Unsupervised multispectral image classification with genetic algorithms

Piotr Krzemiński Krzysztof Nowicki Radosław Warzocha

Wrocław, December 15, 2013

1. Introduction

Multi-spectral image classification is important procedure in remote sensing. Supervised classification requires a human analyst to provide training data for algorithm in order to infer the classifier. On the other as amount of data we get from satellitar imaging devices grows, so grows need for unsupervised classification, which requires almost no human work. Main problem of this algorithms is that they need to know in advance how many different clusters (we can think of them as 'types of terrain', like water, forest, field etc.) will there be on the picture, and this value is usually not known a priori. In this paper we present results of applying some genetic algorithms to this problem.

To test our work we used Landsat 7 multispectral images, containing data for 5 bands - from visible green to mid infrared. All the software was developed using Scala programming language and is attached to this report.

2. Definition of the problem and proposed solution

2.1. Chromosome representation

Our chromosome should be set of potential cluster centroids in multi-spectral space. Set size (length of the chromosome) will be selected form range $[K_{min}, K_{max}]$ where K_{min} should be usually equal 2 (unless some special case is considered) and K_{max} must be selected manually according to experience. Each gene in the set will represent centroid, so it will be point in $[0, 255]^B$, where B is number of bands in our image. In our implementation chromosomes are collection of such points and length K_{max}, but every gene can also be equal to `None` and represent invalid (non-existent) centroid. So for example individual

$$[(100, 100, 100, 100), None, (5, 25, 125, 255), (123, 234, 134, 124), None]$$

represents 3 centroids in four-dimentional multi-spectral space .

2.2. Mutation, crossover and selection

We decided to use standard well-known operators of mutation, crossover and *roulette wheel* selection.

Mutation will be applied with certain low probability to some genes, producing completely new random centroid in place of the mutated one. So for example we can have individual

$$[(100, 100, 100, 100), None, (5, 25, 125, 255), (123, 234, 134, 124), None]$$

which after mutation of the second gene can become

$$[(100, 100, 100, 100), (50, 75, 125, 200), (5, 25, 125, 255), (123, 234, 134, 124), None]$$

Crossover shall pick random natural number k between 0 and K_{max}, divide each of two chromosomes into two parts: first of length k and second of length $K_{max} - k$ and finally switch the second part of the chromosomes. So for example

$$\begin{array}{c} \text{parent 1}: [(100, 100, 100, 100), None, (5, 25, 125, 255), (123, 234, 134, 124), None] \\ \text{parent 2}: [(34, 97, 160, 20), (199, 12, 64, 70), (63, 0, 49, 50), None, (1, 99, 18, 77)] \\ \downarrow k = 3 \\ \text{child 1}: [(100, 100, 100, 100), None, (5, 25, 125, 255), None, (1, 99, 18, 77)] \\ \text{child 2}: [(34, 97, 160, 20), (199, 12, 64, 70), (63, 0, 49, 50), (123, 234, 134, 124), None] \end{array}$$

2.3. Fitness function

K-Means algorithm is one of deterministic algorithms, that given set of observation vectors (vectors of brightness of image pixels in our case) creates set of specified size containing centroids for those vectors. We can use it's way of 'rating' sets generated in each iteration as our fitness function.

In K-Means algorithm each pixel is assigned to the closest cluster centroid. Our fitness function - K-Means Index (KMI) - will represent total variation of those assignments. KMI is computed as follows:

$$KMI = 1/\left(\sum_{k=1}^{K}\sum_{i=1}^{N}\mu_{ik} \parallel x_i - \nu_k \parallel^2\right)$$

where
K is the number of clusters in chromosome,
N is number of pixels in the image,
μ_{ik} is membership function of pixel X_i belonging to the k^{th} cluster and
$\nu_k = \frac{\sum_{i=1}^{N}\mu_{ik}x_i}{\sum_{i=1}^{N}\mu_{ik}}$ is average value of pixels in k^{th} cluster.

Other fitness function we can use is Xie-Beni's Index (XBI) which is not very different from KMI. Again, each pixel is assigned to the nearest cluster centroid. Then, however, we take $d_{min} = \min_{k,j} \parallel \nu_k - \nu_j \parallel$ to be minimum distance between any two of . Finally, XBI value is calculated:

$$XBI = N \cdot d_{min}^2 / KMI = N \cdot d_{min}^2 / \left(\sum_{k=1}^{K}\sum_{i=1}^{N}\mu_{ik} \parallel x_i - \nu_k \parallel^2\right)$$

Our third choice is Davies-Boundin Index (DBI), which is similar to the previous index, only this time we're going to add standard deviation of the cluster.

Let $S_k = \left(\frac{1}{M}\sum_{x_i \in X_k} \parallel x_i - \nu_k \parallel^2\right)^{1/2}$ be the standard deviation of k^{th} cluster. Then

$$DBI = K / \sum_{k=1}^{K} R_k$$

where

$$R_k = \max_{j,j \neq k}\left\{\frac{S_k + S_j}{d_{kj}}\right\}$$

We'll try each of this functions and see if there is one strictly better then the others.

3. Results of experiments

3.1. Methods convergence

First thing we should do when trying to determine if our algorithm is good is methods convergence. To determine if and how fast is our method convergent we conducted simple test by running our program on small multi-spectral image and compared its output with hand-made classification of those data. Following figure shows compliance percentage of those two images.

3.2. Tuning parameters

One of our main concerns was tuning the parameters of the algorithm. Not only do we have to tune standard ones: population size, mutation and crossover probabilities and iteration numbers, but also length of the chromosome and probability of non-existent centroid `None`. We decided that our main focus should be on the latter.

First of all we conducted experiments, that showed, that mutation and crossover probability values equal 0.7 and 0.05 respectively were giving good results to conduct further experiments.

Let's first understand why tuning length of the chromosome and probability of `None` (called also PN) is important. During our experiments we have started with values 6 and 0.1 for those factors respectively. We have noticed that our population contained many individuals with none or very small amount of 'no centroid' genes. It's going to be a problem when we have simple image which contains not so many clusters. On the other hand, if we set PN to be high (like 0.6) we can loose some existing centroids in favor of `None`s (situation presented in the following figure).

Our experiments showed that PN value of ... gives most accurate results. We also suggest to set K_{max} to value ≈ 12 as we believe it is easier to merge clusters afterwards than to extract clusters that were unified by the algorithm.

3.3. Final results

After tuning parameters we can finally present results of our algorithm. First let's take a look at one of the original pictures we used to test the outcome.

Following table shows the pictures that were the output of algorithm for different indexes and population sizes.

Finally it's important to measure the error between results and original image. It was done by manually classifying the original picture and then constructing *error matrix*. Error percentage is shown in the table: