MODEL FOR LAND COVER ESTIMATION USING UNSUPERVISED MACHINE LEARNING ON GOOGLE MAPS COLOR IMAGES

Rajat Subhra Bhowmick [1], Anil Kumar [1 *], Gagan Deep Singh [1], Shashi Kumar [2]

[1] University of Petroleum and Energy Studies, Dehradun, India - sendrajat3060@gmail.com, anil.kumar@ddn.upes.ac.in, gagan@ddn.upes.ac.in
[2] Indian Institute of Remote Sensing, Dehradun, India - shashi@iirs.gov.in

Commission V, SS: Emerging Trends in Geoinformatics

KEY WORDS: Unsupervised learning algorithms, CORINE nomenclature, Google Static Maps API, k-means clustering

ABSTRACT:

Remote sensing data and satellite images are broadly used for land cover information. There are so many challenges to classify pixels on the basis of features and characteristics. Generally it is pixel classification that required the count of pixels for certain area of interest. In the proposed model, we are applying unsupervised machine learning to classify the content of the input images on the basis of pixels intensity. The study aims to compare classification accuracy of different landscape characteristics like water, forest, urban, agricultural areas, transport network and other classes adapted from CORINE (Coordination of information on the environment) nomenclature. To fulfil the aim of the model, accessing data from Google map using Google static API service which creates a map based on URL parameters sent through a standard HTTP (Hyper Text Transfer Protocol) request and returns the map as an image which can be display on any graphical user interface platform. The Google Static Maps API returns an image either in GIF, PNG or JPEG format in response to an HTTP request. To identify different land cover/use classes using k-means clustering. The model is dynamic in nature that describes the clustering as well formulate the area of the concerned class or clustered fields.

1. Introduction

Any developing nation must have enough information on various interrelate aspects of its activities in order to make correct decisions for not only the development of the nation but at the same time achieve sustainable development. Land use or land cover is one such domain, but knowledge about these domains has become increasingly important as the nation plans to overcome the problems of environmental pollution, destruction of forest, uncontrolled developments, loss of prime agricultural lands [1].

Land cover information is required in the analysis of environmental processes and issues that must be comprehended if living conditions are to be enhanced along with the technical developments. Land cover estimation reflects to the vegetative characteristics or manmade constructions on the land's surface. In recent times the available of satellite data has triggered a series of development of many complex tool related to geographic information system or geographical information system (GIS). Many of these tools used mainly supervised classification technique for estimating land cover percentage. Satellite image classification is a process of grouping pixels into meaningful classes [6].

The data of any region on which most of these tool works are not easily available. At the same time much larger in size due to the fact that data is available only in different bands. The bands are again not easy to understand unless we use specific tool feature for understanding the information available for that region.

* Corresponding Author

In recent times Google maps are very useful in getting important information about a region and the data is also easily available. So we use the Google maps for colour image data of a specific region and then use unsupervised machine learning algorithms for estimation of land cove of that region in the process calculating the actual area of land in sq. Km covered in different type of land.

For any advance data analysis or machine learning task nowadays python [13] emerges as one of the best programming language to be used. We have used python to support our model. It has supported python packages like numerical mathematics extension (numpy) [14] which is very useful in handling many of the matrix manipulation tasks very efficiently. Matplotlib [15] is another package which provides an object-oriented API for embedding plots into applications. OpenCV [16] and python Image helps to do various image operations which are very helpful in pre-processing of Google map image. Along with these libraries pyQt helps to create general-purpose graphical user interface (GUI) for the user.

2. Methodology

In the present work, we have built a model which consists of four major stages. First stage is to get data from Google map using Google static API service returns a map as an image which can be displayed on any graphical user interface platform. Figure. 1 depicts the above process. So using the Google API we can have an image of a region of our interest by specifying different parameters given in the request. First stage also includes some image enhancement and cleaning task on the image downloaded from Google map. It provides better visualisation and increases the effectiveness of machine learning algorithms. Second stage is the most important in the model in which we used unsupervised machine learning algorithm for clustering on RGB component of the image. Many techniques have been proposed in the literature of cluster analysis [9, 10].

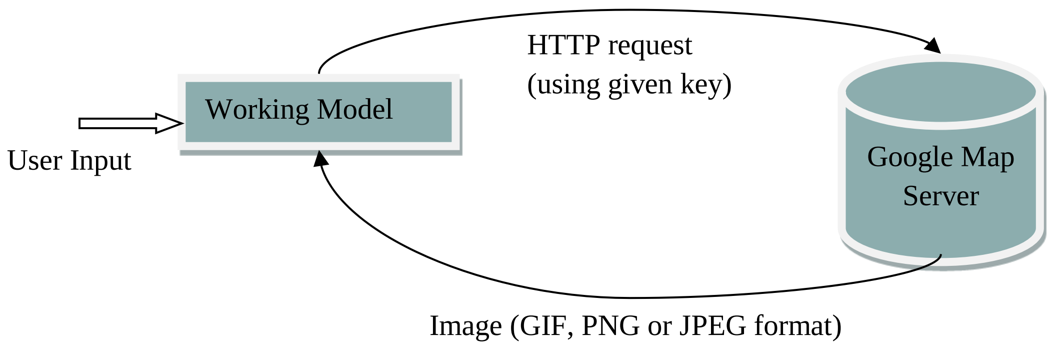

Figure 1. Process of obtaining image from Google server

The RGB colour model is a representation of an image pixel by the combination of three colour model in which red, green and blue colours are added together in many possible ways to reproduce a broad array of colours. This is in agreement with the tristimulus theory of colour [12, 13] according to which the human visual system acquires colour imagery by means of three band pass filters whose spectral responses are tuned to the wavelengths of red, green, and blue. Any image in digital system consists of a set of pixels and each pixel has its own RGB values. The set of these RGB values of all pixels in the image consist the data set upon which we have applied our machine learning algorithm. As mention earlier the machine learning technique used is unsupervised, which is used for drawing certain inferences from datasets consisting of input data without labelled responses. The algorithm used is K-means [4, 11] which is a partition-based cluster analysis method and it is very effective algorithm for finding clusters for a given dataset. Park et al. [12] applied this algorithm to a pattern space representation of RGB coordinates. The k-means is susceptible to local optima. So we have to re-run the k-means several times and using backtracking we have to sure about our initialisation values.

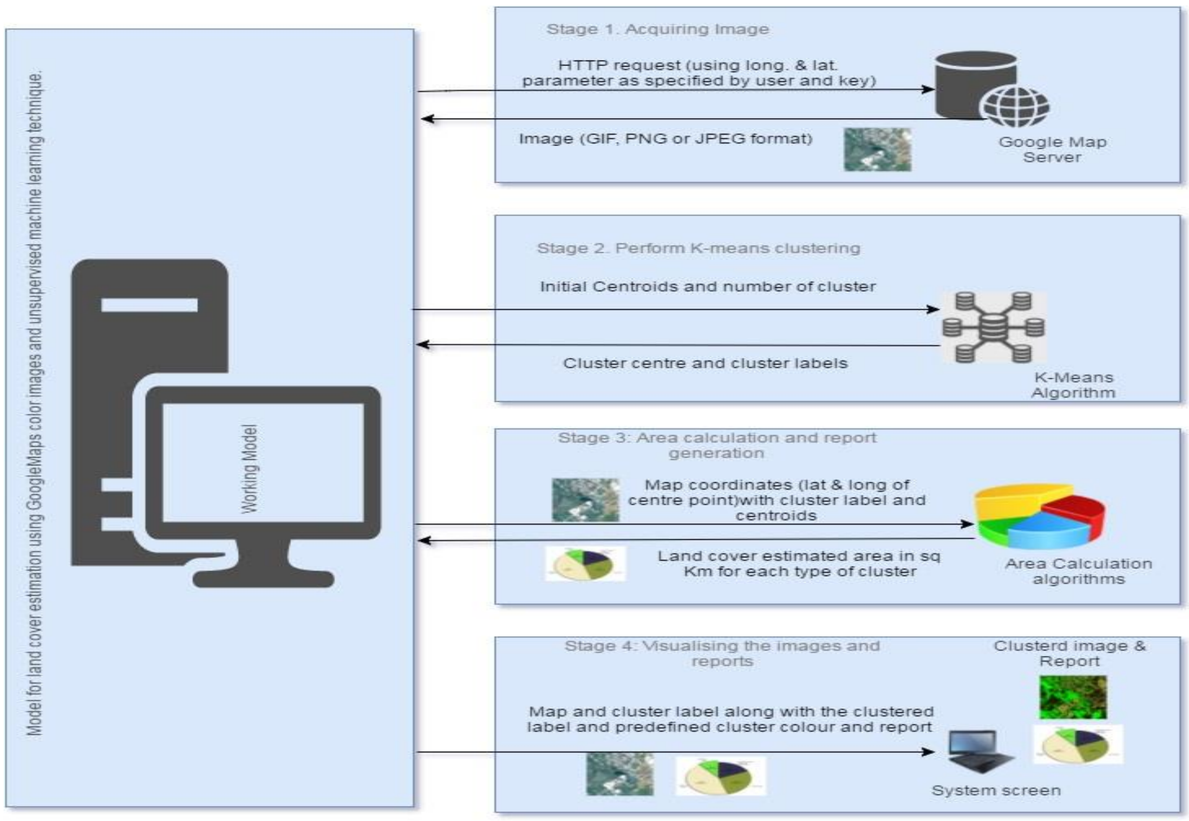
Model for land cover estimation using GoogleMaps color images and unsupervised machine learning technique.
Working Model
Stage 1. Acquiring Image
HTTP request (using long. & lat. parameter as specified by user and key)
Image (GIF, PNG or JPEG format)
Google Map Server
Stage 2. Perform K-means clustering
Initial Centroids and number of cluster
Cluster centre and cluster labels
K-Means Algorithm
Stage 3: Area calculation and report generation
Map coordinates (lat & long of centre point)with cluster label and centroids
Land cover estimated area in sq Km for each type of cluster
Area Calculation algorithms
Stage 4: Visualising the images and reports
Clusterd image & Report
Map and cluster label along with the clustered label and predefined cluster colour and report
System screen

Figure. 2 Depicting the stages in the model

The last stages of our model consist of the calculation of the total area given latitude and longitude value of the centre point of the map and the visualisation of clustered image. The calculation of the area is done in sq. kilometres. Figure. 2 visualise all the major steps in the model in order. Out of these four stages the second stage consist the algorithms which cluster's the image from obtained from Google map. The first stage helps in getting the image and enhancing it. The second last stage gives the estimated area under each region type and the last stage helps in visualizing the overall result of the model.

3. Modeling:

3.1 Getting the image: To obtain the Google Maps image we used the Google Static Maps API which lets us embed a Google Maps image in our model. The Google Static Maps API returns an image of a map in response to an HTTP request via a URL. For each request, we can specify the location of the map image, the size of the map, the zoom level of map and the type [5]. The location of map in our case is the longitude and the latitude of central point of an interested region. The centre of the map is equidistant from all corners of the map. This parameter takes a location as comma-separated (latitude and longitude) pair (e.g. "30.3239, 78.0654"). The zoom level is another parameter which determines the magnification level of a map. This parameter takes an integer value corresponding to the zoom level of the region desired. In our case the zoom level is set to 15 as this zoom level works better for viewing the land spread and in the meantime covers a critical range of a region which we are keen on. A zoom level less than this is not suitable for clustering as this gives very less information about the ground cover vice versa if it is greater than 15 the area spread is less and clustering is more susceptible to noise. Size of a map is the rectangular dimensions of a map image. The parameter can be set in the form of (horizontal value x vertical value) where both values are integer number for example, 640x480 defines an image of a map which has 640 pixels in width and 480 pixels in height. This parameter is also affected by the scale parameter; the final output is the product of the size and scale parameter values. For our case we have opted to get 640 x 640 image. Scale is an optional parameter affects the number of pixels that are returned. Scale equal to two returns twice as many pixels as scale equal to one while retaining the same area which means the contents of the map don't change, in our case the scale is set to one. We have opted for PNG images from Google Static Maps API. Map type defines the type of map to construct from server. We are interested in satellite image so we select our map type as satellite. The original file retrieved from Google Map server contains the Google logo which is not suitable for image clustering. So we need to truncate the lower portion for our case the image size reduced to 620 x 640. The other task which is done in this stage is to enhance the image quality. The enhancement is done mainly for visualisation purpose and will not affect the machine learning process significantly. The three features of image colour, brightness, contrast are increased to 1.2, 1.25, and 1.5 respectively. This completes the first stage of our model and we have the enhanced image from Google Map to work with, for the next stages of our model, Figure. 3 visualise the same.

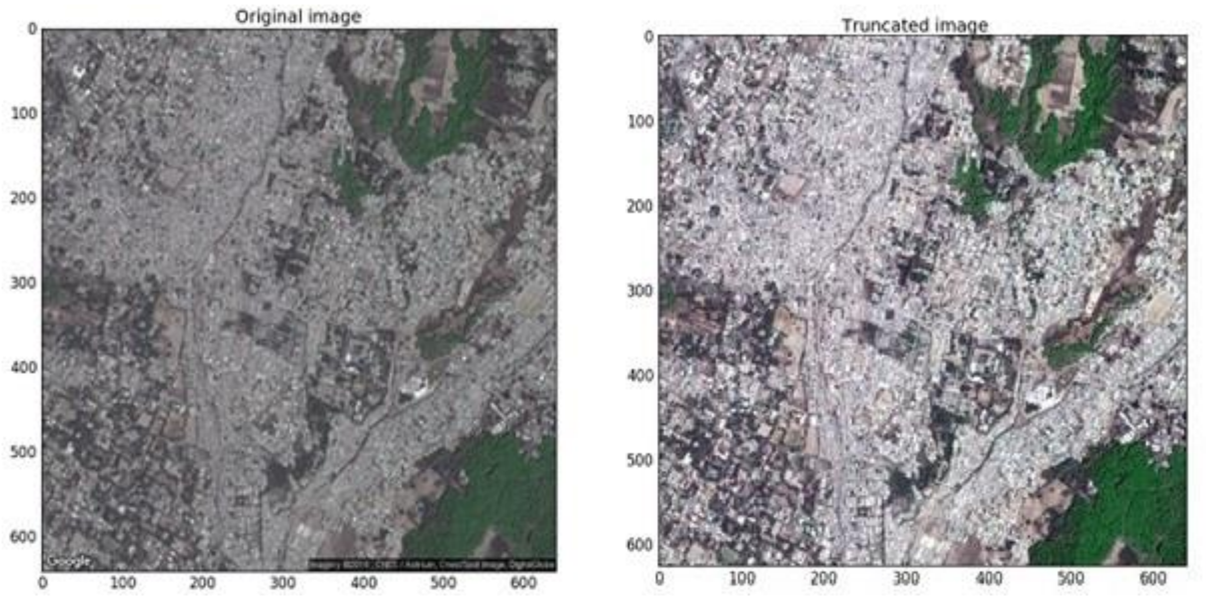

Figure. 3 The original image from the Google server and the image after pre-processing

3.2 Clustering Process

The clustering process is an unsupervised machine learning technique in which one has to generate partitions without any a priori knowledge [17]. The algorithm has been described above, the basic cons of k-means is it is locally optimum so depending upon our initial points the algorithm converges to a certain result. The whole clustering process after obtaining the image with RGB components is explained in Algorithm 1.

Algorithm 1. Clustering process (image[x, y, z])

```
//where x * y is the size of image and z represent the values of RGB in each pixel.

Step 1   Enhance the image by increasing the brightness, colour and contrast by 1.25, 1.2, and 1.5 respectively taking 1 as original value.
Step 2   Set k=4 as the number of division we need to cluster.
Step 3   Set initial centroid in init_centroid [k, z].
         // The dimension of init_centroid is k*z matrix as "k" for number of cluster and "z" represent the three feature which given RGB value in each pixel.
Step 4   Set the number of iteration in n_int equal to an integer, as the higher iteration will result in increased complexity so n_int set to 1.
Step 5   Get final_centroids[k, z] and cluster_assignment[x, y, 1] by running the K-means algorithm k-means(image, init_centroid, n_iter).
         // the dimension of cluster assignment is x*y*1 which gives the cluster value between 0 to k-1 for each pixel in the image.
Step 6   Convert the final_centroids from float type to integer type representation.
Step 7   Get the original clustered image using the algorithm get_image_assigned_cluster(image, final_centroids,cluster_assignment) into org_clust_img[x, y, z].
Step 8   Change the final_centroids to false colour centroid color_centroids to get colour cluster image for proper visualisation of the vegetation.
Step 9   Get the colour clustered image using the algorithm get_image_assigned_cluster(image,color_centroids, cluster_assignment) into color_clust_img[x,y,z].
Step 10  Save the color_clust_img and the org_clust_img.
```

3.3 K-means specification

Now we have the data set to apply k-means algorithm but we need to access convergence to tell if the k-means algorithm is converging. We can look at the cluster assignments and see if they stabilize over time. We'll be running the algorithm until the cluster assignments stop changing or produce very little change. To be extra safe, and to assess the clustering performance, we'll be looking at an additional criterion: the sum of all squared distances between data points and centroids which is given in equation (1):

$$J(V) = \sum_{i=1}^{c}\sum_{j=1}^{c_i}((||x_i - \mu_j||)) \quad (1)$$

Where, '$||x_i - \mu_j||$' is the Euclidean distance between x_i and μ_j. 'c' is the number of cluster centres and 'c_i' is the number of data points in i^{th} cluster. The μ_j cluster centroid can be calculated by equation (2).
For every cluster j, set

$$\mu_j = \frac{\sum_{i=1}^{m} 1\{c^{(i)}=j\}x^{(i)}}{\sum_{i=1}^{m} 1\{c^{(i)}=j\}} \quad (2)$$

Where, '$c^{(i)}$' represents the number of data points in i^{th} cluster.

Suppose that we have n data points (a_1, b_1, c_1), (a_2, b_2, c_2), (a_3, b_3, c_3), …, (a_n, b_n, c_n), then the centroid is calculated as ($\Sigma\, a_i / n$, $\Sigma\, b_i / n$,$\Sigma\, c_i / n$).

The smaller the distances, the more homogeneous the clusters are. In other words, we'd like to have "tight" clusters. This is called heterogeneity. As we already know K-means converges local optimum we need to find the correct initial centroids for right clustering process. One effective way to counter this tendency is to use a smart initialization.

This method tries to spread out the initial set of centroids so that they are not too close together. It is known to improve the quality of local optima and lower average runtime.

In general, we should run k-means at least a few times with different initializations and then return the run resulting in the lowest heterogeneity. After many runs of k-means and accessing the heterogeneity we found the initial centroid parameter for clustering shown below in Table 1.

The next step is to perform K-means with obtained linear data set and initial centroid. The K-means algorithm executes two main steps. First is assign closest cluster centre for every data point "i" and for every cluster j.

Sometimes this called "Map" step which is given equation (3). For every i, set

$$c^{(i)} = \arg\min_{j}\left\|x^{(i)} - \mu_j\right\|^2 \quad (3)$$

Second is to recalculate the mean of each cluster, fitfully studied as "Reduce" step which is given in equation (2). K-means algorithm takes another parameter which is maximum iteration as we have re-run our K-means algorithm

Table 1. Initial centroid for cluster to obtained after series of runs of K-means.

Cluster Number	Red	Green	Blue

Cluster 1	31	64	41
Cluster 2	192	187	190
Cluster 3	133	124	131
Cluster 4	81	86	83

many times to get initial centroids so we will run the K-means map-reduce only once so as to reduce the complexity of clustering of data points, optimising our unsupervised machine learning clustering process, time complexity in comparison with supervised machine learning classification process. The K-means returns two parameters first is the final centroids and second, cluster label for each data point between [1....K] where k is number of cluster. The Algorithm 2 formulates the k-means function step by step.

Algorithm 2: K-means (image[x, y, z], init_centroid [k, z], n_iter)

```
Step 1    Set centroids [k, z] equal to init_centroid [[k, z],1] equal to None and itr = 0.
Step 2    Repeat step 3 to 6 until $itr \leq n\_itr$
Step 3    Make cluster assignments using nearest centroids and set it in cluster_assignment.
Step 4    Compute a new centroid for each of the k clusters, averaging all data points assigned to that cluster and set it in
          centroids.
Step 5    Check for convergence: if none of the assignments changed move to step 7.
Step 6    Increment the iteration by 1.
Step 7    Return centroids [k, z], cluster_assignment[x, y, 1].
```

3.4 Clustering and Visualization

So, now we have to assign colour to each data point so as to generate better intuition about the land covering. As the colour components are 8-bit integer each it has value from 0-255 but according to K-means algorithm it may produce in between float values so we need to convert all centroids into integer values. As cluster label give the clusters which any data point belong after clustering we can reassign the data points RGB components value to cluster centre.

Then we can recast the data points back into the image so to get the notion of how each of the pixels belongs to a cluster. Thus by changing the centroid we can simultaneously change the colour of each cluster. Table 2 represent the colour code for each cluster. Algorithm 3 is showing how the normal image is converted to clustered image. Now because as we have chosen max-iteration in K-Means algorithm to very less and the initial centroid after lots of iteration we are notably sure about what each cluster centre do represent. After running many iterations of K-means algorithm with different initialisation we found what each cluster represents, the results are shown in Table 2.

Algorithm 3: Get_image_assigned_cluster(image [x, y, z], centroids[k, z], cluster_assignment [x, y, 1])

//where image has dimension x, y, z and centroids having dimension k and z and cluster_assignment has same number elements as original image dimension but only one component for each pixel which is cluster value

```
Step 1    Initialize image_copy[x, y, z] to None.
Step 2    for i in range of x do
Step 3    for j in range y do
Step 4    Set image_copy [i, j] = centroids [cluster_assignment [i, j, 1]].
Step 5    end for in step 3.
Step 6    end for in step 2.
Step 7    Return image _copy.
```

Table 2. Colour component for each cluster to superimpose RGB component.

Cluster Centroid RGB components	**Type it represent**	**Color Code**
centroids[0]=[50,240,100]	#Vegitation/Forest	🟢
centroids[1]=[255,255,255]	#Buildings/Houses	⚪
centroids[2]=[37,120,120]	#Unused lands	●

centroids[3]=[0,30,30]	#Unclassified_objects/ (Shadows)/Barren lands /Water bodies	●

Now using the above algorithms we can show the clustered image (depicted in **Fig. 4)**. We can also visualise from **Fig. 5** the superimposed RGB components of the original centroids with our own stipulated cluster colours. The two images have shown to visualise the effect of clustering in original image.

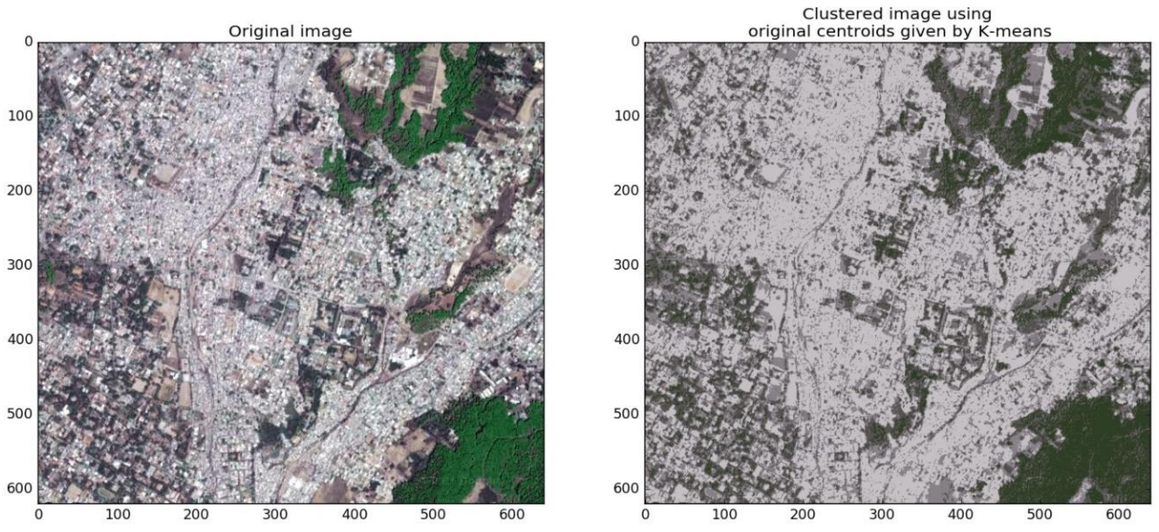

Figure. 4. Comparison between the original image and clustered image using original centre

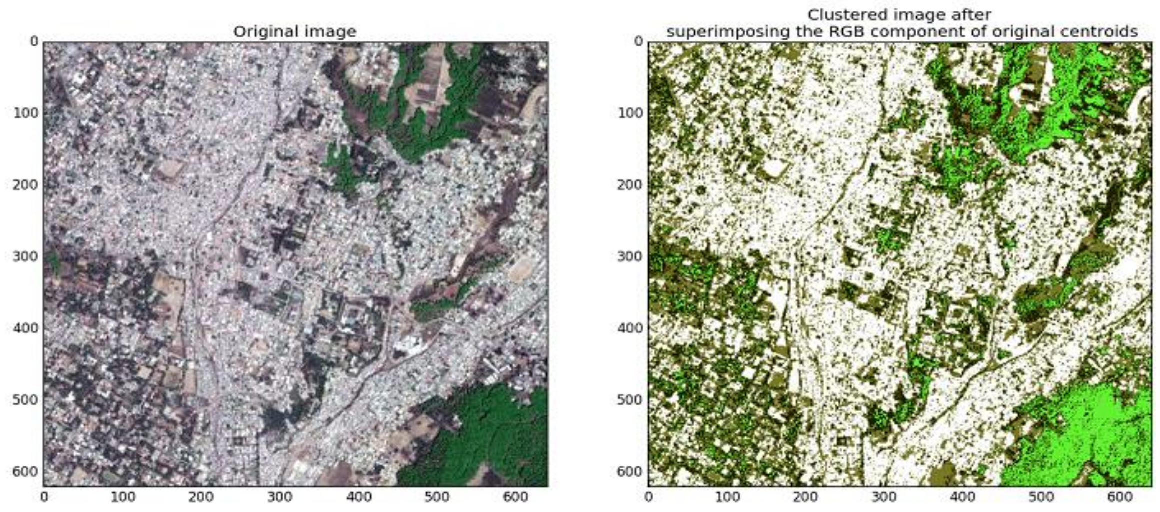
Original image
Clustered image after superimposing the RGB component of original centroids
0
100
200
300
400
500
600

Figure. 5. Comparison between the original image and the clustered image after superimposing the RGB component defined internally.

3.5 Area Estimation & Results

The next phase is to estimate the area under each cluster so as to predict the area each type of land cover in square kilometre. For acquiring image we have specified central coordinate so we need to find the corners value of longitude and latitude to estimate the area under the map image obtained from Google Map. We have used Mercator projection for calculating the corners of the image. The Mercator projection was invented by Gerardus Mercator, a Flemish mapmaker. His name is a Latinized version of Gerhard Kramer [7, 8]. As we have also truncated the lower image portion for removing Google logo. We recalculated the corners, so as to calculate the distance between sides which in turn give area. After getting the total area we create a histogram on number of cluster and number of corresponding pixels belongs to each cluster in the cluster image. We normalized the histogram to have total portion of pixels equal to unity, then multiplied each proportion of histogram to get total area under each cluster or corresponding type of vegetation.

The Algorithm 4 explains the whole process step by step.

```
Algorithm 4: Calculate Area and Report (image[x, y, z], map_point [2], cluster_assignment[x, y, 1], color_centroids [k, z])

//where x and y is the size of image and z represent the values of RGB.

//map_point represents the longitudinal and latitudinal value of the centre point of the   downloaded map image

// cluster_assignment[x, y, 1] and color_centroids [k, z] are the cluster value of each pixel, and colour value of centroids as defined in Algorithm 1.

    Step 1      Calculate the longitudinal and latitudinal point of the corners given the central point using Mercator
                projection method and given zoom level =15 and set it into map_points [4, 2].
          // Four corners having two value latitude and longitude.

    Step 2      Calculate area using map_points.

    Step 3      Calculate normalised histogram norm_hist[k] of cluster_assignment vector.
          // as it would have value between 1 to k so histogram contain k values.

    Step 4      Multiply norm_hist with total area to give each type of area coverage in area_type[k].

    Step 5      Relate to color_centroids [k, z] and area_type[k] to visualise the report.
```

After getting the areas under each cluster we have used pie chart to reveal the area under each cluster, the Figure. 6 demonstrate the area under each type of land cover in the region of map which is shown in Figure. 3.

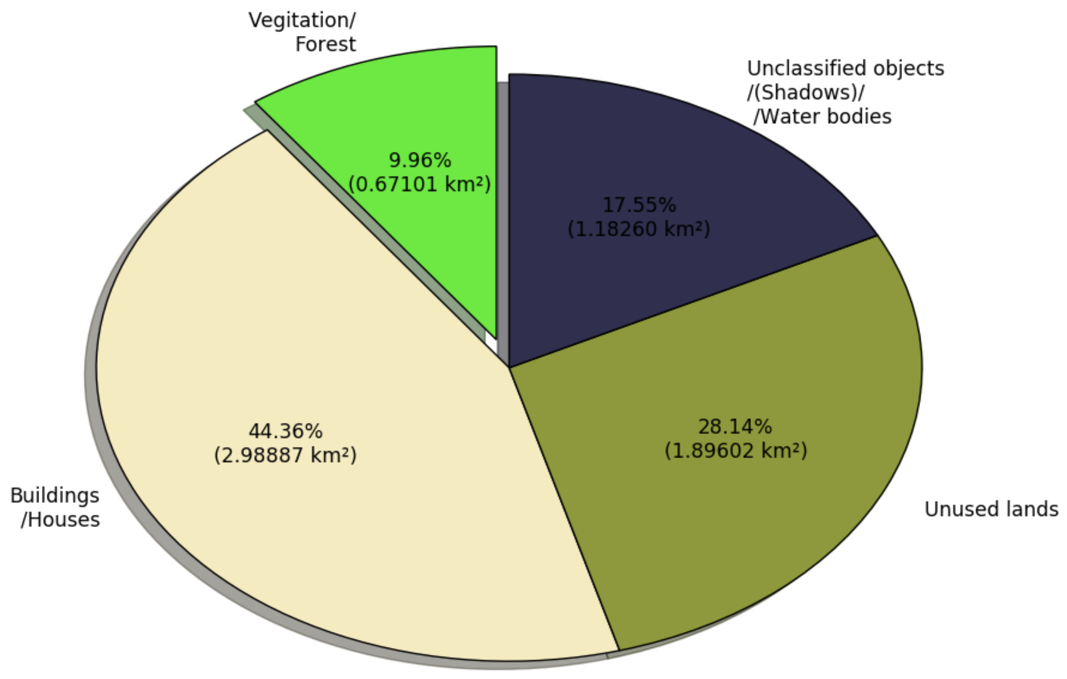

Figure. 6 The area under different types of land cover.

4. Conclusions & Future Scope

The model we have created can predict the land cover estimation by downloading maps from Google Maps given the longitude and latitude of any specific region center. Then performing unsupervised machine learning (K-means) on that color image with specified initial centroids and number of cluster set to 4, to give a clustered image which depict the land cover in that region. The model uses K-means algorithm whose complexity is much lesser than other clustering algorithms like Gaussian Mixture Model so it is quite fast to operate and get result. The model is also useful as it does not need to set any training data to predict the land cover which in case many traditional land cover estimation classification technique needed to specify. So, this model can provide useful information to the groups who are not much familiar with supervised machine learning techniques.

The future works of this model mainly consist of two criteria. First is improving the accuracy of predicting the area covered by using more sophisticated unsupervised machine learning algorithms like Guassian Mixture Model and at the same time reduce the complexity of the whole process by predicting more appropriate initial centroids and covariance. The second is to increase the ability to recognize more types of land cover than these basic 4 – types which can be done if we have more information or bands available (other than RGB component) in image data thus by improving significance of the model to next level.

5. References

[1]. James A. Shine and Daniel B. Carr, "A Comparison of Classification Methods for Large Imagery Data Sets", JSM 2002 Statistics in an ERA of Technological Change-Statistical computing section, New York City, pp.3205-3207, 11-15 August 2002.

[2]. G. Wyszecki and W.S. Stiles, Color Science: Concepts and Methods, Quantitative Data and Formulae, Wiley, New York, NY, 1982.

[3]. R.W.G. Hunt, Measuring Colour, 2nd Ed., Ellis Horwood Ltd. Publ., Chichester, UK, 1987.

[4]. S. Deelers and S. Auwatanamongkol, “Enhancing K- Means Algorithm with Initial Cluster Centers Derived from Data Partitioning along the Data Axis with the Highest Variance,” International Journal of Computer Science, Vol. 2, Number 4 S.T. Bow, Pattern Recognition and Image Preprocessing, Marcel Dekker, Inc., New York, NY, 1992.

[5]. Anders Karlsson, 2003. “Classification of high resolution satellite images”, August 2003, http://infoscience.epfl.ch/record/63248/files/TPD_ Karlsson.pdf.

[6]. Isreal Robert, 2003-01-20, “Mercator's Projection”, http://www.math.ubc.ca/~israel/m103/mercator/mercator.html

[7]. Dana,P. H. "Map Projections. Available at " http://www.colorado.edu/geography/gcraft/notes/mapproj/mapproj_f.html

[8]. M.R. Anderberg, Cluster Analysis for Applications, Academic Press, New York, NY, 1973.

[9]. S.-T. Bow, Pattern Recognition and Image Preprocessing, Marcel Dekker, Inc., New York, NY, 1992

[10]. J. McQueen, \Some Methods for Classification and Analysis of Multivariate Observations," Proc. of the 5th Berkeley Symp. On Math. Stat. and Prob., Vol. 1, pp. 281-296, 1967.

[11]. S.H. Park, I.D. Yun, and S.U. Lee, Color Image Segmentation Based on 3-D Clustering: Morphological Approach, "Pattern Recognition, Vol. 31, No. 8, pp. 1061 1076, Aug. 1998.

[12]. Python Software Foundation. Python Language Reference, version 2.7. Available at http://www.python.org

[13]. Stéfan van der Walt, S. Chris Colbert and Gaël Varoquaux. The NumPy Array: A Structure for Efficient Numerical Computation, Computing in Science & Engineering, 13, 22-30 (2011), DOI:10.1109/MCSE.2011.37

[14]. John D. Hunter. Matplotlib: A 2D Graphics Environment, Computing in Science & Engineering, 9,90-95(2007), DOI:10.1109/MCSE.2007.55

[15]. Bradski, G., opencv_library, Dr. Dobb's Journal of Software Tools, 2008-01-15

[16]. D. Koller and N. Friedman. Probabilistic Graphical Models: Principles and Techniques - Adaptive Computation and Machine Learning. The MIT Press,2009.