rates have more crimes (cf. POI_1). In contrast, cities with low densities and lower employment rates tend to be safer (cf. POI_2).

In our example, `Unemployment` and `Population Density` are interesting because their combination influences the target `Crime`. We can measure this relationship with statistical dependence. This is the most basic assumption behind our study: a view is interesting if its dimensions are *jointly dependent* to the measure. We evaluate the degree of this dependence with our notion of **view strength**.

DEFINITION 1. *Consider a view $V = \{X_1, \ldots, X_D\}$, the target T, and measure of statistical dependence $\mathfrak{S}$. We define the strength of the view V as follows:*

$$\sigma(V) = \mathfrak{S}(\mathcal{X}_1, \ldots, \mathcal{X}_D; \mathcal{T}) \quad (1)$$

We assume that each database column X contains samples from a random variable $\mathcal{X}$ and measure the dependence accordingly.

Observe the two POIs in Figure 1. The measure's distribution in these regions differs from its distribution in the rest of the database: it is skewed to the right in POI_1, it is skewed to the left in POI_2. We name this property **POI divergence**. Intuitively, a region is divergent if the target behaves unexpectedly for its tuples. Consider a view based on the random variables $\mathcal{X}_1, \ldots, \mathcal{X}_D$, with respective sample spaces $\Omega_1, \ldots, \Omega_D$. The set $R \subset \Omega_1 \times \ldots \times \Omega_D$ represents a region in this view. The random variable $\mathcal{T}$ represents the target *for the whole database*. The random variable $\left[\mathcal{T} | (\mathcal{X}_1, \ldots, \mathcal{X}_D) \in R\right]$ represents the target *for the tuples within* R. We shorten this notation to $\mathcal{T}|R$. The region R is a good point of interest if $\mathcal{T}|R$ and $\mathcal{T}$ have large differences in distribution.

DEFINITION 2. *Let $R \subset \Omega_1 \times \ldots \times \Omega_D$ represent a region in the view $\{X_1, \ldots, X_D\}$, and let $\mathcal{T}$ represent our target variable. The function $\mathfrak{D}$ measures the dissimilarity between two probability distributions. We define R's* ***divergence****:*

$$\delta(R) = \mathfrak{D}\big(\mathcal{T}|R; \mathcal{T}\big) \quad (2)$$

Before we present our instantiation of strength and divergence in Section 3, we formulate our general problem statement:

PROBLEM 1. *Consider a dataset DB, a target column T, a measure of statistical dependence $\mathfrak{S}$ and a measure of distribution dissimilarity $\mathfrak{D}$. Find the top K strongest views with at most D columns. For each of these views, find the top P divergent POIs.*

To solve this problem, Claude operates in two steps. First, it detects K strong sets of columns. We call this step *column search*. Then, it extracts P POIs for each view. This is the *POI detection* step.

In the rest of this paper, we will illustrate Claude's views with mathematical notation or visualizations. In practice however, Claude expresses its recommendations with SQL queries. It returns points of interest as follows:

```
SELECT X1, ... , Xd, T
FROM DB
WHERE X1 BETWEEN [L1, H1]
  AND ...
  AND Xd BETWEEN [Ld, Hd]
```

In this query, `X1`, ..., `Xd` represent the variables of the view, `[L1, H1]` ... `[Ln, Hn]` represents the bounds of the POI, and T represents the target.

3. INTERESTINGNESS MODEL

We presented views and POIs without specifying any measure of dependence $\mathfrak{S}$ or dissimilarity $\mathfrak{D}$. In the following section, we instantiate these quantities using fundamental information theory.

3.1 View Strength

A set of dimensions is interesting if its columns are jointly dependent to the target. To quantify this dependence, we use *mutual information*. This measure presents many advantages: it is sensible to non-linear dependences, it can cope with any kind of variables, and it is practical to compute.

The *entropy* $H(\mathcal{X})$ of a variable $\mathcal{X}$ describes its variability [3]. If $\mathcal{X}$ has a constant value, then $H(\mathcal{X}) = 0$. In contrast, if $\mathcal{X}$ is highly unpredictable (e.g., $\mathcal{X}$ is the outcome of flipping a perfectly balanced coin) then $H(\mathcal{X})$ is maximal. Formally, if $\mathcal{X}$ is a discrete variable with sample space Ω, then we have:

$$H(\mathcal{X}) = -\sum_{x \in \Omega} P(\mathcal{X} = x) \cdot \log P(\mathcal{X} = x) \tag{3}$$

If $\mathcal{X}$ is continuous with density p, we define it as follows:

$$H(\mathcal{X}) = -\int_{-\infty}^{+\infty} p(x) \cdot \log p(x) \mathrm{d}x \tag{4}$$

We can use the entropy to describe how variables interact. If two variables are dependent, then conditioning (e.g., restricting the range of values) on one variable will affect the other. In our example, cities with high unemployment have high levels of crime. Therefore, conditioning on the variable `Unemployment` decreases the uncertainty of the variable `Crime`. This causes a loss in entropy, and the value of this loss is the mutual information. Formally, if $\mathcal{X}$ and $\mathcal{T}$ are two random variables, the expression $H(\mathcal{T}|\mathcal{X} = x)$ describes the entropy of $\mathcal{T}$ *given* $X = x$. If we average this expression over all possible values of x, we obtain the conditional entropy: $H(\mathcal{T}|\mathcal{X}) = \mathbb{E}_x[H(\mathcal{T}|\mathcal{X} = x)]$. We define the mutual information I as follows:

$$I(\mathcal{X}; \mathcal{T}) = H(\mathcal{T}) - H(\mathcal{T}|\mathcal{X}) \tag{5}$$

The mutual information is the loss in entropy between $\mathcal{T}$ and $\mathcal{T}|\mathcal{X}$. It is symmetric, and it is always positive or null. We can generalize this quantity to joint distributions, which gives us our new, refined version of view strength:

$$\boldsymbol{\sigma(V) = I(\mathcal{X}_1, \ldots, \mathcal{X}_D; \mathcal{T})} \tag{6}$$

A view is strong if the mutual information between the target and its dimensions is high.

In practice, we do not know the distributions of the variables $\mathcal{X}_n$, but we have access to the samples X_n. Therefore, we *estimate* the strength. If the variables are discrete, we set $\hat{P}(\mathcal{X}_n = x)$ to be the proportion of tuples with $X_n = x$.

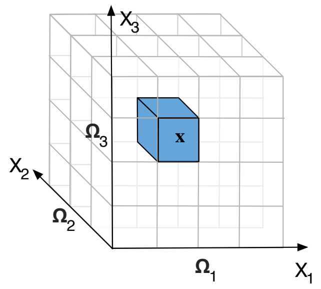

Figure 2: Example of view with three discrete variables $V = \{X_1, X_2, X_3\}$. The average divergence of the cells $\delta(\mathbf{x})$ equals the strength of the view $\sigma(V)$.

We "plug" this estimator in Equations 3, to obtain the estimated entropy $\hat{H}(\mathcal{X})$. We then use this approximation in Equation 5. Dealing with continuous vari'ables is more difficult, for two reasons. First, estimating the density function in Equation 4 is costly, especially with multivariate distributions. Second, it is not clear how to deal with mixed datasets, e.g. continuous and discrete dimensions, or discrete target and continuous dimensions. Therefore, Claude bins all the continuous variables, and treats them like discrete columns.

3.2 Points of Interest and Divergence

Let us now refine our definition of divergence. We established that the divergence of a POI is the dissimilarity between the target's distribution within this POI, and the target's distribution in the whole database. To measure this dissimilarity, we use the *Kullback-Leibler divergence* (KL). The KL divergence measures the difference between two probability distributions. It is null if the two distributions are similar, and it grows when the distributions differ. Formally, if $\mathcal{X}$ and $\mathcal{Y}$ are two discrete random variables with the same sample space Ω, we have:

$$KL(\mathcal{X} \parallel \mathcal{Y}) = \sum_{x \in \Omega} P(\mathcal{X} = x) \cdot \log \frac{P(\mathcal{X} = x)}{P(\mathcal{Y} = x)} \tag{7}$$

As Claude discretizes the continuous variables, we ignore the continuous case (cf. Section 3.1). Our region R is a good point of interest if $KL(\mathcal{T}|R \parallel \mathcal{T})$ is as large as possible:

$$\boldsymbol{\delta(R) = KL(\mathcal{T}|R \parallel \mathcal{T})} \tag{8}$$

In our model, view strength and POI divergence are closely related. Consider a view V, made of discrete variables. If we compute the divergence of each tuple and average the results, we obtain the V's strength. We illustrate this property with Figure 2. Therefore, strength and divergence are "two sides of the same coin". We formalize this property with the lemma below.

LEMMA 1. *If V is a view with d discrete variables and $\mathbf{x} \in \Omega_1 \times \ldots \Omega_D$ is a tuple from this view, then:*

$$\sigma(V) = \mathbb{E}_{\mathbf{x}}\big[\delta(\{\mathbf{x}\})\big] \tag{9}$$

PROOF. The random vector $\mathbf{X}$ describes the columns of V. Applying Bayes's theorem to Cover and Thomas, Equation 2.35 [3], we obtain $I(\mathbf{X}; \mathcal{T}) = \mathbb{E}_{\mathbf{x}}[KL(T|\{\mathbf{x}\} \parallel T)]$. Substituting the left side with Equation 6, and the right side with Equation 8, we obtain the lemma. $\square$

Suppose that we obtained a view V by discretizing a set of continuous variables V^*. The average divergence of V's bins equals the strength of V, but not that of V^*. Fortunately, these quantities converge as the bins get small.

Lemma 2. *The view V is a set of continuous variables, V^b is a discretized version of V in which each variable is binned with bin size b, and $\mathbf{x}^b$ is a tuple from V^b. We have $\mathbb{E}_{\mathbf{x^b}}\left[\delta(\{\mathbf{x^b}\})\right] \to \sigma(V)$ as $b \to 0$.*

Proof. Let the D-dimensional random vector $\mathbf{X}$ describe the (continuous) variables of V, and $\mathbf{X}^b$ describe the (discrete) variables of V^b. By generalizing Cover and Thomas, Theorem 8.3.1 [3], we infer that $H(\mathbf{X}^b) + D \cdot \log b \to H(\mathbf{X})$ as $b \to 0$. Thus, using Equation 5, we have $I(\mathbf{X}^b, \mathcal{T}) \to I(\mathbf{X}, \mathcal{T})$. We conclude that $\sigma(V^b) \to \sigma(V)$. We apply Equation 9 to obtain the lemma. □

4. APPROXIMATE VIEW STRENGTH

We now have a functional definition of view strength. At this point, we could easily envision a greedy heuristic to detect the top K views in a database. We start with simple views, based on one dimension. We then add columns, one by one. To test if a column X is worth adding to a view V, we compute the strength $\sigma(V \cup \{X\})$. If the result is high enough, we keep the candidate. If not, we discard it. We will present such an algorithm in Section 5. However, we must first discuss how to compute $\sigma(V \cup \{X\})$.

Equations 6 and 9 describe several methods to compute the strength of a view. Nevertheless, none of these fit iterative algorithms. Suppose that we wish to compute the strength of a view V, then the strength of another view $V \cup \{X\}$. These equations give us no opportunity to share computations, we must obtain $\sigma(V)$ and $\sigma(V \cup \{X\})$ separately. Furthermore, both expressions are expensive, as they involve group-by queries over the whole database. Therefore, we need an alternative *recursive* formulation for the strength of a view:

Lemma 3. *Consider a view $V = \{X_1, \ldots, X_i\}$, and a target T. For any column X_{i+1}:*

$$\sigma(V \cup \{X_{i+1}\}) = \sigma(V) + I(\mathcal{X}_{i+1}; \mathcal{T}|\mathcal{X}_1, \ldots, \mathcal{X}_i) \qquad (10)$$

Proof. This lemma is a consequence of the Mutual Information's chain rule, Cover and Thomas, Theorem 2.5.2 [3]. □

This lemma describes how adding a column impacts the strength of a view. For any random variables $\mathcal{X}_i, \mathcal{X}_j, \mathcal{T}$, the notation $I(\mathcal{X}_j; \mathcal{T}|\mathcal{X}_i)$ expresses the *conditional mutual information*. The conditional mutual information is a conditioned version of the mutual information: it describes the dependency between $\mathcal{X}_j$ and $\mathcal{T}$ *given restrictions on* $\mathcal{X}_i$. To obtain it, we compute the mutual information between $\mathcal{X}_j$ and $\mathcal{T}$ given all the possible values of $\mathcal{X}_i$, and average the results. Formally:

$$I(\mathcal{X}_j; \mathcal{T}|\mathcal{X}_i) = \mathbb{E}_{x_i}\left[I(\mathcal{X}_j; \mathcal{T})|\mathcal{X}_i = x_i)\right] \qquad (11)$$

The influence of $\mathcal{X}_i$ can go either way: it can weaken the dependency between $\mathcal{X}_j$ and $\mathcal{T}$, or it can strengthen it. The conditional mutual information is positive or null, and it is bounded by the entropy of $\mathcal{X}_j$ and T.

Unfortunately, we cannot directly exploit Lemma 3 in our algorithm: estimating $I(\mathcal{X}_{i+1}; \mathcal{T}|\mathcal{X}_1, \ldots, \mathcal{X}_i)$ for every candidate $\{X_1, \ldots, X_{i+1}\}$ is as expensive as computing the strength directly. However, we can use an approximation. Recall that $V = \{X_1, \ldots, X_i\}$, we exploit the following observation:

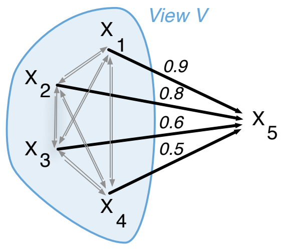

Figure 3: Example of co-dependency graph with 5 dimensions. To approximate the strength of $V \cup \{X_5\}$, we add the weight of edge (X_4, X_5) to V's strength - in this case 0.5.

$$\begin{aligned}\sigma(V \cup \{X_{i+1}\}) &= \sigma(V) + I(\mathcal{X}_{i+1}; \mathcal{T}|\mathcal{X}_1, \ldots, \mathcal{X}_i) \\ &\approx \sigma(V) + I(\mathcal{X}_{i+1}; \mathcal{T}|\mathcal{X}_i)\end{aligned} \qquad (12)$$

The idea behind this approximation is naive: we assume that $I(\mathcal{X}_{i+1}; \mathcal{T}|\mathcal{X}_1, \ldots, \mathcal{X}_i) \approx I(\mathcal{X}_{i+1}; \mathcal{T}|\mathcal{X}_i)$. We simply ignore the high order dependencies. Thanks to this assumption, we can compute the strength of our candidates much faster.

Our new approach operates in two steps, an offline step and an online step. Offline, we compute the conditional mutual information $I(\mathcal{X}_j; \mathcal{T}|\mathcal{X}_i)$ between every pair of variable $(\mathcal{X}_i, \mathcal{X}_j)$. We call the resulting structure **co-dependency graph**. In this graph, the vertices represent the dimensions, and the edges represent the conditional mutual information. The co-dependency graph is oriented and weighted. Online, we run a greedy algorithm as described previously, but we use Equation 12 to evaluate new candidates. To compute the strength of a view $V \cup \{X_{i+1}\}$ with $V = \{X_1, \ldots, X_i\}$, we fetch the value of $I(\mathcal{X}_{i+1}; \mathcal{T}|\mathcal{X}_i)$ in the co-dependency graph and add it to V's strength. We illustrate this method in Figure 3. Previously, computing $\sigma(V \cup \{X_{i+1}\})$ involved heavy groupings and aggregations on the whole dataset. Now, we simply perform a lookup in a graph with N edges, where N is the number of columns in the database.

Note that our approximation has a drawback: it depends on the order in which we include the variables in the view. If we enrich a view by successively adding variables X_1, X_2 and X_3, then we obtain a different strength than if we incorporate X_3, X_2 then X_1. Similarly, in Equation 12, we obtain different approximations if we change the indexing of the dimensions $\mathcal{X}_1, \ldots, \mathcal{X}_i$. For more robustness, we introduce a "pessimistic" variant:

$$\sigma(V \cup \{X_{i+1}\}) \approx \min_{n \in [1,i]} \sigma(V) + I(\mathcal{X}_{i+1}; \mathcal{T}|\mathcal{X}_n) \qquad (13)$$

Instead of adding the strength $I(\mathcal{X}_{i+1}; \mathcal{T}|\mathcal{X}_i)$, where $\mathcal{X}_i$ is the last variable inserted, we add $I(\mathcal{X}_{i+1}; T|\mathcal{X}_n)$, where $\mathcal{X}_n$ is the variable which weakens $\mathcal{X}_{i+1}$ the most. We will use this version in the rest of the paper.

5. PRACTICAL COLUMN SELECTION

This section presents our view search strategy. Our aim is to find the top K views with at most D columns. If our database includes N dimensions, our search space contains $\sum_{n \leq N} \binom{N}{n} = 2^N$ combinations, which is clearly impractical. Therefore, we resort to a greedy, level-wise heuristic.

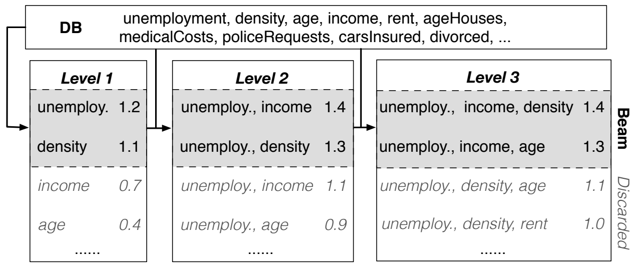
DB
unemployment, density, age, income, rent, ageHouses, medicalCosts, policeRequests, carsInsured, divorced, ...
Level 1
unemploy. 1.2
density 1.1
income 0.7
age 0.4
Level 2
unemploy., income 1.4
unemploy., density 1.3
unemploy., income 1.1
unemploy., age 0.9
Level 3
unemploy., income, density 1.4
unemploy., income, age 1.3
unemploy., density, age 1.1
unemploy., density, rent 1.0
Beam
Discarded

Figure 4: Example of Beam Search, with $D = 3$ and beam size $B = 2$

Algorithm 1 Beam Search for view selection

```
function TopViews($K$, $D$, $B$, $DB$)
    Beam $\leftarrow \{\}$
    for $i \in [1, D]$ do
        Cand $\leftarrow \{\}$, Scores $\leftarrow \{\}$
        for $V \in$ Beam do
            for $X \in$ columns($DB$) do
                Cand $\leftarrow$ Cand $\cup \{V \cup \{X\}\}$
                Scores $\leftarrow$ Scores $\cup\, \sigma(V \cup \{X\})$
            end for
        end for
        Beam $\leftarrow$ findTopK(Cand, Scores, $B$)
    end for
    return findTopK(Cand, Scores, $K$)
end function
```

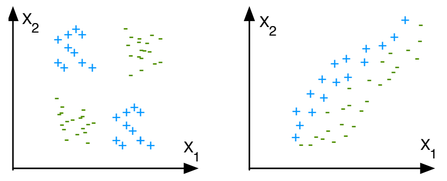

Figure 5: Limit cases of the beam search strategy. The variables X_1 and X_2 represent two dimensions. The symbol and color of the plots represent the value of the target.

5.1 Base algorithm

Our algorithm is based on *beam search*, illustrated in Figure 4. To initialize the algorithm, we compute the strength of each variable separately. We sort the candidates, and keep the top B elements. We call this set the *beam*, greyed in the figure. Then, we generate new candidates, by appending each variable of the database to each variable in the beam. We obtain views with two columns. We compute the strength of these views, keep the top B strongest and discard the others. This gives us a new beam. We repeat the procedure until the views in the beam contain D variables, or the views stop improving. Algorithm 1 presents the full procedure.

Thanks to our strategy, we avoid exploring an exponentially large search space. Instead, we compute the strengths of at most $N.B$ candidates at each level. The size of the beam lets us control the trade-off between accuracy and runtime. With a small beam, we evaluate less candidates, and thus terminate earlier. Oppositely, a large beam lets us explore more candidates. Let us explain why this is necessary.

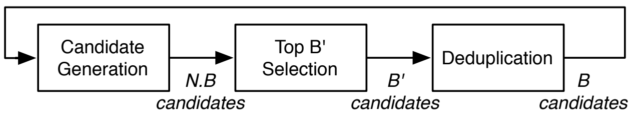

Figure 6: Beam search augmented with a deduplication step. We display in italic the size of the intermediate results. Note that $N \cdot B \geq B' \geq B$.

At each level of the algorithm, we discard the views which are too weak to reach the top B candidates. We assume that if a combination of columns is weak at level i, then it will be weak at all subsequent levels. Unfortunately, this assumption does *not* hold: we can form strong views by combining weak columns; there are “jumps” in the search space. Consider for instance the two classic scenarios pictured in Figure 5. The dimensions X_1 and X_2 taken in isolation are weak: we can infer no useful information about the target from either of them. However, their combination is very interesting. Equivalently, the views $\{X_1\}$ and $\{X_2\}$ have a very poor strength, but $\{X_1, X_2\}$ is an excellent candidate. If the beam is too tight, we may discard $\{X_1\}$ and $\{X_2\}$ early because of low scores. We lose the opportunity to discover $\{X_1, X_2\}$. Therefore, we recommend to set $B > K$. During our experiments, we obtained excellent results with $B \geq 2 \cdot K$ (cf. Section 7.3).

5.2 Approximations and Refinements

In total, we evaluate the strength of $B \cdot N$ candidates for the D levels of the beam search. To carry out this computation, we can either use the exact formulation of strength, as shown in Equation 6, or use the approximation scheme presented in Section 4. In Claude’s implementation, we opted for a hybrid approach. We perform the first two levels of search with the exact strength (which is equivalent to building the co-dependency graph). Then, for all subsequent steps, we use the approximations. Finally, we revert to the exact strength for the top k ranking, at the very end of the procedure. Thanks to this method, we obtain significant speed gains at little accuracy cost.

5.3 Deduplication

Our algorithm seeks strong views. In some cases however, it may be preferable to have weaker but more diverse views. To deal with those cases, we introduce an optional *deduplication* step, during which we reduce the number of views with an algorithm from the literature. As pictured in Figure 6, we run this procedure at the end of each beam search iteration. By definition, deduplication reduces the number of candidates. Therefore, to obtain B views at the end of the algorithm, we must generate $B' > B$ views beforehand. A low B' yields more variety, while a high B' may lead to stronger views.

Authors have proposed quantity of methods to deduplicate itemsets in the pattern mining literature [25, 21]. We opted for a simple compression-based approach. First, we compute the dissimilarity between every pair of views with the Jaccard dissimilarity. Given two views V_i and V_j, it is defined as follows: $d_J(V_i, V_j) = |V_i \cap V_j|/|V_i \cup V_j|$. We then cluster the resulting matrix with Partitioning Around Medoids, an algorithm of type k-medoids. We refer the interested reader to the literature for more details [11].

Dataset	Columns	Rows	#Views	#Variables
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 1: Characteristics of the datasets. The last two columns are used for comparison with 4S, cf. Section 7.3.

6. DETECTING POINTS OF INTEREST

We previously described how to find strong views. We now explain how to identify P points of interests for each of these views.

We instantiate POIs by a well-known analysis called *subgroup discovery* [13, 23], which can be formulated as follows: given a set of tuples, a target column and a measure of exceptionality, detect sets of tuples for which the target behaves exceptionally. In our case, we instantiate the exceptionality measure with divergence.

As pointed our by van Leeuwen and Knobbe [20], we can also solve the subgroup discovery problem with beam search. Let V represent the view to analyze. As the variables are binned, can form a grid over V, as shown in Figure 2. We denote by b the number of bins for each variable. To initialize the algorithm, we compute the divergence of each cell and keep the top B_{POI} most divergent. We obtain our beam. We then "drill" into each of these cells: we decompose them into smaller cells by splitting the edges into b bins. We evaluate the new candidates and keep the top B_{POI} most divergent. We reiterate until the algorithm converges. As shown in the subgroup discovery literature [23, 20], we can generalize this method to binary and nominal data. For each distinct level x_i of a variable X, we create two groups: tuples for which $X = x_i$, and tuples for which $X \neq x_i$.

In practice, KL-based approaches tends to favor smaller regions. Therefore, Beam Search may converge late, or not at all. A practical solution is to set a minimum count threshold. Alternatively, we can alter our model to take the size into account [20]. Let R represents a region with count $|R|$, and $|DB|$ represent the number of tuples in the database. We introduce the *weighted* deviation $\delta_w(R) = |R|/|DB| \times \delta(R)$. This new score introduces a penalty for small POIs.

7. EXPERIMENTS

We now present our experimental results. All our experiments are based on 8 datasets from the UCI Repository, described in Table 1. The files are freely available online[1]. In several experiments, we report the *normalized* view strength instead of the usual strength; if V is a view with entropy $H(V)$, we obtain it as follows: $\sigma_{norm}(V) = \sigma(V)/H(V)$.

7.1 Detailed Example: Crimes in the US

In this section, we showcase Claude with a real-life example: we analyze the Communities and Crime dataset form the UCI repository[2]. Our aim is to understand which US cities are subject to violent crimes. Our database compiles crime data and socio-economic indicators about 1994 communities, with a total of 128 variables. The data comes mostly from the 90's, and it was provided by official US sources - among others, the 1990 US census and the 1995 FBI Uniform Crime Report. All the variables are normalized to have a minimum of 0 and a maximum of 100.

We generated $K = 100$ views with up to $D = 3$ dimensions, both with and without deduplication. We present a selection of views in Table 2, along with 2-dimension heat maps in Figure 7. Observe that strong views have a visual signature: in the top two maps, the blue and red areas are neatly separated. In the bottom two views, the distinction is less clear.

[1]archive.ics.uci.edu/ml/
[2]archive.ics.uci.edu/ml/datasets/Communities+and+Crime

View	Score (normalized)
Police.Overtime, Pct.Vacant.Boarded, Pct.Race.White	0.51
Pct.Families.2.Parents, Pct.Race.White, Police.Requests.Per.Officer	0.49
Pct.Police.White, Pct.Police.Minority, Pct.Vacant.House.Boarded	0.37
Pct.Empl.Profes.Services, Pct.Empl.Manual, Pct.Police.On.Patrol	0.37
Pct.Retired, Pct.Use.Public.Transports, Pct.Police.On.Patrol	0.35
Pct.Recently.Moved, Population.Density, Police.Cars	0.34

Table 2: Example of views generated by Claude for the US Crime dataset.

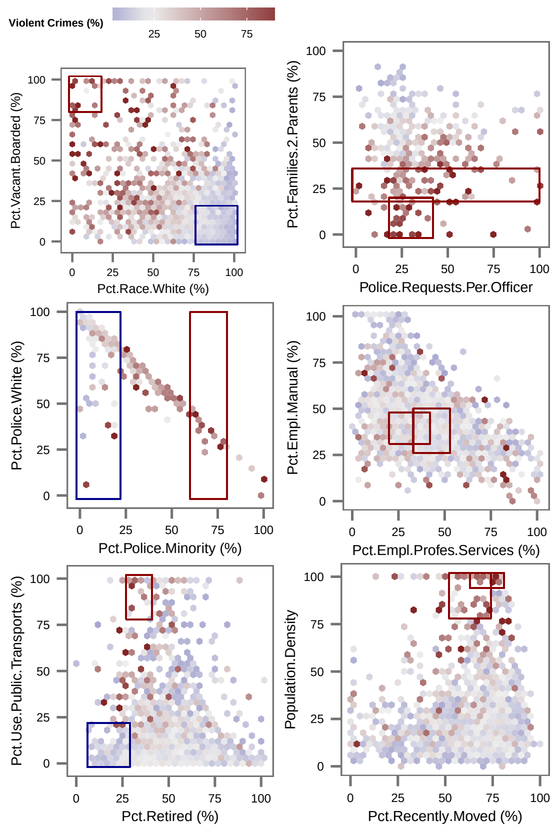

Figure 7: Heatmaps of the US Crime Dataset, based on Claude's output. Each box represents a Point of Interest.

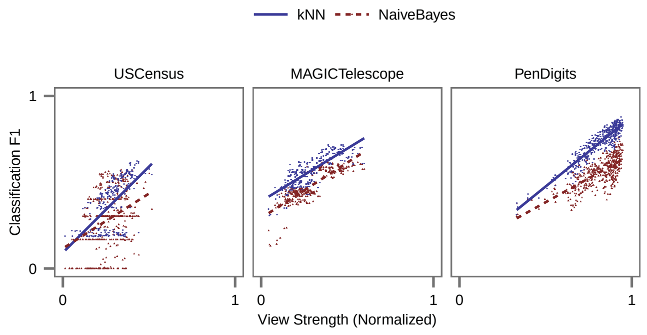

Figure 8: Strength vs. Classification accuracy for 500 random views. We obtained the blue and red lines with simple linear regression.

The first view of Table 2 is the best one we found: `Police.Overtime, Pct.Race.White, Pct.Vacant.Boarded`. It has a score of 0.51. which means that these three variables contain 51% of the target's information. The columns `Police.Overtime` and `Pct.White.Race` respectively describe the average time overworked by the police and the percentage of caucasian population. The third variable, `Pct.Vacant.Boarded` was surprising to us: it describes the percentage of vacant houses which are boarded up. How does this relate to crime? We could assume that boarded houses are associated with long term abandon, and thus, poverty. The top-left plot of Figure 7 shows the relation between race, boarded houses and crime. Observe that the variables complement each other: a high proportion of caucasians may or may not lead to low crime. However, a high proportion of caucasians *combined with* a low rate of boarded house correspond to safe areas, while little caucasians and many boarded houses correspond to more violent communities.

Our second view shows that cities with more monoparental families tend to be more violent: the correlation is clearly visible, and both POIs point to the bottom of the chart. However, close inspection also reveals surprises: a few communities have a relatively high number of two-parents families, but also high indicators of police requests and crime (in the top right corner of the chart). Manual queries reveals that many of these cities are located in the suburbs of Los Angeles, and contain a majority of Hispanics. Does this explain the peculiarity? We leave this question open for future investigations. We see that some findings come from the recommendations directly while others are serendipitous. But in both cases, Claude lets us discover "nuggets" with little prior knowledge and few assumptions.

7.2 View Strength and Prediction

In this section, we show experimentally that our notion of view strength "works", e.g. that strong views effectively provide information about the target column. To verify this assumption, we simulate users with statistical classifiers. Consider a view V over a database. If a classifier can predict the value of the target from V's columns, then V is informative. Oppositely, if the classifier fails, then V is potentially uninteresting. In a nutshell, we should observe a positive correlation between views strength and classification accuracy.

We now detail our experiment. We chose three datasets from the UCI repository. For each dataset, we generated 500 random views and measured their strengths. We then trained classifiers on each of these views, and measured their performance. We report the results in Figure 8. We chose two classification algorithms: Naive Bayes, and 5-Nearest Neighbors. We chose those because they contain no built-in mechanism to filter out irrelevant variables (as opposed to, e.g., decision trees). We measure classification performance with 5-fold validation, to avoid the effects of overfitting.

In all three cases, we observe a positive correlation between the strengths of the views and the accuracy of the predictions. We confirm these observations with statistical tests: the coefficients of determination (R^2) vary between 0.11 and 0.84, which indicates the presence of a trend (despite some variability). Furthermore, the p-values associated to the coefficients are all under 10^{-3}, this gives us excellent confidence that the strength influences positively the prediction accuracy. In conclusion, strong views are indeed more instructive.

7.3 View Selection

We now evaluate Claude's output and runtime in detail. In this section, we verify if Claude's algorithm produces good views in a short amount of time. To do so, we compare it to four methods, three of which come from the machine learning literature. Our first baseline, `Exact`, is similar to Claude, but we removed the approximation scheme presented in 4 - instead we compute the exact the mutual information, as in Equation 6 The method should be slower, but more accurate.

The second algorithm, `Clique`, is a top-down approach inspired by recent work on pattern mining [24]. We build a graph where each vertex i represents a column D_i, and each edge (i, j) represents the view $\{D_i, D_j\}$. We then eliminate all the edges except those which represent the top B views. To detect views with $D > 2$ columns, we seek cliques in this degenerated graph. We used the `igraph` package from R. We expect this algorithm to be very fast, but less accurate.

The third method, `Wrap 5-NN`, is a classic feature selection algorithm [9]. The idea is to train a 5-Nearest Neighbor classifier with increasingly large sets of variables. We first test each variable separately, and keep the column which led to the best prediction. Then we keep adding variables in a breadth-first manner, until the quality of the predictions stops increasing or we reach n variables. Our implementation is based on the `class` package from R. We modified the original algorithm to maintain and update q distinct sets of variables instead of just one. We chose the nearest neighbor algorithm because it is fast, and it gave us good performance, as shown in 7.2. We expect this algorithm to be very slow, but close to optimal.

Finally, the last method, `4S` is a state-of-the-art subspace search method from the unsupervised learning literature [14]. The aim of the algorithm is to detect "interesting" subspaces in large databases, independently of a target variable. To do so, it seeks groups of variables which are mutually correlated, with sketches and graph-based techniques. We used the author's implementation, written in Java. We expect the algorithm to be very fast and reasonably accurate.

We use 8 public datasets, presented in Section 7. For a fair comparison, we must ensure that each algorithm generates the same number of views (K) with the same number of variables (D). However, we have no way to specify these parameters a priori with 4S, because the algorithm has a

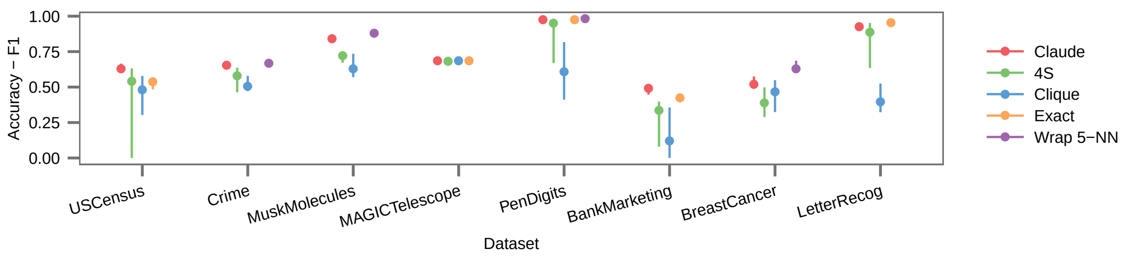

Figure 9: Performance of the View Selection algorithms. For each data set, we generate q views, train a 5-NN classifier over the columns of each view and report the classification accuracy (F1, 5-fold cross validation). The points represent median scores, the bars represent the lowest and greatest scores.

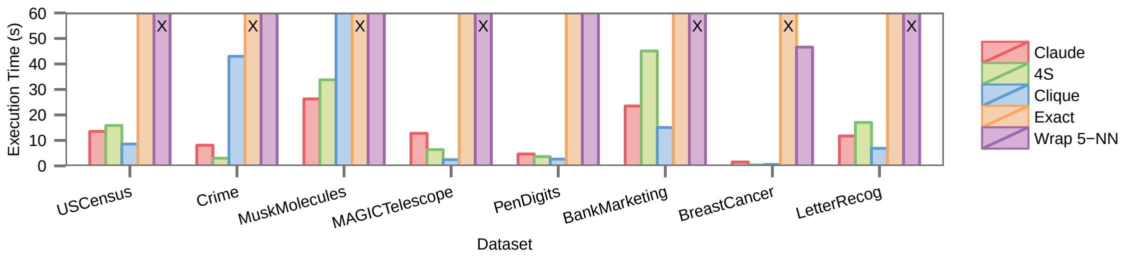

Figure 10: Execution time of the View Selection algorithms. A `X` symbol indicates that the experiment did not finish within 3,600 seconds.

built-in mechanism to pick optimal values. Therefore, we run 4S first on each dataset, we let it chose K and D, and we use these values for the remaining algorithms. We report the obtained parameters in Table 1.

We implemented Claude in R, except for some information theory primitives written in C. For practical reasons, 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. The operating system is Fedora 16.

Accuracy. In Figure 9, we compare the quality of the views returned by each algorithm. For each competitor, we generate q views with n variables, train a classifier on each view and measure the quality of the predictions. For the classification, we use both Naive Bayes and 5-Nearest Neighbors, and report the highest score. We measure accuracy with the F1 score on 5-fold cross validation; higher is better.

The method `Wrap 5-NN` comes first for all the datasets on which it completed. This is not surprising since the algorithm optimizes exactly what we measure: `Wrap 5-NN` is our "gold standard". Our two algorithms, `Claude` and `Exhaustive`, come very close. This indicates that both algorithms find good views, and that our approximation scheme works correctly. The algorithms `4S` and `Clique` come much lower. As `4S` is completely unsupervised, we cannot expect it to perform as well as the other approaches. The assumptions behind `Clique` are apparently too naive.

Runtime. Figure 10 shows the runtime of our experiments. The algorithms `Exact` and `Wrap 5-NN` are orders of magnitude slower than the other approaches. The remaining three approaches are comparable: depending on the datasets, either `Clique` or `4S` come first. Claude comes first for `MuskMolecules`, and close second for all the other datasets. In conclusion, Claude is comparable to its competitors in terms of runtime, but it generates better views.

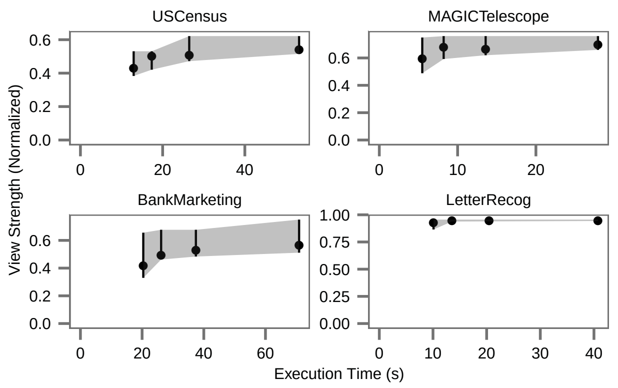

Figure 11: Impact of the beam size on the execution time and view strength. For each dataset, we generated 25 views with beam size 25, 50, 100 and 250. The points represent the medium scores, the bars represent the lowest and greatest scores.

Impact of the beam size. Figure 11 shows the impact of the beam size B on Claude's performance, for 4 databases. To obtain these plots, we ran Claude with $K = 25$ and $D = 5$, and varied B between 25 and 250. We observe that smaller beams lead to lower execution times, while larger beam lead to stronger views. However, the heuristic converges fast: we observe little to no improvement for B greater than 50.

Impact of the deduplication. We show the impact of our deduplication strategy in Figure 12. We ran Claude with $K = 25$ and $D = 5$ and increased the level of deduplication, i.e., varied the value of B' between B and $N.B$ (cf. Section 5.3). A level of 0% means that $B' = B$. A level of 100%

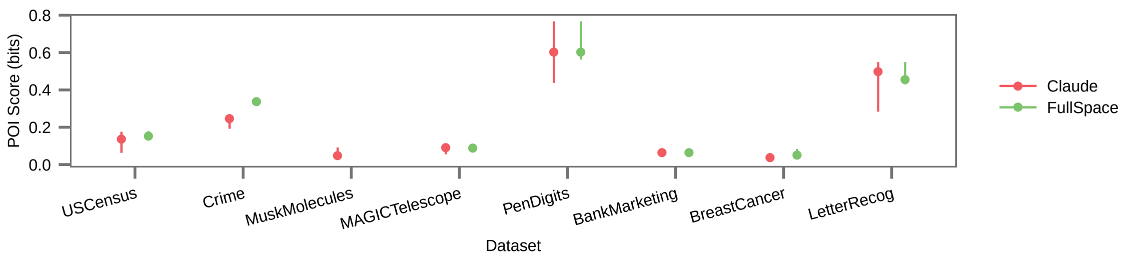

Figure 13: Quality of the Point of Interests. For each view, we detect $P = 10$ POIs. The points represent median scores, the bars represent the lowest and greatest scores.

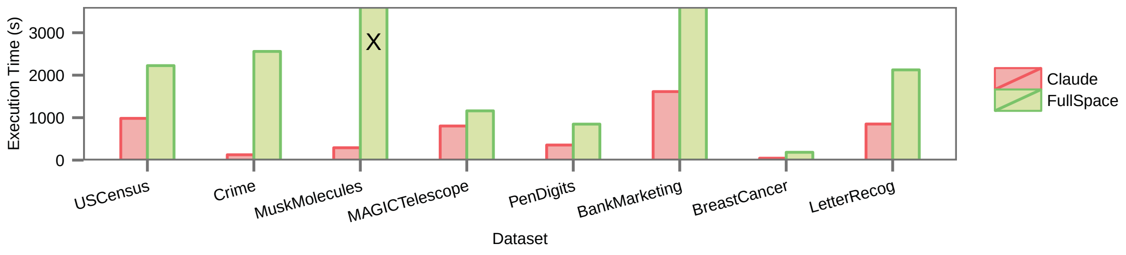

Figure 14: Execution time of the POI detection. A `X` symbol indicates that the experiment did not finish within 7,200 seconds.

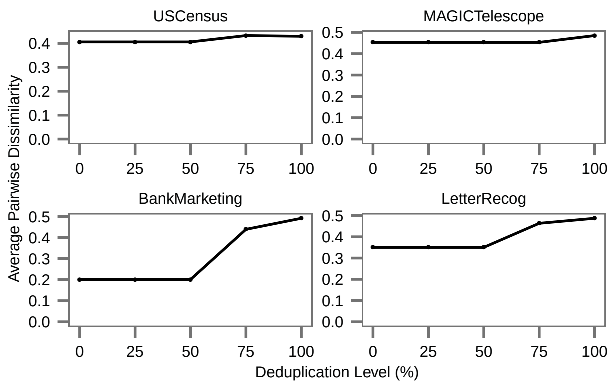

Figure 12: Impact of the deduplication. We generated 25 views for each dataset. The y-axis presents the average Jaccard dissimilarity between every pair of views.

means that $B' = N.B$. To measure the diversity of the views, we measured the Jaccard dissimilarity between every pair of views and averaged the results. We observe that the strategy works in all four cases, but with different levels of efficiency. In the `BankMarketing` case, our strategy almost doubles the pairwise dissimilarity of the views. The effect is much lighter on datasets with few columns, such as `USCensus` and `MAGICTelescope`.

7.4 POI Detection

In this section, we evaluate Claude's POI detection strategy. We compare two approaches. The first approach is the algorithm presented in the paper: first we search K views, then we return P POIs per view. The second approach, `FullSpace`, is the method used in much of the recent Subgroup Discovery literature [20, 7]. The idea is to apply Beam Search on the whole database directly. Instead of seeking P POIs in K projections, we seek $K.P$ selections from the full column space; we skip the view selection step. We use the same datasets as previously. Our default parameters are $K = 25$, $D = 5$, $B = 50$ and $P = 10$. To set the beam size, we use a rule of thumb: $B_{POI} = 2.k$ (B_{POI} is the beam used for POI detection, not for view search). To gather sufficient data, we raise our time limit to 2 hours.

Figure 13 compares the quality of the POIs found by both algorithms. The strategy `FullSpace` gives slightly better results on `Crime` and `PenDigits`, but the difference is close to null. The scores are similar on all the other datasets. We conclude that Claude's POIs are very close to those found by a state-of-the-art Subgroup Discovery approach. Figure 14 compares the runtimes of both approaches. We observe that Claude is much faster than `FullSpace`. The difference grows with the number of columns: the runtimes are almost similar for datasets with few columns (`MAGICTelescope`), but `Claude` is considerably faster for larger databases (more than an order of magnitude difference for `MuskMolecules`). This is a positive side-effect of our approach: decoupling view search and POI extraction allows us to find subgroups faster in high dimension datasets.

8. RELATED WORK

SQL Query Recommendation. We identify two types of approaches: *human-driven* systems and *data-driven* systems. Human-driven systems learn from user feedback. For instance, Chatzopoulou et al. make recommendations from query logs, similarly to search engines [2]. In Explore-by-Example, the system infers queries from examples provided by the user [5]. With Charles, the engine decomposes user queries into smaller queries, which can then be decomposed further [18]. Sarawagi's method builds a maximum entropy model over the database from the user's history [16]. Bonifati et al. propose a similar method to recommend joins [1]. Claude competes with neither of these approaches, since it uses the content of the database only.

Our work is closer data-driven data recommendation. The general idea is to build a statistical model of the database, and find regions which behave unexpectedly. Sarawagi et al.

have published seminal work on this topic [17]. Their system requires that the data is organized in an OLAP cube (with hierarchical dimensions), it supposes that the users know which variables to use, and it seeks thin-grained deviations. Oppositely, our system uses regular tables, it recommends views (not only selections) and it seeks large trends. Similarly, constrained gradient analysis [10, 6] focuses only on hierarchical data cubes. More recently, Dash et al. have proposed a method to reveal surprising subsets in a faceted search context [4]. This method is related to Claude, but it targets document search, it does not recommend views.

Projection Search. Authors from the data visualization literature have proposed methods to detect the "best" projections of multidimensional data sets, such as Projection Pursuit [8], Scagnostics [22], or Tatu et al.'s relevance measures [19]. Such methods would form excellent complements for Claude's recommendations. Nevertheless, most of them focus on 2-dimensional scatterplots, are limited to continuous variables, and involve materializing and analyzing every possible 2D projection of the data.

Feature Selection, Subspace Search. Choosing which variables to use for classification or regression is a crucial problem, for which dozens of methods were proposed [9]. Similarly to Claude, some of these methods rely on mutual information [15]. Nevertheless, the objective is different. A feature selection algorithm seeks *one* set of variables, on which a statistical predictor will perform optimally. Claude seeks several, small sets of variables, simple enough to be interpreted by a humans. In fact, Claude is halfway between inference and exploration. On the unsupervised learning side, our work is close to subspace search. The idea is detect subspaces where the data is clustered distinctly [12, 14]. We compare Claude to state-of-the-art methods in our Experiments section.

9. CONCLUSION

We formalized what makes a query "interesting", using the mutual information and the Kullback-Leibler divergence. We presented practical methods to detect these queries, using carefully designed approximations. Finally, we presented and evaluated Claude, a system based on these ideas. The methods we developed for this study have broader applications than the strict realm of query recommendation. Our column selection scheme competes with state-of-the-art feature selection methods. Also, the idea to decouple column selection selection from subgroup search could benefit a wide range of subgroup discovery algorithms.

We are genuinely excited about the possible extensions of this study. Future work includes dealing explicitly with the structure of the data, e.g., hierarchical dimensions and relational joins. We will study how to integrate Claude more tightly with visualizations, for a fully interactive experience. Finally, we will refine our framework with causation models.

10. ACKNOWLEDGEMENTS

This publication was supported by the Dutch national program COMMIT.

11. REFERENCES

[1] A. Bonifati, R. Ciucanu, and S. Staworko. Interactive inference of join queries. In *EDBT*, pages 451–462, 2014.

[2] G. Chatzopoulou, M. Eirinaki, and N. Polyzotis. Query recommendations for interactive database exploration. In *SSDBM*, pages 3–18, 2009.

[3] T. M. Cover and J. A. Thomas. *Elements of information theory*. John Wiley & Sons, 2012.

[4] D. Dash, J. Rao, N. Megiddo, A. Ailamaki, and G. Lohman. Dynamic faceted search for discovery-driven analysis. In *ACM CIKM*, pages 3–12, 2008.

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

[6] G. Dong, J. Han, J. Lam, J. Pei, and K. Wang. Mining multi-dimensional constrained gradients in data cubes. In *VLDB*, pages 321–330, 2001.

[7] W. Duivesteijn, A. Knobbe, A. Feelders, and M. van Leeuwen. Subgroup discovery meets bayesian networks–an exceptional model mining approach. In *IEEE ICDM*, pages 158–167, 2010.

[8] J. Friedman and J. Tukey. A projection pursuit algorithm for exploratory data analysis. In *IEEE TC*, page 149, 1974.

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

[10] T. Imieliński, L. Khachiyan, and A. Abdulghani. Cubegrades: Generalizing association rules. *Data Mining and Knowledge Discovery*, pages 219–257, 2002.

[11] L. Kaufman and P. J. Rousseeuw. *Finding groups in data: an introduction to cluster analysis*. John Wiley & Sons, 2009.

[12] F. Keller, E. Muller, and K. Bohm. Hics: high contrast subspaces for density-based outlier ranking. In *ICDE*, pages 1037–1048, 2012.

[13] W. Klösgen. Explora: A multipattern and multistrategy discovery assistant. *AAAI Advances in Knowledge Discovery and Data Mining*, pages 249–271, 1996.

[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] H. Peng, F. Long, and C. Ding. Feature selection based on mutual information criteria of max-dependency, max-relevance, and min-redundancy. In *IEEE PAMI*, pages 1226–1238, 2005.

[16] S. Sarawagi. User-adaptive exploration of multidimensional data. In *VLDB*, pages 307–316, 2000.

[17] S. Sarawagi, R. Agrawal, and N. Megiddo. Discovery-driven exploration of olap data cubes. In *EDBT*, 1998.

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

[19] A. Tatu, G. Albuquerque, M. Eisemann, P. Bak, H. Theisel, M. Magnor, and D. Keim. Automated analytical methods to support visual exploration of high-dimensional data. In *IEEE TVCG*, pages 584–597, 2011.

[20] M. van Leeuwen and A. Knobbe. Non-redundant subgroup discovery in large and complex data. In *ECML PKDD*, pages 459–474. 2011.

[21] M. van Leeuwen, J. Vreeken, and A. Siebes. Compression picks item sets that matter. In *PKDD*, pages 585–592. 2006.

[22] L. Wilkinson, A. Anand, and R. L. Grossman. Graph-theoretic scagnostics. In *IEEE Infovis*, page 21, 2005.

[23] S. Wrobel. An algorithm for multi-relational discovery of subgroups. In *PKDD*, pages 78–87. 1997.

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

[25] D. Xin, J. Han, X. Yan, and H. Cheng. Mining compressed frequent-pattern sets. In *VLDB*, pages 709–720, 2005.