Extract Classify - An Application Pattern

1 Overview to Application Patterns

Application domains include a toolbox of different techniques and algorithms perfected over time. Much of this knowledge, however, still remains locked up in the domain literature and not readily accessible to people not familiar with the application area. We believe patterns can help us expose the domain knowledge to others not familiar with the depth of the domain and aid the creation of application frameworks that will let software writers (not necessarily domain experts) create interesting applications. These application patterns also serve to show how application domains feed the structural and computational patterns in OPL. In addition to pattern mining from individual examples, application patterns will help codify common practices and help programming framework development tailored to particular application areas.

2 Introduction to the Extract Classify Pattern

The Extract Classify pattern is used in a variety of contexts and domains. In general, the problem centers around understanding data. An overview of the problem is presented in figure 1. In the Extract Classify pattern, we start with some labeled training data, which has been annotated with results known to be correct. We extract features from this data, and then train a classifier based on these features and our known labels. This training process produces a model.
After the model is derived, we can apply it to new, unknown data. First, the data is processed to obtain features, using the same methods as with the training data. Then, the data is classified according to the model to produce the final classification results.

This pattern is used extensively in any field that deals with computer-aided understanding of data, for example: machine learning, statistics, computer vision, control, biostatistics, computational finance, etc. It is a general pattern that has been found useful in many domains because it breaks the problem of data interpretation into well defined steps. The domain expert who invents new algorithms for understanding data can then focus on filling in this pattern with computations that are best suited for her particular problem.

For example, feature extraction can be optimized to provide better classification performance. More concretely, in image object recognition, we hope to find features who can capture the features of objects such that one object can be distinguished from others. In activity detection, we are investigating features for different activities, how to extract differentiating features from images, or strategies for creating discriminative and accurate classifiers.

Given a set of training data, the classification problem is to label a set of query data. For example, given a set of objects with their categories, we want to identify objects with the same categories in other images. Given a set of marked emails as training data, we want to classify query emails as spam or useful email. Given a training set of patients with disease diagnoses, we want to diagnose other patients’ diseases, given their test results.

Many problems can be successfully understood through this application pattern. For people who are new to these fields, we hope to give them some insights about the problems, some basic solutions, and how we can help them develop their program by the Pattern Language and Design Patterns. In the following sections, we introduce feature extraction and classification in the field of computer vision, as motivation for this application pattern.

### 3 Feature Extraction

Feature Extraction can be viewed as a black box whose input is some set of data (structured or otherwise) and whose output is a set of vectors in some high-dimensional space. The dimensionality of the space can vary widely depending on the application, but for Computer Vision applications is typically several hundred, or potentially several thousand. This space can be either a continuous space (i.e. $\mathbb{R}^n$), or a discrete space (e.g. $\{0,1\}^n$), or some combination of the two. Feature extraction is essentially the process of turning unstructured data into a representation with some mathematical structure, so that it is to utilize many algorithms which can achieve the task at hand.

The later sections of this paper will discuss in more detail how these vectors are used, but let us first consider a trivial example from the domain of computer vision. Suppose that we have a set of labeled, full-frame pictures of cars and of elephants (i.e. we know which of these images are cars, and which are
elephants), and we wish to decide whether some previously unseen image is of a car or of an elephant. This is a supervised classification problem: we need to use the labeled examples to compute parameters to some binary classification model. One potentially distinguishing characteristic of our two classes is color: cars are typically painted in bright, unnatural colors, while elephants are typically some shade of brown or gray. Thus potentially useful feature is a histogram of the colors that appear in our images. Most importantly, once we have extracted these features from our images, we are now able to use any number of general-purpose classification algorithms to solve our problem: e.g. Support Vector Machines, Logistic Regression, or Gaussian Processes. The best classifier to choose depends on the characteristics of our data and our computational constraints.

This separation is powerful. Assuming that our features capture the most important information to our desired classification task, we now have extraordinary flexibility in choosing our classifier. There is not much more to say in general terms about feature extraction. As illustrated by our simple example, designing a feature requires specific knowledge about the problem at hand: we would not have chosen color as a feature if we were trying to distinguish cars from motorcycles, for example. The only requirement is that the output be vectors suitable for input to more general purpose classification algorithms. We shall now present a few examples of features used in Computer Vision applications.

3.1 Point Features: SIFT

A popular class of image features identify particular small regions of the image as “interesting”, and provide a vector for each such region. It is beyond the scope of this paper to discuss much detail about the definition of “interesting” used to identify these regions, as there is a large body of literature from the Vision community discussing the topic. However, all definitions usually try to achieve invariance to particular imaging conditions, so that the same regions are identified in multiple images of the same object. For example, most feature schemes achieve scale invariance: that is, they attempt to detect an interest point regardless of the size of the object in the image, since the object’s distance from the camera will be different in different images. Typically an entire object is detected by detecting a set of local features features: for example, a car might be represented by a feature for each wheel, features for the headlights, etc.

The most popular such feature is the Scale Invariant Feature Transform (SIFT) of David Lowe. SIFT is designed for invariance to scale, position, lighting conditions, and rotation. It describes an interesting region via a spatially-binned histogram of the edge directions surrounding the interest point. Essentially, this describes the shape of the region.
3.2 Global Features: HOG

Frequently, we wish to describe entire regions of images, rather than particular points. In the case of elephant-car classification, we were trying to describe the entire image (which was assumed to contain mostly the object of interest). It is usually impossible to assume a priori that the location of interesting objects is known. Thus, we have two choices: we can either require that the input to our system is a full-frame picture of the objects of interest (as we assumed in the elephant-car example), or use a “sliding-window” scheme. That is, we can guess that the object will be with a bounding box of a certain size, and then perform our feature extraction on bounding boxes at every possible position in the image. Since we don’t know a priori what the size of the object will be, we can repeat this for bounding boxes of several sizes. Obviously, we need to be told the locations of objects in the training set, but the sliding window approach is frequently a very effective, if simple, approach to locating objects in images.

A well-known application of this principle is the Pedestrian Detector of Dalal and Triggs. This pedestrian detector uses a descriptor much like the SIFT descriptor for describing regions of an image which contain people standing upright, and train linear Support Vector Machines to distinguish regions containing people from regions not containing people. Essentially, describing regions in terms of edge directions causes the SVMs to detect the silhouettes of people.

4 Classification

The classification problem is to find the corresponding category of a given entity. It is a widely used high-level pattern in various area. For example, in object recognition, we hope to identify objects in an image. In medical diagnostic, we want to decide the disease of a patient by his symptoms. In virus detection, we need to investigate whether an executable is infected. In order to solve this problem, a set of data and their corresponding categories are given, such that we can learn some classification criteria from the given categorized exemplars, and apply the learned criteria to classify other queries. A formal definition of the problem is given as following:

**Definition 1** Let $C$ be the set of possible categories. Let $D = \{(\hat{x}_1, \hat{y}_1), (\hat{x}_2, \hat{y}_2), \ldots, (\hat{x}_n, \hat{y}_n)\}$ with $\hat{y}_i \in C \forall 1 \leq i \leq n$ be the training set of $n$ (data, category) pairs. Given $Q = x_1, x_2, \ldots, x_m$ be a set of $m$ data, find their corresponding categories $y_1, y_2, \ldots, y_m$ with $y_i \in C \forall 1 \leq i \leq m$.

A wide variety of algorithms are proposed to solve the classification problem. The most common approaches include instance-based methods, probabilistic models, linear models, and decision models. In this section, we discuss three widely used classification algorithms.
4.1 K-Nearest Neighbor

The k-nearest neighbors algorithm (k-NN) is an algorithm for classifying objects based on the closest k training exemplars. It is one of instance-based method, where the function is only approximated locally. The basic idea of the k-NN method is very simple. Given a query, find the k nearest neighbors from the training data set, and the k nearest neighbors will vote on the category of the query by the.

In k-NN algorithm, k is a hyperparameter that influences the performance of the algorithm. In order to decide a proper k, training is inevitable. A general training approach is cross-validation. In cross-validation, the training data is divided into a training set and a validation set, train the hyperparameter k by the training set, and validate the performance by the validation set. Finally pick up the k value with the best performance on the validation set. Moreover, usually k is chosen to be an odd number to avoid possible ties.

There are two ways for realizing the k-NN algorithm. One is exact k-NN, the other is approximate k-NN. As their name suggests, the k-NN is finding the exact k nearest neighbors, while the approximate k-NN is finding the approximate k nearest neighbors. The exact method is achieved by an explicit sorting on distances. The approximate method uses some specific data structures such that it only evaluates part of the search space for finding the k nearest neighbors.

There are 3 major computations related to k-NN. Now we analyze the patterns for the computations.

- Distance computation for each pair of data.
  The computation on distance of different pairs of data is task parallel. The computation on distance of one pair of data depends on the definition of distance. For Euclidean distance, the computation is a map-reduce operation.

- Find the k nearest neighbors.
  For the exact k-NN algorithm, the computation is a parallel sorting problem. Different parallel sorting algorithm may be realized by different patterns. For example, insertion sort can be implemented by data parallel, while the quick sort can be implemented by recursive splitting.
  For the approximate k-NN algorithm, usually the data structure is represented by a tree for quick access of the data. Therefore, it can be be parallelized by task parallelism.

- Voting among the k nearest neighbors.
  It is a map-reduce procedure.

V. Garcia and E. Debreuve proposed a GPU implementation on exact k-NN algorithm [1]. For distance computation, it explores the data parallelism on distances of different pairs of data. For finding the k nearest neighbors, it also explores the data parallelism on insertion sort. Finally, they compared the computation time for the GPU exact k-NN implementation with 3 different serial
implementations. The results are summarized in Figure 2. In this table, n refers to the number of training data and query data, d refers to the dimensionality of the data, BF-Matlab is the exact k-NN implemented by MATLAB, BF-C is the exact k-NN implemented by C++, ANN-C++ is the approximate k-NN, and BF-CUDA is the parallel exact k-NN implemented on GPU. According to the results, the efficiency gain by exploring data parallel on GPU is very significant when the number of total nodes and the dimension are large.

<table>
<thead>
<tr>
<th>Methods</th>
<th>n=1200</th>
<th>n=2400</th>
<th>n=4800</th>
<th>n=9600</th>
<th>n=19200</th>
<th>n=38400</th>
</tr>
</thead>
<tbody>
<tr>
<td>d=8 BF-Matlab</td>
<td>0.51</td>
<td>1.69</td>
<td>7.84</td>
<td>35.08</td>
<td>148.01</td>
<td>629.90</td>
</tr>
<tr>
<td>BF-C</td>
<td>0.13</td>
<td>0.49</td>
<td>1.90</td>
<td>7.53</td>
<td>29.21</td>
<td>127.16</td>
</tr>
<tr>
<td>ANN-C++</td>
<td>0.13</td>
<td>0.33</td>
<td>0.81</td>
<td>2.43</td>
<td>6.82</td>
<td>18.38</td>
</tr>
<tr>
<td>BF-CUDA</td>
<td>0.01</td>
<td>0.02</td>
<td>0.04</td>
<td>0.13</td>
<td>0.43</td>
<td>1.89</td>
</tr>
<tr>
<td>d=16 BF-Matlab</td>
<td>0.74</td>
<td>2.98</td>
<td>12.60</td>
<td>51.64</td>
<td>210.90</td>
<td>893.61</td>
</tr>
<tr>
<td>BF-C</td>
<td>0.22</td>
<td>0.87</td>
<td>3.45</td>
<td>13.82</td>
<td>56.29</td>
<td>233.88</td>
</tr>
<tr>
<td>ANN-C++</td>
<td>0.26</td>
<td>1.06</td>
<td>5.04</td>
<td>23.97</td>
<td>91.33</td>
<td>319.01</td>
</tr>
<tr>
<td>BF-CUDA</td>
<td>0.01</td>
<td>0.02</td>
<td>0.06</td>
<td>0.17</td>
<td>0.60</td>
<td>2.51</td>
</tr>
<tr>
<td>d=32 BF-Matlab</td>
<td>1.03</td>
<td>5.00</td>
<td>21.00</td>
<td>84.33</td>
<td>323.47</td>
<td>1400.61</td>
</tr>
<tr>
<td>BF-C</td>
<td>0.45</td>
<td>1.79</td>
<td>7.51</td>
<td>30.23</td>
<td>116.35</td>
<td>568.53</td>
</tr>
<tr>
<td>ANN-C++</td>
<td>0.39</td>
<td>1.78</td>
<td>9.21</td>
<td>39.37</td>
<td>166.98</td>
<td>688.55</td>
</tr>
<tr>
<td>BF-CUDA</td>
<td>0.01</td>
<td>0.03</td>
<td>0.08</td>
<td>0.24</td>
<td>0.94</td>
<td>3.89</td>
</tr>
<tr>
<td>d=64 BF-Matlab</td>
<td>2.24</td>
<td>9.37</td>
<td>38.16</td>
<td>149.76</td>
<td>606.71</td>
<td>2353.40</td>
</tr>
<tr>
<td>BF-C</td>
<td>1.71</td>
<td>7.28</td>
<td>26.11</td>
<td>111.91</td>
<td>455.49</td>
<td>1680.37</td>
</tr>
<tr>
<td>ANN-C++</td>
<td>0.78</td>
<td>3.56</td>
<td>14.66</td>
<td>59.28</td>
<td>242.98</td>
<td>1008.84</td>
</tr>
<tr>
<td>BF-CUDA</td>
<td>0.02</td>
<td>0.04</td>
<td>0.11</td>
<td>0.40</td>
<td>1.57</td>
<td>6.65</td>
</tr>
<tr>
<td>d=80 BF-Matlab</td>
<td>2.35</td>
<td>11.53</td>
<td>47.11</td>
<td>188.10</td>
<td>729.52</td>
<td>2852.68</td>
</tr>
<tr>
<td>BF-C</td>
<td>2.13</td>
<td>8.43</td>
<td>33.40</td>
<td>145.07</td>
<td>530.40</td>
<td>2127.08</td>
</tr>
<tr>
<td>ANN-C++</td>
<td>0.98</td>
<td>4.29</td>
<td>17.22</td>
<td>73.22</td>
<td>302.44</td>
<td>1176.39</td>
</tr>
<tr>
<td>BF-CUDA</td>
<td>0.02</td>
<td>0.04</td>
<td>0.13</td>
<td>0.48</td>
<td>1.98</td>
<td>8.17</td>
</tr>
<tr>
<td>d=96 BF-Matlab</td>
<td>3.30</td>
<td>13.89</td>
<td>55.77</td>
<td>231.69</td>
<td>901.38</td>
<td>3390.45</td>
</tr>
<tr>
<td>BF-C</td>
<td>2.54</td>
<td>10.56</td>
<td>39.26</td>
<td>168.58</td>
<td>674.88</td>
<td>2649.24</td>
</tr>
<tr>
<td>ANN-C++</td>
<td>1.20</td>
<td>4.96</td>
<td>19.68</td>
<td>82.45</td>
<td>339.81</td>
<td>1334.35</td>
</tr>
<tr>
<td>BF-CUDA</td>
<td>0.02</td>
<td>0.05</td>
<td>0.15</td>
<td>0.57</td>
<td>2.29</td>
<td>9.61</td>
</tr>
</tbody>
</table>

Figure 2: Computation time for different parallel implementation and serial implementations.

4.2 Logistic Regression

The logistic regression is a probabilistic model for the classification problem. Given the training data D, it finds a set of parameters θ such that the maximum log likelihood of p(θ|D) is maximized. Then it uses the parameter set θ as well as the probability model to classify each query data.

In logistic regression, the conditional probability p(y|x) is modeled by a function φ(θ^T x), where φ is the logistic function. Therefore, the conditional probability p(Y = k|x, θ) is modeled by the softmax-linear model:

\[
\frac{e^{\theta_k^T x}}{\sum_l e^{\theta_l^T x}}
\]  

(1)
where the summation in the denominator is summing over all possible category label \( l \).

Let

\[
\mu^i = \frac{e^{\theta^T x}}{\sum_l e^{\theta^T x^l}},
\]

we can calculate the conditional probability of the category \( y_i \) given data \( x_i \) and parameter set \( \theta \) by

\[
p(y_i|x_i, \theta) = \prod_k (\mu^i_k)^{y^i_k},
\]

where \( k \) is the set of possible categories, and \( y^i_k = 1 \) if and only if the category of data \( x_i \) is \( k \). Therefore, the log likelyhood can be computed as

\[
l(\theta|D) = \prod_n \prod_k y^k_i \log u^k_i,
\]

where \( n \) is the set of training data.

In order to find parameter set \( \theta \) that maximizes log likelyhood, we can apply optimization algorithms. For example, nonlinear conjugate gradient, steepest descent, and Newton-Raphson algorithm can be applied to solve the optimization problem.

The kernel logistic regression is a nonlinear form of the logistic regression. By applying kernel trick on the data set, the data \( x \) is mapped to a high-dimension Hilbert space, and then apply the logistic regression on it.

Now we analyze the computation involved in the logistic regression method. The map-reduce pattern can be applied to the dot product operation. Dense linear algebra can be applied on the iterative optimization methods for optimizing the parameter set.

### 4.3 Support Vector Machine

Support Vector Machine are a class of classifiers based on the maximum-margin principle. In general terms, out of all possible planes that separate the classes, SVM picks the classifier with the largest margin between the classes. This gives classifiers that fit the training data well, and also perform well on other data by avoiding overfitting.

We consider the standard two-class soft-margin SVM classification problem (C-SVM), which classifies a given data point \( x \in \mathbb{R}^n \) by assigning a label \( y \in \{-1, 1\} \).

#### 4.3.1 SVM Training

Given a labeled training set consisting of a set of data points \( x_i, i \in \{1, \ldots, l\} \) with their accompanying labels \( y_i, i \in \{1, \ldots, l\} \), the SVM training problem can
be written as the following Quadratic Program:

$$\max_{\alpha} F(\alpha) = \sum_{i=1}^{l} \alpha_i - \frac{1}{2} \alpha^T Q \alpha$$

subject to  
$$0 \leq \alpha_i \leq C, \forall i \in 1 \ldots l$$

$$y^T \alpha = 0$$

(5)

where $x_i \in \mathbb{R}^n$ is training data point $i$, $y_i \in \{-1, 1\}$ is the label attached to point $x_i$, and $\alpha_i$ is a set of weights, one for each training point, which are being optimized to determine the SVM classifier. $C$ is a parameter which trades classifier generality for accuracy on the training set, and $Q_{ij} = y_i y_j \Phi(x_i, x_j)$, where $\Phi(x_i, x_j)$ is a kernel function. Two popular kernel functions are

- Linear: $\Phi(x_i, x_j) = x_i^T x_j$
- Polynomial: $\Phi(x_i, x_j; a, r, d) = (ax_i \cdot x_j + r)^d$
- Gaussian: $\Phi(x_i, x_j; \gamma) = \exp\left\{-\gamma ||x_i - x_j||^2\right\}$
- Sigmoid: $\Phi(x_i, x_j; a, r) = \tanh(ax_i \cdot x_j + r)$

In the optimal solution of the QP, the points which have non-zero $\alpha$’s are called Support Vectors.

This problem can be solved by different techniques. One of the most popular ways to solve it is through Sequential Minimal Optimization (SMO) [5]. We refer the reader to [5] for details of the algorithm. The important patterns in the computation are

- Computing the gradient of the objective function
  This involves a parallel map computation. Each map computes the kernel function of a point with a point or a set of points.

- Finding the next set of points to update
  This computation involves a reduce function (usually min or max). This step is usually combined with the previous stage to produce a map-reduce pattern.

- Convergence
  Update the points until the objective stabilizes or the duality gap goes to zero. This is done using a simple while loop.

Other methods for solving the training problem involve Linear Algebra, Map-Reduce and Data Parallelism.

4.3.2 SVM classification

The SVM classification problem is as follows: for each data point $z$ which should be classified, compute

$$\hat{z} = \text{sgn}\left\{b + \sum_{i=1}^{l} y_i \alpha_i \Phi(x_i, z)\right\}$$

(6)

8
where $z$ is a point which needs to be classified, and all other variables remain as previously defined. The exact computational needs depend on the kernel used.

For linear SVMs, the computation involves a single dot product between $z$ and $\sum_{i=1} y_i \alpha_i x_i$.

For other kernels, the computation involves a matrix-vector product like operation for each classification. If more than one point is being classified, it is better to combine all the points for classification leading to a matrix-matrix product. Matrix-matrix product (BLAS3) operations are highly optimized for compute throughput and are processed very efficiently. For example, the norm operation in a Gaussian kernel can be written as $||x - y||^2 = x \cdot x + y \cdot y - 2x \cdot y$ to recast the computation into vector norm computations and a Matrix Matrix multiplication.

5 Related Patterns

1. Dense Linear Algebra
   Dense Matrix operations are needed for most of the techniques - esp distance computations in k-NN, kernel operations in SVM etc.

2. Map Reduce
   Many computations can be written as map reduce - e.g. SVM training using SMO.

3. Data Parallelism
   Data parallelism is prevalent in all these techniques - e.g classify each point separately, distance computation can be done for each point in parallel etc.

References


6 Authors

Narayanan Sundaram, Bor-Yiing Su, Mark Murphy, Bryan Catanzaro