Scientific Computing Practical,

University of Göttingen

Efficient Graph-based Image Segmentation

The Felzenszwalb-Huttenlocher Algorithm and Steps of Preparation

Student:	Nils Dörrer
Id:	21345471
E-Mail:	nils.doerrer@stud.uni-goettingen.de
Date:	01.03.2018 - 17.04.2018
Supervisor 1:	Dr. J. Schulz
Supervisor 2:	R. Budinich

Inhaltsverzeichnis

1 Motivation, Goals and Means

Data segmentation, so mapping data points to classes has been a universal problem for a long time and it still is. The methods for achieving this goal range from distance-[13] or density-driven[4] clustering techniques up to deep machine learning like convolutional neural networks[3]. In this project the focus lies on the basic idea to implement a method which is capable of segmentating a huge variety of different data forms. The approach is to use graph based segmentation, so the breaks down in two tasks:

1. Creating a graph from given data using attribute differences and structure to compute edge weights.

2. Performing a segmentation algorithm on the obtained graph and transforming the result into something meaningful.

For the first step some methods were developed to handle images (png, tiff, jpg) and attribute tables (csv) and convert them into graphs. The second step is an implementation of the Felzenzwalb-Huttenlocher algorithm[5].
Scope of this project is to create a python module containing functions handling the data (loading, storing, converting, etc.), building graphs from arrays in different ways and performing the segmentation by using a union-find datastructure[12]. Additionally the segmentation is then transformed back to either an image (for image input data) or a colored scatterplot of PCA-attribute[14] values (for CSV input). The implementation makes use of the python programming language[9] and the modules numpy[10], scikit-image[11], networkx[6], matplotlib[7] and scikit-learn[8].

2 Theoretical Foundation

In this section the theoretical basis used for the implementation is roughly explained.

2.1 Obtaining a graph from a grayscale image

An image contains two features which can be made use of when creating a graph from it.

1. Intensities (pixel values in a grayscale image)
2. Neighborhood structure (contrast between adjacent parts of the image)

In the model used here, every pixel corresponds to a vertex of the resulting graph. Two vertices u, v are connected by an edge if and only if the distance $d(u, v)$ is smaller or equal to a constant R.

$$d(u, v) := \sqrt{(u_x - v_x)^2 + (u_y - v_y)^2} \tag{1}$$

for u_x, u_y the coordinates of pixel u and v_x, v_y the coordinates of pixel v. The weight w of this edge is:

$$w(u, v) := |I(u) - I(v)| \cdot d(u, v) \tag{2}$$

for $I(u)$ and $I(v)$ the intensity values of the pixels u and v. So an edge weight is high if there is a high difference in intensities of the corresponding pixels, or if the distance of these pixels is large.
For $R = 1$ the resulting graph is a grid graph containing only the direct neighbor information. In general if R is set constant, the number of edges m is in $O(n)$, for n the number of vertices.

2.2 Obtaining a graph from an attribute table

Here it is assumed that attribute tables consist of datapoints represented by a row. The first column should contain a unique identifier. All the other columns contain values of features, such that there is one reasonable distance metric for all the different attributes. For this project the following metric is used:

$$d(u, v) := \sum_i |(T_u^*)_i - (T_v^*)_i| \tag{3}$$

where T_u^* is the attribute vector of the item u, so the u-th row of the table except the value in the first column which corresponds to the id and is not an attribute (T_v^* analogue).

In the model used here, each row of the table corresponds to a vertex of the resulting graph. Two vertices u, v are connected by an edge if and only if the distance $d(u, v)$ is smaller or equal to a constant R. The weight w of this edge is:

$$w(u, v) := d(u, v) \tag{4}$$

This might lead to an extremely high number of edges $m \in O(n^2)$, so with increasing number of vertices (number of rows in the table) the number of edges increases quadratically. By setting R in a clever way the number of edges may be reduced heavily which increases performance of the succeeding steps of the algorithm.

2.3 The Felzenszwalb-Huttenlocher Algorithm

The segmentation task on a given graph $G = (V, E)$ with edge weights $w((u, v))$ for $(u, v) \in E$ is performed by the Felzenszwalb-Huttenlocher Algorithm[5]:

1. Sort the edge-set E into sequence $\pi = (o_1, ..., o_m)$, such that $w(o_i) \leq w(o_j) \quad \forall i \leq j$
2. Create a segmentation S^0 of the vertices, where each vertex is in its own component. Here a segmentation refers to a mapping from the vertices to components (C_v is the component containing vertex v).
3. For i from 1 up to m do:
   From the previous Segmentation S^{i-1} construct the next one S^i.
   This is done by either merging the components C_u^{i-1} and C_v^{i-1} containing u and v, connected by edge $o_i = (u, v)$ or keeping the previous state segmentation. The two components are merged, if $w((u, v))$ is small compared to the internal differences of those components, i.e. if $w((u, v)) \leq MInt(C_u^{i-1}, C_v^{i-1})$ (see below).
4. Return S_m

$MInt(C_u, C_v)$ is a measure of the internal differences of the components C_u and C_v:

$$MInt(C_u, C_v) = \min(Int(C_u) + \tau(C_u), Int(C_v) + \tau(C_v)) \tag{5}$$

$$Int(C) = \max_{e \in MST(C,E)} w(e) \tag{6}$$

$$\tau(C) = \frac{k}{|C|} \tag{7}$$

In these equations E is the set of all edges of the graph and $MST(E, C)$ is the minimal spanning tree[12] of the Component C (subset of the set of vertices) given the edge set E. The edges used for merging two components are belonging to $MST(E, C)$, because the considered edge sequence is ordered. This works in a way analogue to the Kruskal algorithm[12]. k is a constant which can be used to adjust the merging capabilities of the algorithm (the higher k the more components may be merged).

2.4 Union-Find

To efficiently manage segmentation tasks like merging of components and computing Int-values, a union-find-structure[12] has been implemented. Union-find is a forest where each component is a tree. Merging components is done by adding the root of one component as an additional child of the root of the other. Like that looking for the identifying node (being the root) of a component is computationally cheap ($O(\log n)$ on average).
This concept can be implemented using two arrays, one for the parent relation between nodes and another one holding cardinality values of the components. E.g. parent[i] contains the index of the parent node of i (which is i if i is a root) and cardinality[i] contains the number of elements in the sub-tree rooted at i (which is 1 if i is the only node in a component).
Using this concept it is efficiently possible to keep track of Int values of components, by adding another array for that. When two components are merged, the resulting Int value (stored at the roots index) is the maximum of the Int values and the weight of the edge connecting the merged components. This works, because Algorithm2.3 uses a sorted edge sequence and like that the edges leading to merge operations build up MSTs.

2.5 Principal Component Analysis

Principal component analysis (PCA) is a tool which can reduce the number of dimensions of a vector while having minimal information loss. This is done by transforming the vector into another basis, inducing the property that the first dimension explains the largest part of the variance of the data, the next dimension the second largest part and so on. The first dimension axis is found by linear regression and the following ones are always orthogonal to those already found. Using PCA makes it possible to remove the last few dimensions of a vector and like that reduce its complexity. The resulting vector still explains a large part of the variance of the original one.
In this project PCA is used to visualize points in a highdimensional vectorspace, namely datapoints of an attribute table. Like that, after the segmentation into classes, the first two principal components of the attribute vectors are shown in a scatterplot where each class is represented by a different color.

3 Implementation

The script egbis.py (efficient graph based image segmentation) was written to give access to the functionality of the egbislib module in a user friendly manner. Pseudocode of this script is this:

- Parse input (parameters exp, $sigma$, k, R, $rows$, $weighted$, $useskimage$, $help$). The default values are: $exp = 1$, $sigma = 0$, $k = 1$, $R = 1$, $rows = all$, $weighted = false$, $useskimage = false$, $help = false$
  - if help parameter is set: display full usage and parameters information, afterwards stop script
- if input file is an image:
  - Load image, convert to grayscale and stretch to intensity range (0,1). Then the image is scaled by the factor 2^{exp}.
    - if parameter $useskimage$ is set, compute image segmentation with the skimage builtin function felzenszwalb with parameters k and $sigma$
    - else apply a gaussian filter of standard deviation $sigma$ to the image and build a graph with distance threshold R (if parameter $weighted$ is set, use product of intensity and distance as edge weights). After that run the Felzenszwalb-Huttenlocher algorithm 2.3 and convert the resulting segmentation back to an image.
  - Save the result as a file in 'images' directory and show both original image and result on screen.
- if input file is an attribute table (CSV):
  - Load attribute table file as array (using $rows$ as specified as parameter)
  - Build a graph from this array with respect to distance threshold R
  - Perform the Felzenszwalb-Huttenlocher algorithm 2.3 with threshold factor k and store the segmentation.
  - For the classes of this segmentation compute summary statistics like mean and variance of each attribute
  - Additionally perform a PCA of these attributes and plot first two principal components as scatterplot with class as point color, show the plot on screen and save it as a file.

Additionally there is a small script test.py, which performs the algorithm on the image falcon.jpg (downscaled by factor 2^4 in width and height). This uses $sigma = 1.3$, $R = 3$, $k = 2$ and $weighted = true$. The script is intended to be used for testing purposes, like checking whether the python libraries are all present.

4 Application

4.1 Requirements

To execute the code the Python2.7 [9] interpreter is necessary. Additionally the following libraries are needed:

- matplotlib 2.2.0 [7]
- networkx 2.1 [6]
- numpy 1.14.2 [10]
- scikit-image 0.13.1 [11]
- scikit-learn 0.19.1 [8]

Usually those libraries are backwards compatible so having newer version might not be a problem, but that cannot be guaranteed.
As for hardware, a lot of RAM may be needed if working with larger images or attribute tables. E.g. for uncompressed 480 x 270 Pixel image the programm needs approximately 9GB.

4.2 Examples

4.2.1 Image Examples

As a testing image a wallpaper of the Millenium Falcon (from starwars by George Lucas)[2] was used. Some results are shown below:
Using a scaling exponent $exp = -2$, distance threshold $R = 3$, threshold factor $k = 2.5$, a gaussian standard deviation $sigma = 1.3$ and a weighted distance graph can be done by calling

python egbis.py –weighted –exp −2 –sigma 1.3 –k 2.5 –R 3 images/falcon.jpg (8)

Fig. 1: Original image (grayscale).

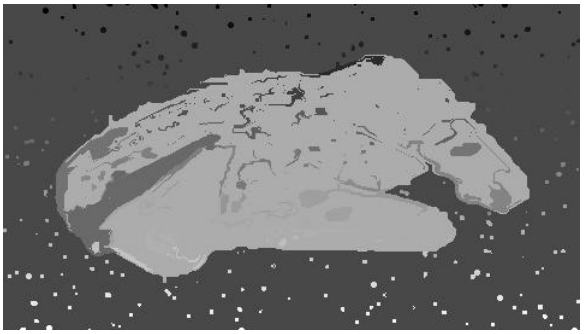

Fig. 2: Classes of the segmentation.

By increasing the threshold factor to $k = 5.0$, a segmentation containing fewer classes can be created. This is done by calling:

python egbis.py –weighted –exp −2 –sigma 1.3 –k 5.0 –R 3 images/falcon.jpg (9)

Fig. 3: Original image (grayscale).

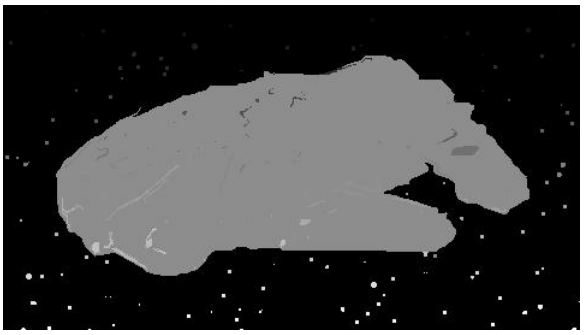

Fig. 4: Classes of the segmentation.

Both these examples take approximately 3 hours to run on my PC (average statistics), so for running only tests it is recommended to rather use $exp = -4$ which reduces the runtime to less than a minute.

4.2.2 Attribute Table Examples

To test the algorithm on data different from image data an attribute table containing attributes of different football players of the FIFA PC game[1] has been chosen. The first column contains a unique identifier for the players. All the other columns contain values between 0 and 100 and correspond to different attributes. This is an example of the raw data:

Listing 1: First 5 rows of fifastats.csv

```
16510  46  55  37  43  32  68  67  65  50  69  62  54  39  49  53  41
17511  52  49  18  21  17  69  67  40  35  68  51  47  35  64  48  40
2076   61  42  69  78  67  61  81  78  87  44  58  65  75  41  28  37
9001   64  65  62  63  65  77  75  75  37  91  90  76  54  50  35  59
496    85  80  52  55  65  72  71  77  65  81  75  69  57  72  62  76
```

Using a distance threshold $R = 250$ and threshold factor $k = 4000$ on the first 400 rows of the data file can be done like this:

python egbis.py –k 4000.0 –R 250 –rows 400 football-dataset/fifastats.csv (10)

Since the result would be a set of 16-dimensional vectors, a PCA (2.5) of the data is performed and the first two principal components are plotted.

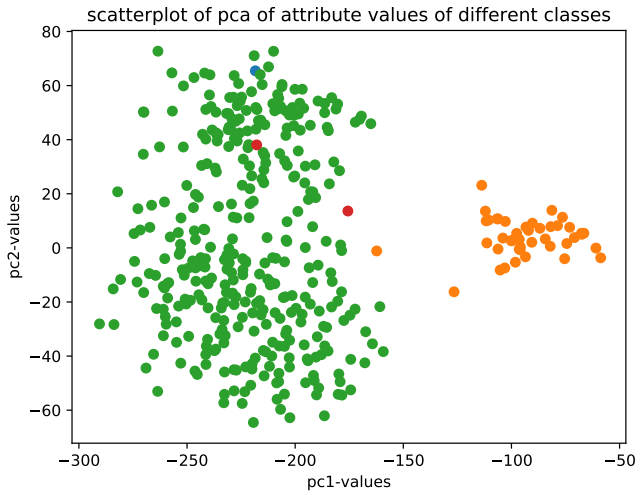

Fig. 5: PCA-scatterplot for 400 rows.

It can be seen that even in this PCA-view of the data an effect of clustering is visible. Setting $R = 150$ to reduce runtime and $k = 8000$, the algorithm can be applied on the first 4000 rows, by using:

python egbis.py –k 8000.0 –R 150 –rows 4000 football-dataset/fifastats.csv (11)

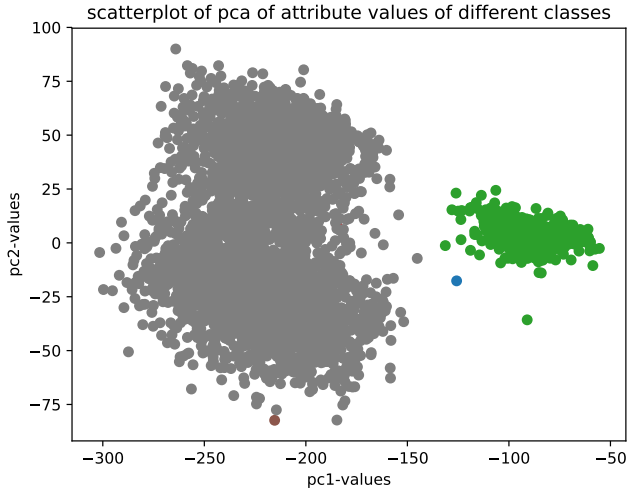

Fig. 6: PCA-scatterplot for 4000 rows.

5 Summary and Lookout

The Felzenszwalb-Huttenlocher algorithm[5] is capable of segmentating any kind of graph. That means that using this algorithm is a feasable solution to any kind of segmentation problem, as long as its possible to transform the given input data into a graph. Since these graphs can be really huge, the runtime of the algorithm can vary a lot and may be bad, depending on the graph generation parameters. Unlike other segmentation algorithms, the Felzenzwalb-Huttenlocher algorithm does not tend to produce round class shapes.
On the data tested here, namely grayscale images as well as attribute tables the segmentation seems convincingly good and the pivotal point really is the graph generation to ensure good results.
To further improve the project, there are a few points where runtime performance might be increased. One idea is to use path compression[12] in the union-find merge steps, which would decrease the time for find-operations. Additionally iteration over image pixels is quite inefficient in python language, which could be improved by using a language like Fortran or C. Another aspect to be considered is trying to find a way of automatic parameter selection which would minimize the need for human input during algorithm execution.

Literatur

[1] fifastats - fifa 2017 player dataset.

[2] Millenium falcon wallpaper from starwars by george lucas.

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Esegnet: A deep convolutional encoder-decoder architecture for image segmentation. *IEEE TRANSACTIONS ON PATTERN ANALYSIS AND MACHINE INTELLIGENCE*, 39, Dec 2017.

[4] Derya Birant and Alp Kut. St-dbscan: An algorithm for clustering spatial–temporal data. *Data and Knowledge Engineering*, 60(1):208 – 221, 2007. Intelligent Data Mining.

[5] Pedro F. Felzenszwalb and Daniel P. Huttenlocher. Efficient graph-based image segmentation. *International Journal of Computer Vision*, 59(2):167–181, Sep 2004.

[6] Aric Hagberg, Dan Schult, and Pieter Swart. networkx, 2002.

[7] J.D. Hunter. Matplotlib: A 2d graphics environment, 2007.

[8] T. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, and E. Duchesnay. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830, 2011.

[9] Python Software Foundation Python Core Team. Python: A dynamic, open source programming language, 2015.

[10] S. van der Walt, S.C. Colbert, and G. Varoquaux. The numpy array: A structure for efficient numerical computation, 2011.

[11] S. van der Walt, J.L. Schönberger, J. Nunez-Iglesias, F. Boulogne, J.D. Warner, N. Yager, E. Gouillart, T. Yu, and the scikit-image contributors. scikit-image: image processing in python, 2014.

[12] Stephan Waack. Scriptum of informatics 3 lecture. Oct 2015.

[13] Kiri Wagstaff, Claire Cardie, Seth Rogers, and Stefan Schroedl. Constrained k-means clustering with background knowledge. *Proceedings of the Eighteenth International Conference on Machine Learning*, 2001.

[14] S. Wold, K. Esbensen, and P. Geladi. Principal component analysis. *Chemometrics and Intelligent Laboratory Systems*, 2(1):37 – 52, 1987. Proceedings of the Multivariate Statistical Workshop for Geologists and Geochemists.