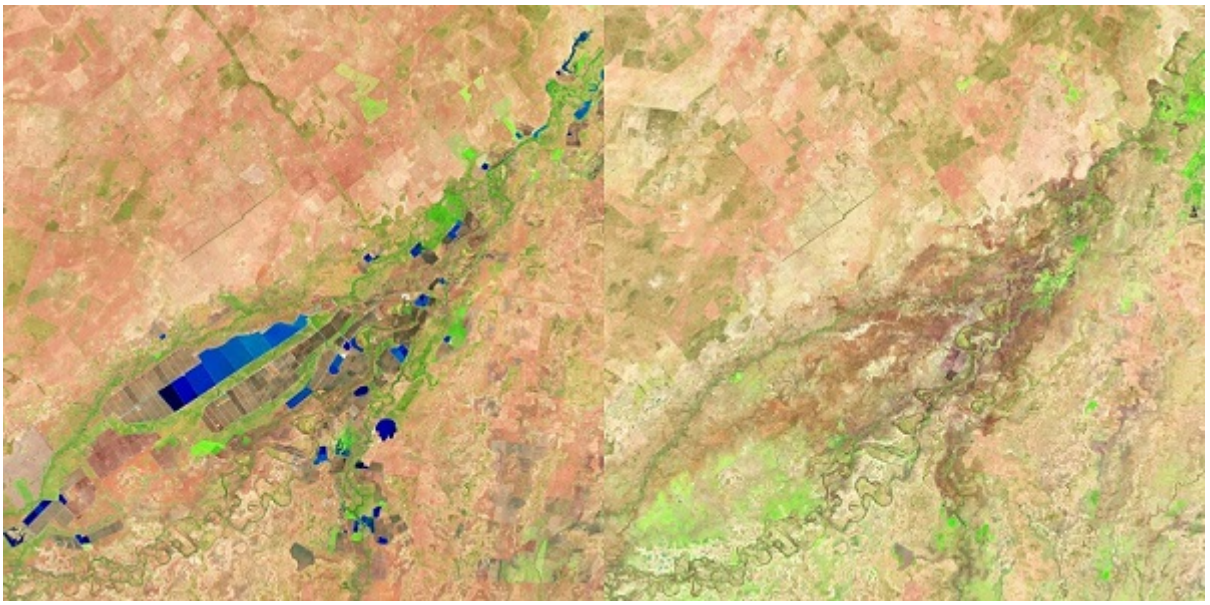
Machine Learning in Action

A perfect hands-on practice for beginners to elevate their ML skills

APPLICATIONS, CLUSTERING, IMAGE PROCESSING

Unsupervised Changed Detection in Multi-Temporal Satellite Images using PCA & K-Means : Python code



Date: November 25, 2017 Author: ML bot1 0 Comments

Automatic change detection in images of a region acquired at different times is one the most interesting topics of image processing. Such images are known as multi temporal images. Change detection involves the analysis of two multi temporal satellite images to find any changes that might have occurred between the two time stamps. It is one of the major utilization of remote sensing and finds application in a wide range of tasks like defence inspections, deforestation assessment, land use analysis, disaster assessment and monitoring many other environmental/man-made changes.

We will be outlining an unsupervised method for change detection in this blog post. It involves the automatic analysis of the change data, i.e. the **difference image**, constructed using the multi temporal images. A difference image is the pixel-by-pixel subtraction of the 2 images. Eigen vectors of pixel blocks from the difference image will then be extracted by Principal Component Analysis (PCA). Subsequently, a feature vector is constructed for each pixel in the difference image by

projecting that pixel's neighbourhood onto the Eigen vectors. The feature vector space, which is the collection of the feature vectors for all the pixels, upon clustering by K-means algorithm gives us two clusters – one representing pixels belonging to the changed class, and other representing pixels belonging to the unchanged class. Each pixel will belong to either of the clusters and hence a **change map** can be generated. So, the steps towards implementing this application are:

1. difference image generation and Eigen vector space (EVS)
2. building the feature vector space (FVS)
3. clustering of the feature vector space and change map

We will be demonstrating the working of the algorithm on an image corpus developed from the LANDSAT images available United States Geological Survey (USGS) website. The website provides the images in various resolutions and also provides detailed description of the changes that have occurred in the images. Some of the multi-temporal image pairs can be downloaded from here.

Let's dive in!

1. Difference image and the Eigen vector space

As mentioned before, the difference image has the absolute valued differences of the intensity values of the corresponding pixels of the 2 grayscale images. The computed difference image would hence be such that the values of the pixels associated with land changes will have values significantly different from those of the pixels associated with unchanged areas.

$$\text{difference image}(i, j) = |\text{image}_1(i, j) - \text{image}_{2(i, j)}|$$

This can be done in Python as follows if image1 and image2 variables are the 2 images:

```
from scipy.misc import imread, imsave, imresize
import numpy as np

image1 = imread(imagepath1)
image2 = imread(imagepath2)
new_size = np.asarray(image1.shape) / 5 * 5
image1 = imresize(image1, (new_size)).astype(np.int16)
image2 = imresize(image2, (new_size)).astype(np.int16)
diff_image = abs(image1 - image2)
```

Next in line is the task of building the Eigen vector space. Before that, lets take a quick look at what PCA is. PCA is a technique to emphasise variation and bring out strong patterns in a data set. It converts a set of observations of possibly correlated variables into a set of values of linearly uncorrelated variables called **principal components**. It is a widely used method for dimensionality reduction.

PCA takes a data set and determines its co-variance matrix after performing mean normalisation on it. The Eigen vectors and Eigen values of the co-variance matrix are computed (giving us the EVS) and then the Eigen vectors are sorted in the descending order of Eigen values. This sorting step is the actual revelation of the PCA algorithm. The Eigen vectors have been sorted in the decreasing order

of the Eigen values because the Eigen vector with the highest Eigen value is the principal component of the data set. That vector shows along which direction the majority of the data is inclined. Thus by PCA, we have been able to extract the lines that characterise the data. Since this has been a brief introduction to PCA, we encourage you to read more about it and the related concepts from here and here.

In this method, we take non-overlapping blocks of size 5 x 5 from the difference image and flatten them into row vectors. The image can be resized to make both the dimensions a multiple of 5 by `scipy.misc.imresize()`. Collection of these row vectors forms a vector set. In `change_detection.py` script, `find_vector_set()` does exactly this. If the size of our difference image is `m x n`, then the number of rows in the vector set would be $\frac{mxn}{5x5}$.

```
def find_vector_set(diff_image, new_size):

    i = 0
    j = 0
    vector_set = np.zeros(((new_size[0] * new_size[1]) / 25,
25))
    while i < vector_set.shape[0]:
        while j < new_size[0]:
            k = 0
            while k < new_size[1]:
                block   = diff_image[j:j+5, k:k+5]
                feature = block.ravel()
                vector_set[i, :] = feature
                k = k + 5
            j = j + 5
        i = i + 1

    mean_vec   = np.mean(vector_set, axis = 0)
    vector_set = vector_set - mean_vec   #mean normalization

    return vector_set, mean_vec
```

PCA is then applied on this vector set to get the Eigen vector space. The Eigen vector space will be a 25 x 25 matrix; its each column is an Eigen vector of 25 dimensions.In Python, from `sklearn.decomposition`, we can simply import the `PCA` module and use it to perform PCA on `vector_set` variable to get the variable `EVS`.

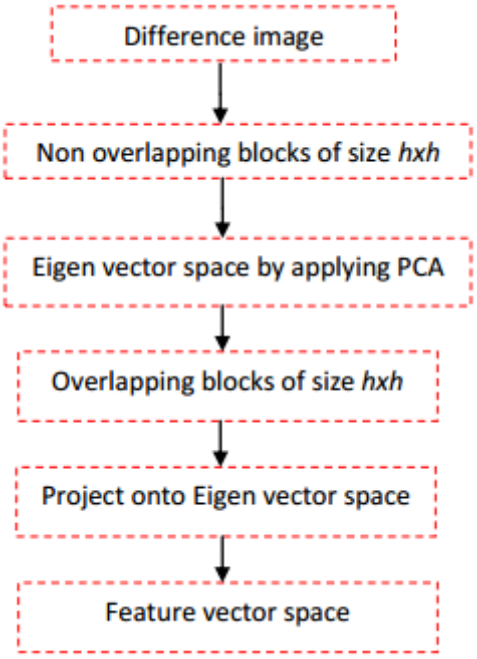
```
from sklearn.decomposition.PCA import PCA

pca = PCA()
pca.fit(vector_set)
EVS = pca.components_
```

2. Building the feature vector space

Building the FVS involves again taking 5 x 5 blocks from the difference image, flattening them, and lastly projecting them onto the EVS, only this time, the blocks will be overlapping. A vector space (VS) is first made by constructing one vector for each pixel of the difference image such a way that one 5 x 5 block is actually a pixel's 5 x 5 neighborhood. It is to be noted here that by this logic, 4 boundary rows and 4 boundary columns pixels won't get any feature vectors since they won't have a 5 x 5 neighborhood. (We can manage with this exclusion of these pixels, since it is safe to assume here that any changes occurring would be concentrated in the middle regions of the images, rather than the edges). So, we will have `(m x n)- 8` feature vectors in the FVS, all 25 dimensional. Projecting the FVS to the 25 dimensional EVS simply means to perform the following matrix multiplication

$$(VS)((m \times n - 8) \times 25) . (EVS)(25 \times 25) = (FVS)(m \times n - 8) \times 25$$

Figure below summarises the steps that have been followed so far in the method.



Flowchart for building the feature vector space

Function `find_FVS()` determines the feature vector space for us. The function is similar to `find_vector_set()`, but extracts overlapping blocks from the difference image.

```
def find_FVS(EVS, diff_image, mean_vec, new):

    i = 2
    feature_vector_set = []

    while i < new[0] - 2:
        j = 2
        while j < new[1] - 2:
            block = diff_image[i-2:i+3, j-2:j+3]
            feature = block.flatten()
            feature_vector_set.append(feature)
            j = j+1
        i = i+1

    FVS = np.dot(feature_vector_set, EVS)
    FVS = FVS - mean_vec
    return FVS
```

The feature vectors for the pixels now lie in a space where their variance has been maximized. This will help the subsequent step of clustering to better categorize the pixels into the 2 classes – changed pixels (pix_c) and unchanged pixels (pix_u).

3. Clustering of the feature vector space, and change map

The feature vectors for the pixels carry information whether the pixels have characteristics of a changed pixel or an unchanged one. Having constructed the feature vector space, we now need to cluster it so that the pixels can be grouped into two disjoint classes. We will be using the K-means algorithm to do that. Thus each pixel will get assigned to a cluster in such a way that the distance between the cluster's mean vector and the pixel's feature vector is the least. Each pixel gets a label from 1 to K, which denotes the cluster number that they belong to.

```
from sklearn.cluster import KMeans
from collections import Counter

def clustering(FVS, components, new):

    kmeans = KMeans(components, verbose = 0)
    kmeans.fit(FVS)
    output = kmeans.predict(FVS)
    count  = Counter(output)

    least_index = min(count, key = count.get)
    change_map  = np.reshape(output,(new[0] - 4, new[1] - 4))
    return least_index, change_map
```

During our experiments, it was empirically found that the best results were obtained with K = 3. Thus the argument `components` in `clustering()` will be 3. Remember, even though we have to do divide the pixels into 2 categories, we have chosen K = 3, instead of 2. Now how do we decide which of these clusters contains the pixels that belong to the changed class? It can be postulated that the cluster which contains the lowest number of pixels (denoted by variable `least_index`) is the cluster denoting the changed class, since the background remains more or less the same in satellite images and the changes occurred are comparatively less. Also, the mean of this cluster will be the highest. The reason behind the highest value of mean for that cluster is that the values of the difference image pixels in a region where some changes have occurred are higher than the values of pixels in the regions where there is no change.

Thus, in conclusion, the cluster with the ***lowest number of pixels***, and also the ***highest mean*** is the cluster belonging to the changed class.

With this information, we will now build a change map – a binary image to show the output of change detection. We have chosen to keep the background black and will show the changes in white, i.e., intensity value of those pixels will be 255. You can do the reverse as well. Thus

$$change\_map(i,j) = \begin{cases} 255, & \text{if (i, j)} \in pix_u \\ 0, & \text{otherwise} \end{cases}$$

The following Python lines achieve this task:

```
change_map[change_map == least_index] = 255
change_map[change_map != 255] = 0
imsave('change_map.jpg', change_map)
```

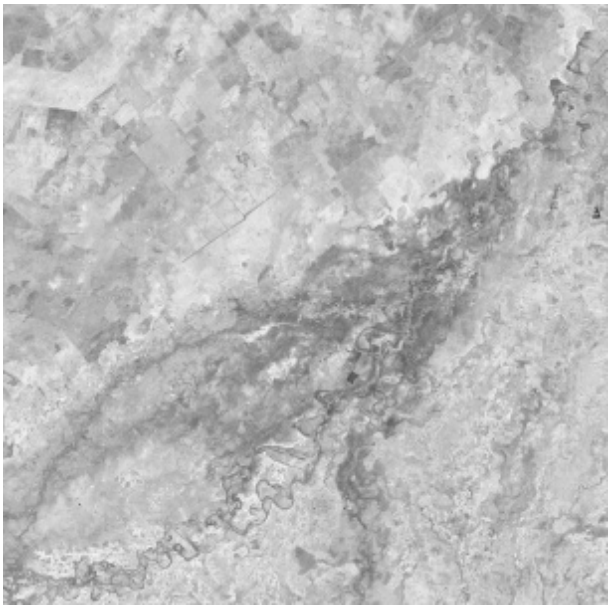
The entire Python code for performing change detection on satellite images is as follows:

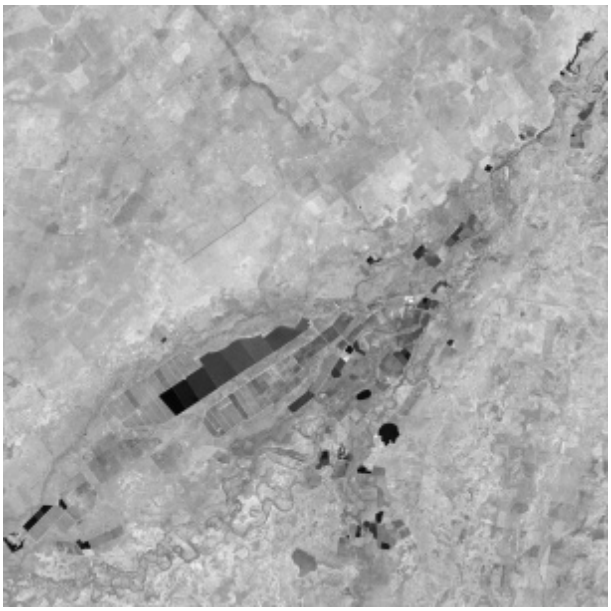
```
import cv2
import numpy as np
from sklearn.cluster import KMeans
from sklearn.decomposition import PCA
from collections import Counter
from scipy.misc import imread, imresize, imsave

def find_vector_set(diff_image, new_size):

    i = 0
    j = 0
    vector_set = np.zeros(((new_size[0] * new_size[1]) / 25,
25))
    while i < vector_set.shape[0]:
        while j < new_size[0]:
            k = 0
            while k < new_size[1]:
                block   = diff_image[j:j+5, k:k+5]
                feature = block.ravel()
                vector_set[i, :] = feature
                k = k + 5
            j = j + 5
        i = i + 1

    mean_vec   = np.mean(vector_set, axis = 0)
    vector_set = vector_set - mean_vec
    return vector_set, mean_vec

def find_FVS(EVS, diff_image, mean_vec, new):

    i = 2
    feature_vector_set = []

    while i < new[0] - 2:
        j = 2
        while j < new[1] - 2:
            block = diff_image[i-2:i+3, j-2:j+3]
            feature = block.flatten()
            feature_vector_set.append(feature)
            j = j+1
        i = i+1

    FVS = np.dot(feature_vector_set, EVS)
    FVS = FVS - mean_vec
    print "\nfeature vector space size", FVS.shape
    return FVS

def clustering(FVS, components, new):

    kmeans = KMeans(components, verbose = 0)
    kmeans.fit(FVS)
```

```
    output = kmeans.predict(FVS)
    count  = Counter(output)

    least_index = min(count, key = count.get)
    change_map  = np.reshape(output,(new[0] - 4, new[1] - 4))
    return least_index, change_map


def find_PCAKmeans(imagepath1, imagepath2):

    image1 = imread(imagepath1)
    image2 = imread(imagepath2)

    new_size = np.asarray(image1.shape) / 5 * 5
    image1 = imresize(image1, (new_size)).astype(np.int16)
    image2 = imresize(image2, (new_size)).astype(np.int16)

    diff_image = abs(image1 - image2)
    imsave('diff.jpg', diff_image)

    vector_set, mean_vec = find_vector_set(diff_image,
new_size)
    pca     = PCA()
    pca.fit(vector_set)
    EVS = pca.components_

    FVS     = find_FVS(EVS, diff_image, mean_vec, new_size)
    components = 3
    least_index, change_map = clustering(FVS, components,
new_size)

    change_map[change_map == least_index] = 255
    change_map[change_map != 255] = 0

    change_map = change_map.astype(np.uint8)
    kernel     = np.asarray(((0,0,1,0,0),
                             (0,1,1,1,0),
                             (1,1,1,1,1),
                             (0,1,1,1,0),
                             (0,0,1,0,0)), dtype=np.uint8)
    cleanChangeMap = cv2.erode(change_map,kernel)
    imsave("changemap.jpg", change_map)
    imsave("cleanchangemap.jpg", cleanChangeMap)

if __name__ == "__main__":
    a = 'image1.jpg'
    b = 'image2.jpg'
    find_PCAKmeans(a,b)
```

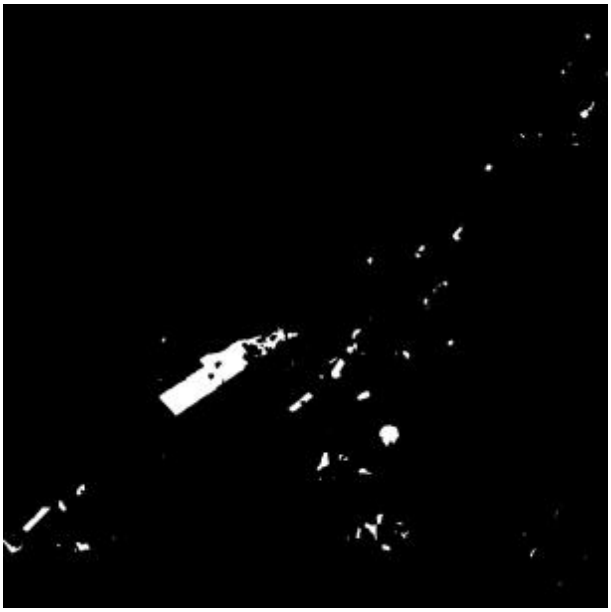
Illustrative example

The following 2 images show the Cubbie Station at two different time stamps – September August 1987 and August 2013. Cubbie Station is an irrigation project in Australia, and its construction had started in the 1990s.





The difference image generated by the pixel by pixel subtraction and the change map made by the above methodology have been shown below.



You can view other examples in the repository at the Github link provided at the end and can verify if you are able to reproduce the results.

References

Concluding remarks

Hope everyone reading this was able to grasp the method adopted and was able to follow the code without much difficulty. We would like to give some additional pointers to you regarding this application:

1. One of the most fundamental requirements of change detection is the *registration* between the two input images. It implies that the images should have perfect alignment between them, otherwise change detection would give many false alarms. Image registration is an open research in itself and we encourage you to explore about it. Most of the images in the USGS image

archives are thankfully registered to each other and hence we could directly perform change detection on them.

2. To evaluate the performance of the method, we need to compare the generated change map with the ground truth for the 2 images. Ground truths are not easy to be identified and need considerable efforts and survey of the locations from the remote sensing agencies. Rather, users of the application would prefer some alternative tests which are somehow able to quantify the performance of the method, for example, by introducing artificial changes and assessing whether the method could identify them. You could think about other such tests which would be a desirable extension to this work.

Readers are also encouraged to research about other methods to perform change detection on images, for example, with wavelets, involving Bayes' theory, etc.

The full implementation of the followed approach, the sample USGS image pair and the generated change map can be downloaded from GitHub link here. You can also run the code on other pair of images available in dropbox.

If you liked the post, follow this blog to get updates about upcoming articles. Also, share it so that it can reach out to the readers who can actually gain from this. Please feel free to discuss anything regarding the post. I would love to hear feedback from you.

Happy machine learning 🙂

CHANGE DETECTION **CHANGE MAP** **DIFFERENCE IMAGE** **K-MEANS CLUSTERING** **MULTI-TEMPORAL IMAGES** **PRINCIPAL COMPONENT ANALYSIS** **PYTHON IMPLEMENTATION** **REMOTE SENSING** **SATELLITE IMAGERY** **UNSUPERVISED LEARNING**

Published by ML bot1

View all posts by ML bot1