Fast attribute-based table clustering using Predicate-Trees: A vertical data mining approach

Arjun G. Roy∗, Arijit Chatterjee, Mohammad K. Hossain and William Perrizo
Department of Computer Science and Operations Research, North Dakota State University, Fargo, ND, USA

Abstract. With technological advancements, massive amount of data is being collected in various domains. For instance, since the advent of digital image technology and remote sensing imagery (RSI), NASA and U.S. Geological Survey through the Landsat Data Continuity Mission, has been capturing images of Earth down to 15 meters resolution. Likewise, consider the Internet, where, growth of social media, blog Web sites, etc. generates exponential amount of textual data on a daily basis. Since clustering of data is time-consuming, much of these data is archived even before proper analysis. In this paper, we propose two novel and extremely fast algorithms called imgFAUST or Fast Attribute-based Unsupervised and Supervised Table Clustering for images and a variation called docFAUST for textual data. Both these algorithms are based on Predicate-Trees which are compressed, lossless and data-mining-ready data structures. Without compromising much on the accuracy, our algorithms are fast and can be effectively used in high-speed image data and document analysis.

Keywords: Data mining, Predicate Trees, vertical data processing, big data

1. Introduction

Digital Image Technology developed in 1970s has caused an exponential growth in the amount of image data throughout the world. This includes, personal photographs, medical images, scientific data such as remote sensing imagery, etc. The Landsat Data Continuity Mission by NASA and U.S. Geological Survey has been providing image captures of Earth with moderate resolution from 15 meters to 100 meters for the last 38 years [10]. Invention of high-definition camera devices, camera embedded phones has led to massive image data generation. Image segmentation or color clustering is important. Consider the current scenario of usage of Unmanned Air Vehicles (UAVs) for security purpose where there is massive data collection and classifying objects of interest such as tanks, enemy hideouts, etc. is of utmost importance. Due to slowness of existing algorithms, much of these data is archived without proper analysis. Thus, there is a sudden demand for fast clustering algorithms to cope up with massive image data collection.

Now, consider the Internet, where growth of World Wide Web generates terabytes of textual data on a daily basis. This is majorly because of two reasons, a) number of Internet users has risen to about 2.3
billion in 2011 from 360 million in 2000 [8] and b) growth of social media, news, free email services and blog Web sites such as Facebook, CNN, Gmail and Blogger, respectively. Blog Web sites such as blogger.com, wordpress.com, etc. allow users to blog with no restriction on file size or number of blogs they can post. Search and advertising company Google last year announced putting millions of public library books online for free [5]. Apart from this, majority of information in the form of technical/non-technical publications, digital libraries, etc. are available online and form the basis of knowledge sharing in present times. Thus, like image data generation and its clustering problems, textual data follows the similar pattern.

Data clustering by definition is partitioning data into classes with high intra-class similarity and low inter-class similarity. This process of data mining is different from classification where the classes are predefined. These clustering algorithms differ from one another on the basis of the underlying model. Some of the major categories include hierarchical techniques such as agglomerative (AGNES) and divisive (DIANA), partitioning techniques such as k-means, bisecting-k-means, k-medoids, etc. Each of these techniques has associated advantages and disadvantages. For example, agglomerative hierarchical technique is shown to prove better results, detect clusters of different shapes, independent of choice of initial seeds and number of clusters. On the other hand, they suffer from high algorithmic complexity [1] gives a comprehensive study on data clustering methodologies.

Document clustering is a part of data clustering which specifically deals with grouping documents based on similar subject/topic. Formally, clustering can be defined by a mapping function M which maps a collection of documents D to classes C1, C2, . . . , etc. In clustering, there is no prior knowledge of the different classes of text documents. The basic idea of clustering is to have documents assigned to a cluster which are similar to one another and dissimilar to documents of other clusters. Majority of research related to text clustering considers the text document as a vector space model [6]. In vector space model, the document is represented as a bucket of words along with their associated weights. The weights signify how important each word is, within the document as well as among all the documents in the document space. The most common approach in assigning the weight to each word is the \((tf \times idf)\) method. In this method, weight of any term ‘t’ is the product its frequency in a document ‘d’ and the frequency of the document in the entire document space. For document similarity, cosine measure is the most commonly used metric. It calculates the angle between the two document vectors in a V-dimensional Euclidean space, where V is the vocabulary size. The similarity function differs from one algorithm to another. In simplest case, it can be count of the common words shared between the documents. In this paper, we propose a semi-supervised algorithm where the domain knowledge of the document space is somewhat known in advance. That is, we take a small chunk of dataset and consider them as the seed points (class label already known) for the algorithm.

Next, let’s shift our focus to the data organization which forms the most important aspect of our algorithms. Traditionally, data is stored horizontally. Some of the most popular databases such Microsoft SQL Server, Oracle family, IBM DB2, etc. store data horizontally and process them vertically. Recently, there has been a slow but increasing focus on the column-oriented database management systems (DBMSs) [2,4,7,11,15]. Column-oriented DBMSs are different from the traditional ones in the way data is stored and accessed. Traditional ones store data row-wise while the column-oriented ones store the data column-wise. This allows column-oriented DBMSs to have extensive usage in various data warehouse applications because of their better performance in terms of read I/O in comparison to the conventional DBMSs. This is primarily due to the fact that column-oriented DBMSs only retrieve the columns defined in the query rather than the entire row in case of traditional DBMSs. Additional significant performance gain is because of compression since columns of similar datatype can be compressed.
much better in comparison of rows which are a collection of attributes of different datatypes and sizes. As a result, in real-life scientific applications, where the number of columns are in the order of hundreds, vertical databases can have significant performance speedup over row-based database systems. In addition, they have several other advantages over conventional row-based databases including better consistency in absence of record-level locking, cheaper insertion and deletion operation, etc. [11].

The organization of the paper is as follows: In Section 1, we begin with the introduction. We discuss the importance of image and document clustering followed by the underlying concept of clustering. We briefly explain the underlying concept of column-oriented database management systems and their advantages over the conventional DBMSs. In Section 2, we explain the Predicate-Trees in detail. Predicate-Trees form the basis of how we process the vertical data, horizontally. As we will see, slicing the data into vertical strips has significant performance benefits when big data is involved. We also include a subsection on Inequality P-trees which are used in our algorithm. In Section 3, we propose our novel algorithm called imgFAUST or Fast Attribute-based Unsupervised and Supervised Table clustering of colors in images. We use P-Trees to classify the pixels as explained in the following sections. These data structures basically convert the attributes comprising the image into vertical strips of binary data. We then perform P-Tree specific operations to achieve extremely fast results. In Section 4, we propose a slight variation algorithm called docFAUST for document clustering. Finally in the next couple of section, we give our insight on how our algorithms are faster than other peer algorithms and conclude with future works.

2. Predicate-Trees

In this section, we give a detailed background information about the Predicate-Trees which form the basis of our algorithms. Predicate-Trees or P-Trees are data-mining-ready, compressed and lossless data structures. Simplest forms are the Peano-Trees which are bitwise trees and can be 1, 2 or n-dimensional depending on the application. For e.g., a spatial image can be efficiently represented by 2-dimensional Peano-Trees. P-Trees have been used in a wide variety of research areas including text mining [6], DNA Microarray data analysis [18], association rule mining [14], etc.

Consider a dataset $X$ with $d$ attributes represented as $X = (A_1, A_2, \ldots, A_d)$ and the binary representation of any $k^{th}$ attribute, $A_k$, be represented as $b_{k,m-1}b_{k,m-2}b_{k,m-3} \cdots b_{k,0}$. Here, $m$ is the number of bits required to represent values in $A_k$. For e.g., 9 can be represented by 1001, so $m = 4$. We then decompose each attribute into bit files, i.e., one file for each bit position [16]. The P-Tree is simply constructed by taking bit files (one at a time) and recursively partitioning them into halves and sub-halves until each sub-half is absolutely pure i.e., entirely 1-bits or 0-bits. Consider Fig. 1 containing 1-attributed data containing values 2, 3, 2, 2, 5, 7 and 7. This is converted to binary resulting in three vertical strips of
Table 1: IRIS data mean

<table>
<thead>
<tr>
<th>SL</th>
<th>m</th>
<th>SW</th>
<th>m</th>
<th>PL</th>
<th>m</th>
<th>PW</th>
<th>m</th>
</tr>
</thead>
<tbody>
<tr>
<td>se</td>
<td>51</td>
<td>ve</td>
<td>32</td>
<td>se</td>
<td>14</td>
<td>se</td>
<td>2</td>
</tr>
<tr>
<td>vi</td>
<td>63</td>
<td>vi</td>
<td>32</td>
<td>ve</td>
<td>47</td>
<td>ve</td>
<td>14</td>
</tr>
<tr>
<td>ve</td>
<td>70</td>
<td>se</td>
<td>35</td>
<td>vi</td>
<td>60</td>
<td>vi</td>
<td>25</td>
</tr>
</tbody>
</table>

Table 2: IRIS data mean gap

<table>
<thead>
<tr>
<th>SL</th>
<th>mg</th>
<th>SW</th>
<th>mg</th>
<th>PL</th>
<th>mg</th>
<th>PW</th>
<th>mg</th>
</tr>
</thead>
<tbody>
<tr>
<td>se</td>
<td>12</td>
<td>ve</td>
<td>0</td>
<td>se</td>
<td>33</td>
<td>se</td>
<td>12</td>
</tr>
<tr>
<td>vi</td>
<td>7</td>
<td>vi</td>
<td>3</td>
<td>ve</td>
<td>13</td>
<td>ve</td>
<td>11</td>
</tr>
<tr>
<td>ve</td>
<td>0</td>
<td>se</td>
<td>0</td>
<td>vi</td>
<td>0</td>
<td>vi</td>
<td>0</td>
</tr>
</tbody>
</table>

data (since each value can be represented by a 3-bit binary number. P0, P1 and P2 are the three P-Trees generated from the above data. The construction process of each P-Tree is independent of other and thus can be parallelized over multicore processors.

The most frequently used operations in P-Tree are the AND, OR and NOT logic operations. These operations are computed on a level by level basis starting from the root node. There are certain rules associated with these P-Tree operations. For example, an AND operation between pure-0 node and any subtree results in a pure-0 node, an OR operation between a pure-1 node and any subtree results in the subtree itself, etc. Here, a P-Tree is said to be pure-1 if its subtree consists of only 1s and called pure-0 otherwise. For more information on P-Tree structure, construction and operations, please refer to [16].

In this paper, we only provide our FAUST algorithms specific details about P-Trees. More specifically, we will now discuss the inequality Predicate-Trees which are used in evaluating the range predictions.

2.1. Inequality P-Trees

The Inequality P-Trees are a class of Predicate trees used in evaluating range predictions. They represent data points within a dataset which satisfy the inequality predicates like $x \geq v$, $x \leq v$, etc. A P-Tree representing $P_{x \geq v}$ is computed as follows: Consider $x$ to be a data point inside a dataset $X$. Let $x$ be an m-bit data and let $P_{m-1}, P_{m-2}, \ldots, P_0$ be the P-Trees representing the dataset $X$. Let $v$ be the value represented by $b_{m-1}, b_{m-2}, \ldots, b_0$ for which we need to find the equality. Then a Predicate Tree representing the predicate $x \geq v$ is given by $P_{x \geq v} = P_{m-1} \land P_{m-2} \land \ldots \land P_0$, where $i$ varies from 1 to m. Also, $\land$ is a binary AND operation if $b_i$ is 1 and a binary OR operation if $b_i$ is 0. The operators are right binding. For e.g., $P_{x \geq 14}$ or $P_{x \geq 1110}$ can be computed as $(P_3 \land (P_2 \land (P_1 \lor P_0)))$. Values less than 14 in the given example can be computed by taking the complement of the Predicate tree.

2.2. Computation of mean using P-Trees

Our algorithm uses computation of mean from the vertical representation of the data. P-Trees can be efficiently used to compute the aggregate functions such as count, sum, mean, max, min, etc. For e.g., count is nothing but the RootCount of the P-Trees. Sum function can be computed using the following algorithm:

```plaintext
total ← 0
for i = n − 1 to 0 do
    total ← total + 2^i × Rootcount(P_i)
end for
```

Here, $n$ is the number of bits representing the attribute or column of the table. The mean can be computed by $\frac{total}{count}$. A detailed research on max, min and other aggregate functions is given in [17].
### Table 3

<table>
<thead>
<tr>
<th>cl</th>
<th>att</th>
<th>m</th>
<th>gap&lt;h&gt;l</th>
<th>gap&lt;h&gt;u</th>
<th>gap&lt;sub&gt;rel&lt;/sub&gt;</th>
</tr>
</thead>
<tbody>
<tr>
<td>se</td>
<td>SL</td>
<td>51</td>
<td>12</td>
<td>12</td>
<td>0.235</td>
</tr>
<tr>
<td>se</td>
<td>SW</td>
<td>35</td>
<td>2</td>
<td>2</td>
<td>0.057</td>
</tr>
<tr>
<td>se</td>
<td>PL</td>
<td>14</td>
<td>33</td>
<td>33</td>
<td>2.357</td>
</tr>
<tr>
<td>se</td>
<td>PW</td>
<td>2</td>
<td>12</td>
<td>12</td>
<td>6</td>
</tr>
<tr>
<td>vi</td>
<td>SL</td>
<td>63</td>
<td>12</td>
<td>7</td>
<td>0.15</td>
</tr>
<tr>
<td>vi</td>
<td>SW</td>
<td>33</td>
<td>1</td>
<td>2</td>
<td>0.045</td>
</tr>
<tr>
<td>vi</td>
<td>PL</td>
<td>60</td>
<td>13</td>
<td>13</td>
<td>0.217</td>
</tr>
<tr>
<td>vi</td>
<td>PW</td>
<td>25</td>
<td>11</td>
<td>11</td>
<td>0.44</td>
</tr>
<tr>
<td>ve</td>
<td>SL</td>
<td>70</td>
<td>7</td>
<td>7</td>
<td>0.1</td>
</tr>
<tr>
<td>ve</td>
<td>SW</td>
<td>32</td>
<td>1</td>
<td>1</td>
<td>0.031</td>
</tr>
<tr>
<td>ve</td>
<td>PL</td>
<td>47</td>
<td>33</td>
<td>13</td>
<td>0.489</td>
</tr>
<tr>
<td>ve</td>
<td>PW</td>
<td>14</td>
<td>12</td>
<td>11</td>
<td>0.821</td>
</tr>
</tbody>
</table>

Initially, let \( P_{\text{ REMAINING}} \) be a pure-1 P-Tree. From the training set:

1. For each attribute, calculate the mean for each class, sort ascending according to the mean.
   - Calculate all the mean gaps = difference of consecutive means.

   Create MeanTable(Class, Attribute, Mean, gap<h>l, gap<h>u, gap<sub>rel</sub>) and sort descending on gap<sub>rel</sub> = (gap<h>u - gap<h>l)/2*mean.
   - Here, gap<h>l, gap<h>u, and gap<sub>rel</sub> are the gaps on the low and high side of the mean.

2. Choose and remove a MeanTable record with maximum gap<sub>rel</sub>.
   - Calculate \( C_L = \text{Mean} \cdot \text{gap}<h>/2 \) to produce \( P_L = P_A > C_L \).
   - Calculate \( C_U = \text{Mean} + \text{gap}<h>/2 \) to produce \( P_U = P_A > C_U \).

3. Assign class mask \( P_{\text{CLASS}} = P_L \lor P_R \lor P_{\text{REMAINING}} \).

4. Update \( P_{\text{REMAINING}} = P_{\text{REMAINING}} \land \neg \text{P}_{\text{CLASS}} \).

5. Repeat steps 2 to 4 until all the classes have a P-Tree.

6. Repeat steps 1 to 5 until means stop changing (much).

Fig. 2. imgFAUST algorithm.

### 3. ImgFAUST algorithm

In this section, we propose our algorithm called the imgFAUST or Fast Attribute based Unsupervised and Supervised Table Clustering for images and explain it using the IRIS dataset. The dataset is freely available from the UCI Machine Learning Laboratory [9]. The pseudocode of the algorithm is as follows in Fig. 2.

The algorithm functions on IRIS data in the following way. We first remove the decimal point from all the data points to use integer values. We then choose 10 random samples from each of the class, i.e., Iris-setosa, Iris-versicolor and Iris-virginica. Because of space limitation, we do not show the 30 points (10 points per class) but their means (m) in ascending order are listed in Table 1.

The successive mean gap (mg) is calculated for each class and all the attributes.

In Table 2, each integer represents the mean gap between the row it corresponds to and the successive row. Hence, the last row is populated with 0. SL, SW, PL and PW in the table are the four attributes of the IRIS dataset representing sepal length, sepal width, petal length and petal width. Iris-setosa, Iris-
Table 4

<table>
<thead>
<tr>
<th>Document Vs Words Matrix</th>
<th>Mountain</th>
<th>Beauty</th>
<th>Adrenaline</th>
</tr>
</thead>
<tbody>
<tr>
<td>Travel Guide 1</td>
<td>WF</td>
<td>WF</td>
<td>WF</td>
</tr>
<tr>
<td>Sports Magazine 1</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Sports Magazine 2</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
<tr>
<td>Travel Guide 2</td>
<td>...</td>
<td>...</td>
<td>...</td>
</tr>
</tbody>
</table>

Table 5

<table>
<thead>
<tr>
<th>docFAUST Mean Table</th>
<th>cl</th>
<th>att</th>
<th>m</th>
<th>mg</th>
</tr>
</thead>
<tbody>
<tr>
<td>se</td>
<td>pl</td>
<td>14</td>
<td>33</td>
<td></td>
</tr>
<tr>
<td>ve</td>
<td>pl</td>
<td>47</td>
<td>13</td>
<td></td>
</tr>
<tr>
<td>se</td>
<td>pw</td>
<td>2</td>
<td>12</td>
<td></td>
</tr>
<tr>
<td>ve</td>
<td>pw</td>
<td>14</td>
<td>11</td>
<td></td>
</tr>
<tr>
<td>vi</td>
<td>sl</td>
<td>63</td>
<td>7</td>
<td></td>
</tr>
<tr>
<td>vi</td>
<td>sw</td>
<td>32</td>
<td>3</td>
<td></td>
</tr>
</tbody>
</table>

Let the Current Cluster $CC = \{\text{Class}_1, \text{Class}_2, ..., \text{Class}_k\}$ be represented by a Predicate Tree $P_{CC}$.

Initially, $P_{CC}$ is set to a pure-1 P-Tree.

From the training set,

1. For each shortlisted word, calculate the mean of each class in $CC$ and sort in ascending order of the mean.

2. Calculate all the mean gaps = difference of consecutive means.

3. Choose and remove the MeanTable record with maximum gap.

4. Use $P_{AC} (c = \text{mean} + \text{gap}/2)$ to separate the current cluster into two clusters.

   The cluster masks are:

   $P_{\text{NEWCLUSTER}_1} = P_{\text{AC}} \land P_{\text{CC}}$

   $P_{\text{NEWCLUSTER}_2} = P'_{\text{AC}} \land P_{\text{CC}}$

   $\text{NEWCLUSTER}_1 = \{\text{All classes corresponding to the mean that had the max gap and those above it from CC}\}$

   $\text{NEWCLUSTER}_2 = \{\text{All other classes in CC}\}$

5. Repeat step 2 with $CC = \text{NEWCLUSTER}_i (i=1,2)$ until all clusters are singleton sets of classes.

6. Repeat steps 1, 2 and 3 until means stop changing (much).

Fig. 3. docFAUST algorithm.
we convert 8 to binary 01000 and apply the Predicate Tree formula, i.e., $P_{A>CH} = (P_{4,4} \lor (P_{4,3} \lor (P_{4,2} \lor (P_{4,1} \lor P_{4,0}))))$. Here, $P_{4,4}$, $P_{4,3}$, $P_{4,2}$, $P_{4,1}$ and $P_{4,0}$ are the P-Trees representing the PW attribute for the entire dataset. Thus $P_{setosa} = P_{A>CL} \land P_{A>CH} \land P_{REMAINING} \land P_{REMAINING}$ now holds only those data points which have not been classified as $P_{setosa}$. We repeat the above procedure to classify $P_{versicolor}$ and $P_{virginica}$.

4. docFAUST

As a prerequisite of the docFAUST algorithm, the documents in the document space are required to go through preprocessing phase. This is done to avoid inconsistency and add value to the words representing the documents. We perform the preprocessing phase in two steps. The first step is to remove the punctuation marks such as (., !, ?, \\, etc.). In addition, we remove the most commonly occurring words in English vocabulary also called the stop words. This list includes words like am, is, are, etc. [12]. In the second step, we make use of a stemming algorithm to reduce the words to their stem or the root form. For example, words like elects, electable, electability, etc. are reduced to elect [13]. [3] is a source of bag of words where the documents which have already been processed. Once preprocessing has been done, we proceed for the algorithm shown in Fig. 3.

Similar to imgFAUST algorithm, for explanation, we use the IRIS dataset consisting of 50 samples each of Iris-setosa, Iris-virginica and Iris-versicolor with four attributes namely, petal length, petal width, sepal length and sepal width. It must be noted that while using docFAUST to cluster the documents, the only changes we make is that the attributes of the flower and the flower sample are replaced by the shortlisted words in the document and the document itself. To illustrate, the documents Vs words matrix is represented in Table 4 below. Here, we have two types of documents, the travel guides and sports magazines. The documents are listed row-wise and the words are listed column-wise. Words are listed in the decreasing order of their weighted word frequency across the documents. For example, words like mountain, beauty, adrenaline, etc. have weighted word frequency on a higher side and hence they contribute significantly to deciding which document belongs to which cluster.

Next, like imgFAUST algorithm, we construct the data mean table and data mean gap tables. We then construct the MeanTable and sort the rows in decreasing order of the mean gap as shown in previous Tables 1 and 2. In the figure below, cl and att represent the class and attributes of the data.

Once the MeanTable has been constructed as shown in Table 5, we start by choosing the first record {se, pl, 14, 33}. c is mean + gap/2 or 14 + 33/2 = 31. $P_{NEWCLUSTER1} = P_{A>31} \land P_{CC}$ and $P_{NEWCLUSTER2} = P'_{A>31} \land P_{CC}$. The & operation among the Predicate Trees is a single pass operation as shown in [16]. Thus in the first pass of the algorithm, $P_{NEWCLUSTER2}$ represents a cluster comprising of singleton set of Iris-setosa and $P_{NEWCLUSTER1}$ represents a cluster consisting of Iris-versicolor and Iris-virginica. Subsequent pass of algorithm is able to separate out the two subclusters represented by $P_{NEWCLUSTER1}$.

5. Implementation and algorithm analysis

We execute our algorithm on a standard Ubuntu Linux machine with JAVA and GNU g++ compiler installed on it. We use an extractor program written in JAVA to extract the red, green and blue values along with the x,y pixel coordinates of the image in raster order. The values are then converted into P-Trees using the APIs as defined in [16]. The P-Trees are then fed to the FAUST P algorithm. The
algorithm has been implemented in standard C++ language. It outputs the new P-Trees corresponding to the classes defined in the input. This is then sent back to the JAVA program to output the resultant image. The entire application is generalized to take any number of classes and attributes as input. To calculate the algorithm speed, we use the standard gettimeofday system call with a resolution of 1 microsecond.

As can be analyzed from the algorithms described in section , our algorithms have a complexity of O(k) where k is the number of attributes or columns. This is extremely fast considering the fact that all the horizontal methods have at least O(n) assuming no suitable indexing. The value of k is generally small ranging from 2 to 7 (in case of Landsat data). Our algorithm achieves an accuracy of 95% on IRIS dataset with only 1 epoch. Higher accuracy can be achieved at the cost of time.

6. Conclusion and future work

In this paper, we propose two novel algorithms, imgFAUST and docFAUST based on Predicate-Trees. Predicate-Trees are data-mining-ready and compressed data structures which process vertical data, horizontally. Thus our algorithms are an ideal choice for data mining queries involving large dataset since we only bring those columns to memory which contribute to getting the query result rather than bringing entire row of data in case of conventional database systems. Our algorithms have a complexity of O(k) where k is the number of attributes. For future work, we have developed a variation of existing FAUST algorithm and plan to apply it on DNA microarray data.

References