Project in Practical Machine Learning

Radar image clustering

Marko Hassinen
014029585

Helsinki 20.03.2016

University of Helsinki
Department of Computer Science

Terms of Reference

A report submitted in fulfilment of the requirements for Course 582739 Project in Practical Machine learning, Deparment of Computer Scinece, University of Helsinki.

Abstract

Weather Radar image clustering is the topic of this project. This project also investigated if it is possible to recognize weather types using predictive unsupervised machine learning algorithms to form clusters of weather radar images. This report explains which tools and methods were used and reports results that were gathered. The machine learning algorithms implemented in this project were k-means, k-means++ and k-medoids. The Project also

Introduction

The goal of this project was to find clusters within weather radar images using different predictive clustering methods and identify weather types based on the clusters. Also an additional goal was to compare the results and evaluate the performance of the clustering methods.

The aim of clustering is to divide data into subsets to recognize structure or patterns within the data. For example divide n amount of images into k clusters. This project has implementations of predictive clustering algorithms. Predictive clustering focuses on predicting one possible way to make clusters of the data in way where similar data points belong to same cluster. Clustering relates to machine learning because clustering is considered to be part of unsupervised machine learning.

The machine learning algorithms implemented in this project were k-means, k-means++ and k-medoids. Each of these algorithms tries to group similar data points into same clusters. These algorithms have two basic steps which they iterate over grouping step and update step. Initialisation step comes before grouping and updating where initial cluster centers will be set. Each of these algorithms has some kind of randomized manner to get initial cluster centers. K-means and k-medoids have similar initialization step by k-means++ has different kind of initialization step. After that grouping will occur where every data point will be grouped with the most similar cluster center. Update step will try to find a new possible center and this is the part where k-means and k-medoids differ. If update step does not change the cluster center, the algorithm will converge and return results.

The machine learning algorithms of this project have been implemented in a manner that their results are possible to visualize in a website. A Hosting platform was used to simulate the usage of the project in a website. Only a simulation of the usage was possible, because image clustering problem is resource and calculation time demanding. Only a small portion of data is analyzed in the hosting platform to achieve faster computation time. The hosting platforms data set updates periodically.

The results and data for this project were gathered around late winter and early spring of the year 2016.

Methods

Tools used in this project were programming language Python 3.4.3, and its modules NumPy, SciPy, MatPlotLib and OSWLib. Web application framework Django 1.9.4 was used for web development and the project was hosted using the services of an online Integrated Development Environment and Web Hosting Service PythonAnywhere.

Hosting platform

Initially the hosting platform was going to be Heroku, but the dependencies of the project exceeded the limitations of Heroku platform. Free users on Heroku have 300 MB capacity to store data. An alternative hosting platform PythonAnywhere was used instead of Heroku, because users on PythonAnywhere can implement Django projects and host them easily. Also PythonAnywhere offers access to commonly known Python modules like NumPy, SciPy and MatPlotLib. Other modules can be installed using pip.

PythonAnywhere offers the use of an IDE, virtual terminal and file system graphical user interface. PythonAnywhere can be used with github. A Free account were created to get access to PythonAnywhere platform. Free account had enough power to provide simple simulation of the project. Only limitation was that cloud users have limited access to computational power on PythonAnywhere platform. This led to some realistic compensations so that the use of the machine learning algorithms could be simulated as a web application. For example in order to have fast computation cycle on the machine learning algorithms only a small set of images could be processed. In order to simulate the real-timeness of the system a buffer for 100 images was implemented. Periodically the buffer images would be updated. If the data set was larger, for example 1000 images, then the running time of the algorithms would be too long and server would force timeout.

The website is simple. It has some introductional text and three text fields and three buttons. Each button will run a different machine learning algorithm and each text field will be used to choose a k value between 1-10. K value was limited between this range, because empty cluster are more likely to occur on higher k values. If an empty cluster happens, the web application will detect it and display an error message. After the algorithm has finished cluster kernels will be shown on the web page. For k-means and k-means++ the closest data point to the kernel mean will be displayed instead of the actual mean.

Data Collection

Data analyzed in the project was provided by Finnish Meteorological Institute (FMI) and retrieved from FMIs Web Map Service. FMI has 10 different weather radars located around Finland which provide the data for WMS. WMS can also merge all of the weather radars data into a single image covering almost all of Finland. Only single radar images were used in the project, since they all shared

the same shape. Every radar also has four different measures, radar reflectivity (dbzh), radial velocity (vrad), rain classification (hclass) and cloud top height (etop_20).

Data was stored locally and on hosting server. Local data was collected between 6.2.2016 and 20.3.2016. Local data set contained 1181 images with the resolution of 720x480 and 624 images with the resolution of 300x250. Hosting server had a different kind of system for data storage. A Buffer was implemented to maintain better running times. The Buffer on the hosting server has a size of 100 images and all of those images have the resolution of 300x250. The Buffer content is updated periodically and the reason for this is that if the buffer would be updated during each HTTP request to the server, the content of the buffer would quickly become uninteresting. The Main purpose of the online implementation is to simulate the usage of the machine learning algorithms thus some compenstations have been made. All of the results in this report have been produced with the local data set to preserve objectivity.

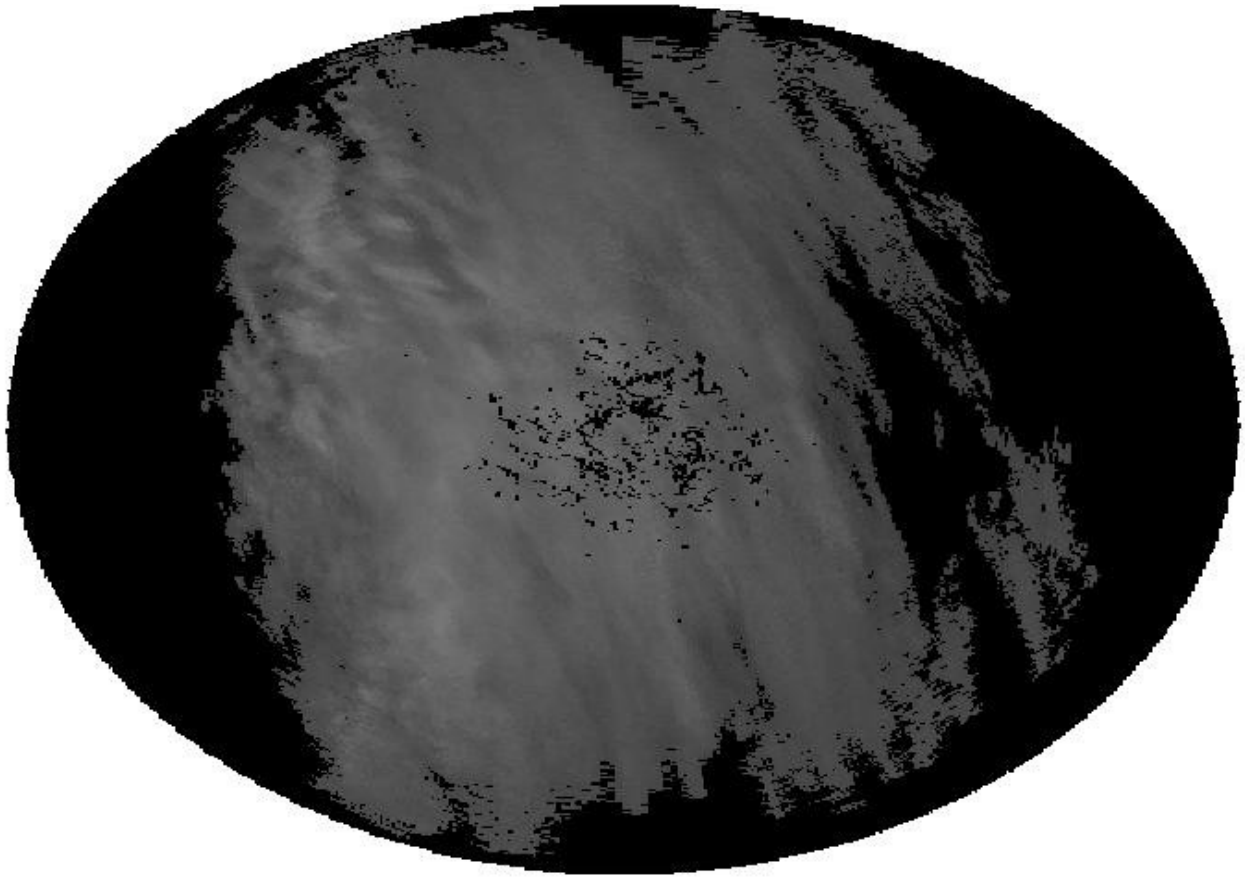

Illustration 1: Sample image with the style raster and resolution 720x480

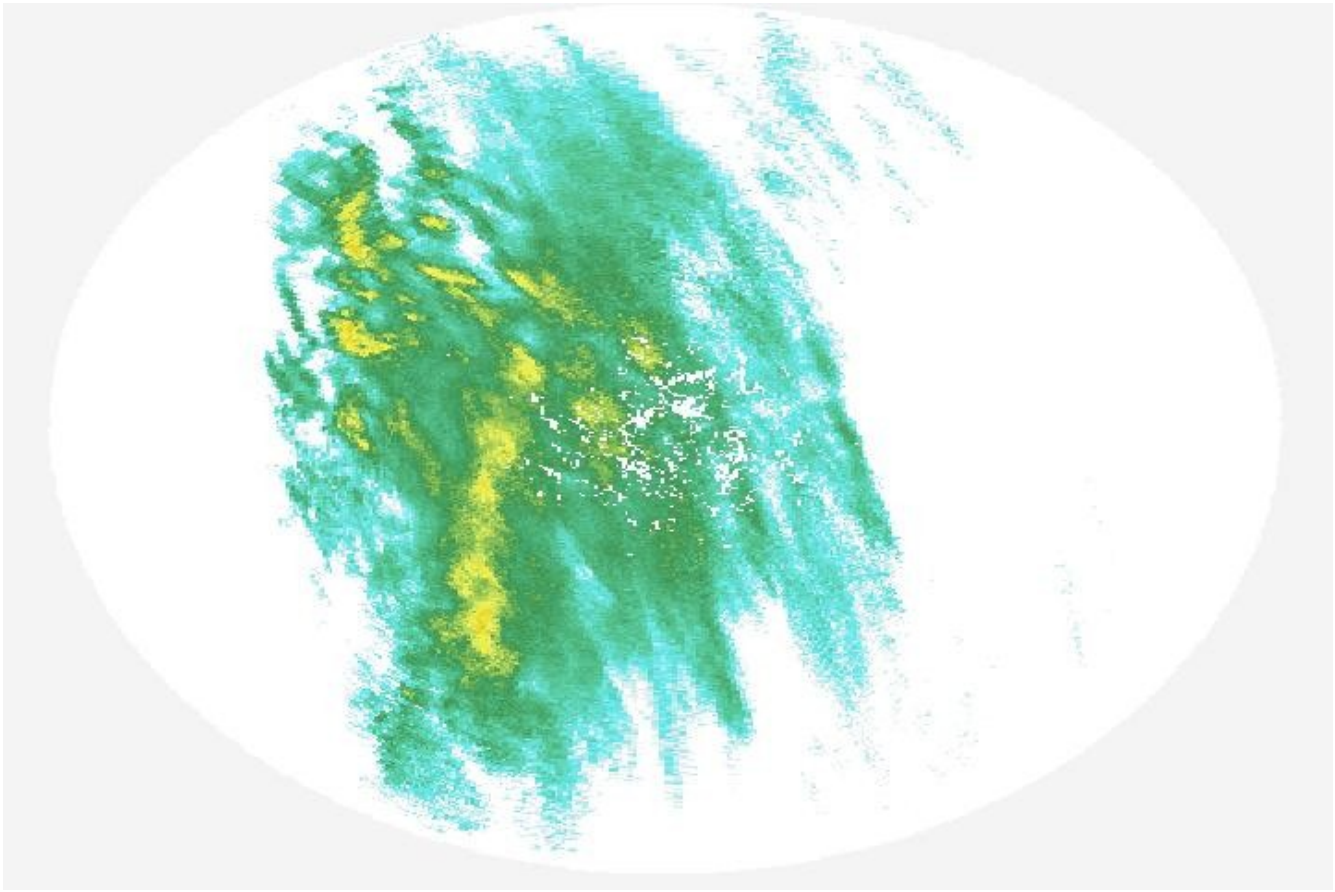

Illustration 2: Sample image with the style Radar dbz Summer and resolution 720x480

Illustration 1 and *2* depict the same radar image with different style. These images represent all the available style options. They both have the same resolution also. Style Radar dbz Summer was used in this project, because the images more detail in terms of colors and the differenses in the cloud chunks are more visible.

Modules

A module *datagathering-py* was implemented to connect to FMI's WMS server. This module is able to retrieve 40 different weather radar images and store them on disk. *datagathering.py* is module that is able to pull all of the different kinds of images from every FMIs weather radar in real time. The initial resolution of the images was 720x480. Image resolution had different options, but 720x480 was selected because it is a standard size. Image resolution affects greatly the clustering algorithms performance. Also larger image resolution was considered, but the algorithm run time on a local machine suggested that smaller resolution would be better for further cluster and algorithm analysis. Because the run time was poor with 480p images, a smaller resolution 300x250 was chosen and new data set was gathered from FMI. New resolution was almost half of the size of the original resolution and thus the algorithm run time improved greatly. Faster algorithm performance with smaller resolution means also loss of detail in images so it is a trade-off between image detail and clustering algorithm run time.

The File *preprocessor.py* is used to read images into numpy arrays. The File defines a class *preprocessor*, which has three local attributes *data, amount* and *path*. *Data* data type is python's built-in list, because with large amount of images *data'*s conversion into numpy array consume a large portion of execution time. *Amount* holds the information about the amount of images read with *preprocessor*. Directory path to the images read by *preprocessor* has been saved into *path*. *preprocessor* has three public functions and one local. The local function is called *read_data()* and it has no parameters, but the conventional *self* parameter. It uses the module *os* to read traverse through the file system. The function also assumes that the files within the root are images, which the module *matplotlib.image* can read. *matplotlib.image.imread* returns *numpy* arrays with the shape (height, width, 3). The shape is because images are read as RGB matrices, where height and widht represent the resolution and their product is the amount of pixels in the image and the third component holds all the values for one pixel's RGB representation. Data type of the matrix is np.uint8. r*ead_data()* appends the class attribute *data* and returns it. Functions *count_data()* and *get_path()* work as typical getter-functions. *get_data()* has one parameter *path*, which is a string that represents the directory path to the data. *get_data()* uses *read_data()* to get the data and stores *path, amount* and the *data* itself. If *get_data()* is called twice with same *path*, then *read_data()* is called only once. *get_data()* returns a four dimensional matrix, which contains all the images as pixel arrays. *get_data_as_2d()* has also one parameter *path* and it uses *get_data()*, but returns data as two dimensional matrix instead of four, where all single images have been reshaped into one dimension with *numpy's reshape* function.

All clustering related code has been encapsulated in *my_models.py* file. The file has a class *my_models* that includes every machine learning algorithm used in the project. K-means [2], k-medoids[2] and k-means++[1] algorithms were selected for cluster analysis and they are implemented as functions inside *my_models*. K-means, k-medoids and k-means++ require initial cluster kernels. *randomly_init_cluster_kernels()* produces initial cluster kernel for k-means and k-medoids and *kpp_kernel_init()* does it for k-means++. *Clusterify()* is the main part of the group step of the algorithms. It returns a dictionary, where every data point's index has placed to its coresponding cluster. *Clusterify()* is also one of the slowest part of the functions, because it uses looping throught the data to get its results. A functional implementations might have yielded better run time or heavier use of *numpy's* functions.

Machine learning algorithms

All of the machine learning algorithms have been implemented into file called *my_models.py.* Module provides means to get access to the data set stored on disk. *my_models.py* contains a class called *my_models* and its attributes describe the properties of the data set.

K-means [2] is very well known predictive clustering algorithm. This project's version of k-means algorithm follows heavily the pseudo-code in Flasch's textbook [2]. K-means algorithm selects the initial cluster kernels uniformly at random. After that it proceeds into grouping the data into clusters based on similarity. The similarity measure used here was euclidean distance. In the final step it will

update its kernels. Kernels will be selected within the cluster and they will be the means of the clusters. Most likely the mean will not be any actual image in the data set.

K-means++ [1] is an improved version of k-means. K-means++ uses the same method to group the data points and update the cluster kernels as k-means, but it has improved way to select initial cluster centers. K-means++ selects the first initial cluster kernel uniformly at random, but rest of the data points have weighted probability of beign the next intial cluster kernel. The weight of the probability is calculated by comparing each data points distance to the existing cluster kernels. So further the data points are from the existing cluster kernels, higher the probability of being the next cluster kernels will be.

K-medoids [2] has same way to initialize cluster kernel and group the data points as k-means, but the update step is different. K-medoids will always update an actual data point as the cluster kernels. Update step will sum every distance to a single cluster point for every cluster point and then the cluster point with the lowest sum will be placed as the new cluster kernel. Update step is also the reason why k-medoids has worse running time than k-means [2].

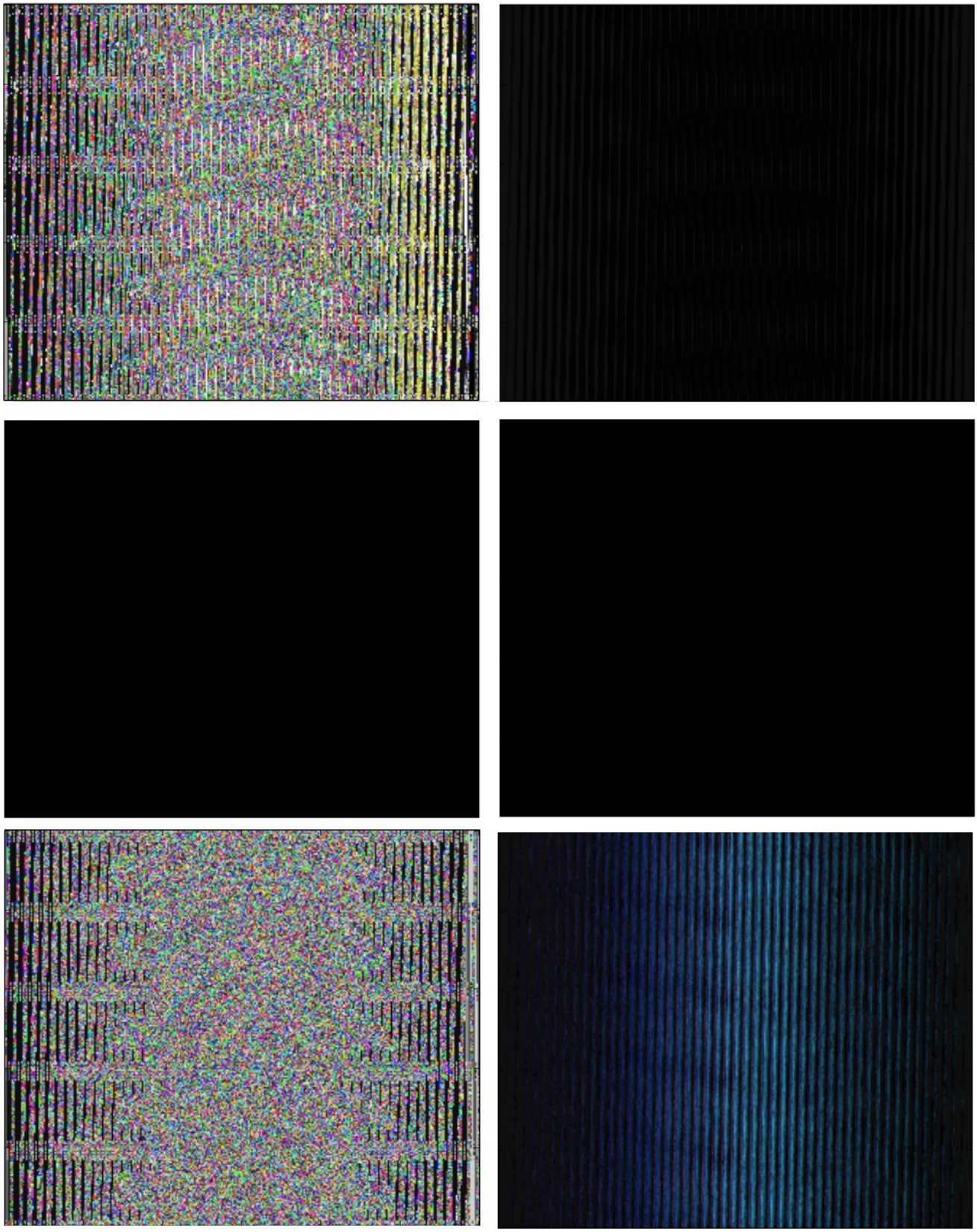

Illustration 3: Samples of cluster means. Left side images have data type np.float64 and right side images have data type np.uint8.

Illustration 3 has sample images of actual cluster means. They were produced by k-means++ algorithm from data set containing 624 images with the resolution of 300x250 where k value was three. One row has the same kernel with different data type. Left column images in *Illustration 3* have data type np.float64 and right column images were rounded and then converted into data type np.uint8, because this was the data type of the original images.

Results

Initial choice for image resolution was 720x480. After gathering total of 1142 images some experimentation with the algorithms were made. Setting k value to 10 loop of k-means took 69 seconds to complete on a local machine. Of course the algorithms requires more loops in order it to converge and higher the k value, more looping would be required. So for instance if k-means were to converge after 50 loops it would take around 3450 second which is approximately one hour of computation time. After this result a smaller image resolution was chosen to improve the running time of the algorithms.

Performance of the algorithms

A small experiment was made to evaluate the performance of the machine learning algorithms. Test data consisted of 624 weather radar images with the resolution 300x250. Every algorithm was tested with k values between the range of 1-6 and one k value was repeated for 10 times to come up with an average value for the dissimilarity within all the clusters and average iterations required for the algorithms to converge.

Illustration 4 shows what kind of effect the increase of k value had on this experiment. Illustration 4 suggests that the iterations required will be monotonically increasing when k value increases. The initialization will also affect the iteration amount. For example it would be possible to get k-medoids to converge in two iterations, if the initial cluster centers would be some centers that converge meaning that the second iteration would not change the centers. For k-means and k-medoids it is little bit tricky, because it is almost certain that means will not be any data points.

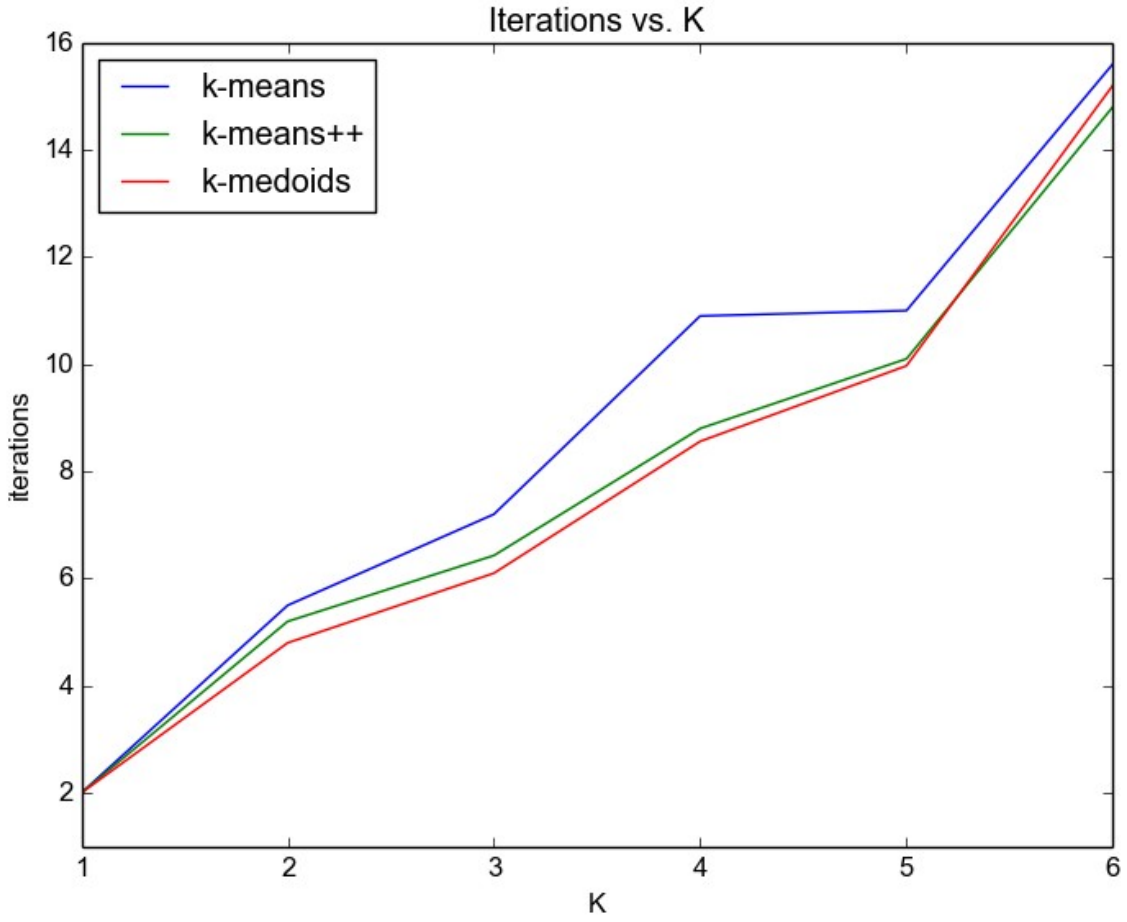

Illustration 4: Illustration has three curves that represent the different machine learning algorithms. X-axis show the different k values and y-axis the amount of iterations required for the algorithm to converge.

Illustration 5 shows how dissimilarity within clusters behaves when k increases. Dissimilarity was calculated by summing every clusters distances between each datapoints and taking the average. Then the dissimilarity values were normalized to the range 0-1. For every curve the lowest dissimilarity value were selected to divide every value and then subtracting one from every value. By doing so the lowest dissimilarity value wil be set to 0 and the rest will be above that. It is also important to note that the last value (k=6) might not have the lowest dissimilarity value.

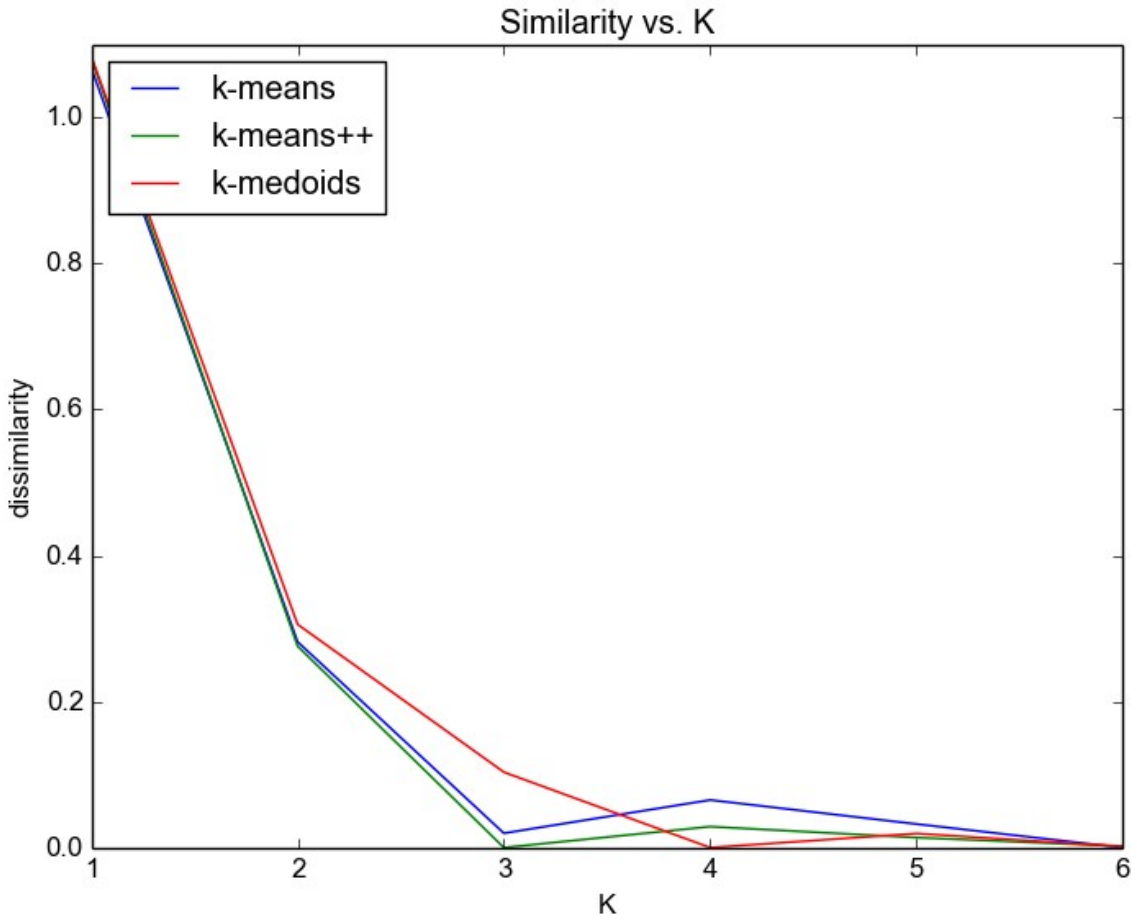

Illustration 5: X-axis represents different k values and y-axis represents dissimilarity.

Discussion

Article [3] also suggests similar kind of results as does this project. Basic predictive clustering methods compare difference between single pixels with every image. This means that for example k-means algorithm does not take account the shapes and position of the clouds within the images. There are more advanced methods to analyze images for example image segmentation by clustering [4] or image mining which is part of data mining [5].

Visualizing the cluster images created by the machine learning algorithms shows that weather radar images can be grouped roughly into two clusters clear weather and not-clear weather. If the k value is larger than two, then the not-clear weather images will be grouped into more fine-grained subgroups, but the shape of the clouds and the position of the clouds in the image will not affect the group forming. The more fine-grained groups consist different portion of the cloud chunks. For example if k equals to three, then the groups will most likely be clear weather, small chunk of clouds and larger chunk of clouds.

So basic predictive machine learning algorithms do exactly what they are ment to do: compare

differences between two data points. If the data points happen to be images, more details are required or better techniques to produce more meaningful clusters.

Conclusion

Clustering algorithms implemented in this project are not powerful enough to find out proper weather types within weather radar images. Only two weather types can be identified clear weather and everything else. If the k size is larger than two, then the everything else class will be divided into more finer-grained groups where only the portion of the clouds matters and shape or location of the clouds on the image does not matter.

Three different clustering algorithms were implemented and their performance was analyzed. K-medoids had the best convergence in terms of number of iterations required for the algorithm to converge. K-means and k-means++ were faster in terms of their running times, but k-means++ had the slight advantage in convergence compared to k-means.

References

[1] Arthur, D. and Vassilvitskii, S. (2007). k-means++ : The Advantages of Careful Seeding. *Proceedings of the Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, pp.1027-1035.

[2] Flach, P. (2012). *Machine learning*. Cambridge: Cambridge University Press.

[3] Ahmed, N. (2015). Recent review on image clustering. *IET Image Processing*, 9(11), pp.1020-1032.

[4] Akhtar, N., Agarwal, N., Burjwal, A. (2014). K-Mean Algorithm for Image Segmentation Using Neutrosophy. *2014 International Conference on Advances in Computing, Communications and Informatics (ICACCI)*, pp.2417-2421.

[5] Haiwei, P., Li, J., Wei, Z. (2005). Medical Image Clustering for Intelligent Decision Support. *Proceedings of the 2005 IEEE Engineering in Medicine and Biology 27th Annual Conference*, pp.3308-3311.

Data provider:

ilmatieteenlaitos.fi

Project repository:

https://github.com/mihassin/ubiquitous-spoon

Project is hosted at:

http://mihassin.pythonanywhere.com/

Project schedule:

https://docs.google.com/spreadsheets/d/1SMEjF9h94tKEg0vpC7j1yXE0jVpetmsGcNhnBnqYh8M/edit#gid=0

Hour accounting chart:

https://docs.google.com/spreadsheets/d/1SMEjF9h94tKEg0vpC7j1yXE0jVpetmsGcNhnBnqYh8M/edit#gid=2095313416.