University of Oslo

FYS-STK4155

Helle Bakke

# k-means clustering of Ca II 8542 as a solar flare diagnostics tool

December 18, 2019

# Contents

**Abstract**

Observational data from the Swedish 1-m Solar telescope (SST) containing two B-class flares and a microflare in the Ca II 8542 was studied by developing a k-means clustering method. The method was parallelised using OpenMPI, as the large data set was computationally heavy ro run in serial. The developed code was compared to the k-means clustering code from Scikit-Learn in terms of computation time, where the performance was measured for numbers of processors between 1-6. A training sample including relevant frames from the pre-flare, flare and post-flare phases was chosen, and the optimal number of clusters was found by analysing the mean standard deviation through the elbow method. The final centroids, found by training the model for 10 runs using the optimal number of clusters $k = 50$, were applied to the entire time series to cluster the solar flares. It was found that solar flare signatures were sampled in the clusters of high peak-intensity centroids, proving that the trained model works as a solar flare diagnostics tool in the Ca II 8542 Å channel.

# 1 Introduction

The application of machine learning in science has become more and more common in the past years. Numerous papers in scientific fields explore both supervised and unsupervised learning methods. Supervised learning algorithms learn from labelled data, meaning that you have input variables and an output variable where the mapping function from the input to the output is learned. The network is trained by back-propagating an error calculated using the predicted output and label. Unsupervised learning algorithms aim to model the underlying structure of the data from input data only, meaning that there is no labelled output. One of the main unsupervised learning methods is called *clustering*. Clustering algorithms find generative features in the data set and group them together. The different groups are then labelled by eye later on. One of the most popular clustering methods is called *k-means* clustering, which partitions the data into $k$ clusters. In this project, we apply the k-means clustering algorithm to observational data from the Swedish 1-m Solar Telescope (SST, Scharmer et al. (2003)). The aim is to cluster the pixels of each frame in the time series based on their Ca II 8542 Å line profile in order to detect flaring events.

The structure of this report is as follows. We start by introducing the observational data, k-means algorithm and developed code in Section 2. In Section 3 we present our findings, which are then discussed in Section 4. Section 5 gives a conclusion and suggestions for future work.

# 2 Method

## 2.1 Observational data

The data set used in this project is a time series of an enhanced region of the Sun. The observation is taken with the Swedish 1-m Solar Telescope (SST), and shows full spectro-polarimetry in the Ca II 8542 Å line over a duration of 165 minutes. The complete data set consists of 1198 pixels in $x$-direction, 1162 pixels in $y$-direction, 311 scans (timesteps) and 12 wavelength points. Figure 1 represents a single timestep in the series. Each timestep consists of 12 frames representing the wavelength points, and a single pixel from all of the 12 frames make up the Ca II 8542 Å spectral line.

The proposed data set is particularly interesting due to the occurence of solar flares. Solar flares are brightenings in the solar atmosphere caused by a release of energy stored in the magnetic fields. The energy released varies from $\sim 10^{27}$ erg in the smallest events to $\sim 10^{32}$ erg in the largest events. The size of the flare depends on its soft X-ray (SXR) flux. Most solar flares are classified by the letters A, B, C, M and X, with X-class flares being the most energetic. They have a peak flux at around $10^{-4}$ W/m$^2$, where the peak flux of each class is ten times greater than the preceeding one (Priest, 2014). Smaller flares are not classified by a letter, and are rather referred to as either microflares or nanoflares depending on how energetic the event is. Over the duration of the observational data, two B-class flares and a microflare occur. The main goal of this project is to cluster spectral lines of different features in order to label the pixels where solar flares are present. The spectral lines of such events will have brightenings in the line core, meaning that a central line reversal is visible. We are interested in seeing whether or not the microflare will be clustered as a flare or not, as the released energy of the event is lower than the B-class flares and might not be visible in the Ca II 8542 Å line.

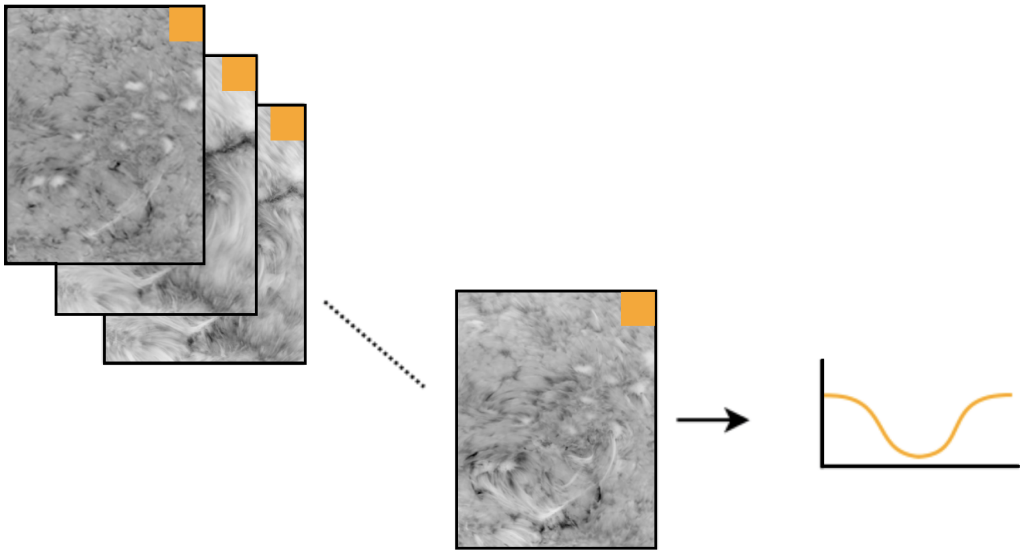


Figure 1: Simple illustration of how each pixel in a single scan (timestep) of the data set makes up a Ca II 8542 Å spectral line.

## 2.2 k-means clustering algorithm

In machine learning, clustering is the process of finding subgroups within the data such that all points in each cluster is as similar as possible (Dabbura, 2017). The similarity is a distance-measure, and varies based on the clustering method. In k-means clustering, the data is partitioned into $k$ pre-defined clusters. For each iteration, data points are assigned to a cluster such that the euclidean distance between the points and the cluster centroid is minimal. The algorithm also aims to keep the clusters as different as possible, leaving $k$ distinct clusters at convergence. In short, the k-means algorithm works as follows:

- Initialise $k$ centroids.
- Compute the euclidean distance between data points and centroids.
- Assign each data point to the closest centroid.
- Compute new centroids by taking the mean of all data points belonging to each cluster.

- Iterate until the centroids do not change and convergence is reached.

The above steps are general, and in order to create a k-means algorithm optimised for the observational data, we need to modify the algorithm to some extent. In the following, we take a deeper dive into the algorithm and modifications made for the specific problem at hand.

The first step in the k-means algorithm is to initialise the centroids. One option is to select the centroids randomly, but a drawback is that the centroids can end up close to each other and more iterations are needed in order to reach convergence. Another possibility is to implement the `k-means++` initialisation method (Arthur and Vassilvitskii, 2007). `k-means++` aims to spread out the initial centroids so that they are not too close. The first centroid $c_1$ is chosen uniformly at random from the data set $\mathcal{X}$, and the euclidean distance between $\mathcal{X}$ and $c_1$ is calculated as

$$D(x) = \sum_{i=1}^{n} ||x_i - c_1|| \tag{1}$$

where $x \in \mathcal{X}$. Each data point $x$ is then chosen with the probability

$$P(x) = \frac{D(x)^2}{\sum_{x \in \mathcal{X}} D(x)^2} \tag{2}$$

and the new centroid $c_2$ is generated non-uniformly at random from $\mathcal{X}$. By choosing each $x$ with the probability given in Equation (2), the new centroid is as far away as possible. New centroids $c_i$ are computed until $k$ centroids are obtained. This method produces initial centroids with relatively low heterogeneity, but sometimes the method can get stuck at local minima. The k-means clustering provided by Scikit-Learn runs the algorithm `n_iter` times with different initialisation, and returns the configuration which results in the lowest heterogeneity. For small data sets this is relatively fast, but with larger data sets this is quite time-consuming. The developed algorithm in this project makes use of the `k-means++` initialisation method, but leaves out the reruns for optimal clustering heterogeneity due to time constraints. However, we keep in mind that this is a potential drawback of our algorithm.

After the initial centroids have been selected, the euclidean distance between the data points and centroids are calculated as in Equation (1). Each data point is assigned to the closest centroid, before new centroids are computed by taking the mean of the data points belonging to each cluster. In

order to determine when the centroids do not change significantly, we calculate the relative error of the within-cluster sum of squares (WCSS). The WCSS is calculated as

$$\text{WCSS} = \sum_{k=1}^{K} \sum_{j \in S_k} \sum_{i=1}^{n} (x_{ij} - c_{ik})^2 \tag{3}$$

where $K$ is the total number of cluster and $S_k$ is the set of observations in the $k^{\text{th}}$ cluster. The inner sum is identified as the squared euclidean distance. The relative error is calculated as

$$\epsilon_{\text{rel}} = \left| 1 - \frac{\text{WCSS}_l}{\text{WCSS}_{l-1}} \right| \tag{4}$$

where the subscript $l$ denotes the iteration. Convergence is reached when the relative error is below a certain tolerance set in the initialisation of the algorithm.

## 2.3 Code

The developed k-means clustering algorithm is implemented in both serial and parallel. The reason for the parallelisation is that the observational data is large, and a serial implementation takes too long to execute. The k-means clustering from Scikit-Learn has a parallel option, hence we want to introduce the same flexibility to our code. The implementation of the algorithm follows the bulleted list in the previous section. The `k-means++` initialisation method can be seen in the following pseudocode:

```
import numpy as np

centroids = np.zeros((k, data.shape[0]))

# define random index for first centroid
random_index = np.random.choice(data.shape[0])

centroids[0] = data[random_index]

# calculate distance
distance = np.linalg.norm(data - centroids[0], axis=1)

i = 1
while i < k:
```

```
  prob = distance**2
  random_index = np.random.choice(data.shape[0], size=1,
                                  p=prob/np.sum(prob))
  centroids[i] = data[random_index]

  new_distance = np.linalg.norm(data - centroids[i], axis=1)
  distance = np.min(np.vstack((distance, new_distance)),
                    axis=0)
  i += 1
```

This part of the code works in both serial and parallel, as the initial centroids are being broadcast from the root process to all other processes using OpenMPI. OpenMPI is an open source Message Passing Interface allowing programs from a variety of coding languages to exploit multiple processors. We use the `python` package `mpi4py` to parallelise the k-means clustering code. Since each spectral line in the data set is independent on position, the clustering of data can be parallelised without too much complication. After the initial centroids are distributed and the data is split into semi-equal[1] sub-matrices on each process, local clusters are calculated and gathered to the root process. The gathering of local clusters to a global cluster array is seen in the following pseudocode:

```
for i in range(k):
  distances[:, i] = np.linalg.norm(mpi_data - centroids[i],
                                   axis=1)

local_clusters[:] = np.argmin(distances, axis=1)

comm.Gatherv(local_clusters, [global_clusters, pN], root=0)
```

`mpi_data` are the local sub-matrices distributed in the beginning of the code and `pN` is the number of points per processor. On the root process, the new centroids are calculated before being broadcast to all processors, so that the sum of local distances needed for the within-cluster sum of squares can be calculated in parallel. The sum of local distances are then reduced with a sum to the root process, where the WCSS and relative error are calculated. The relative error is broadcast to all processors, and the clustering converges when the error is lower than the tolerance.

[1] If the total number of data points are non-divisible by the number of processors, the code distributes the sub-matrices so that the last processor obtains the one including the remainder.

Our aim has been to parallelise as many parts of the code as possible. However, with our lack of experience with OpenMPI, additional parts of the code might be parallelisable as well. In the results section, we investigate the performance of the developed code compared to Scikit-Learn.

# 3 Results

## 3.1 Model verification

The model was verified using a simple data set as illustrated in Figure 2 (a and d). The top row shows clustering of the data when the initial centroids are chosen at random, while the bottom row shows clustering of the data using the `k-means++` initialisation method. Clustering after the first iteration can be seen in the middle panels, while the panels to the right give the final configuration from both runs. The figure shows that the model is working using both random and `k-means++` initialisation. The final centroids are well distributed, meaning that they are neither too close or too far away from each other.

From the final configurations, we observe how much the initialisation of the centroids matter for the outcome. The cluster that is purple in panel (c) is two different clusters in panel (f). This can have a significant impact if important samples, such as our flares, end up in two different cluster. This implies that we need to train our model sufficiently, meaning that we need to run it several times in order to decide on the best final configuration.

Animations of the clustering using both random and `k-means++` initialisation can be found at `http://folk.uio.no/hellmb/FYS-STK4155/`.

## 3.2 Performance

Since a version of our code is parallelised using OpenMPI, we compare the performance of our model to the k-means clustering model by Scikit-Learn. In order to obtain comparable times, we set equal random seeds in both implementations, run the models with 10 clusters and use the `k-means++` initialisation method. We choose a single scan from the observation data, but because the data set is large, we reduce it to approximately 10% of its size. The models are run 10 times for each number of processors ranging from 1-6, and the result is seen in Figure 3. We note, however, that we had some issues

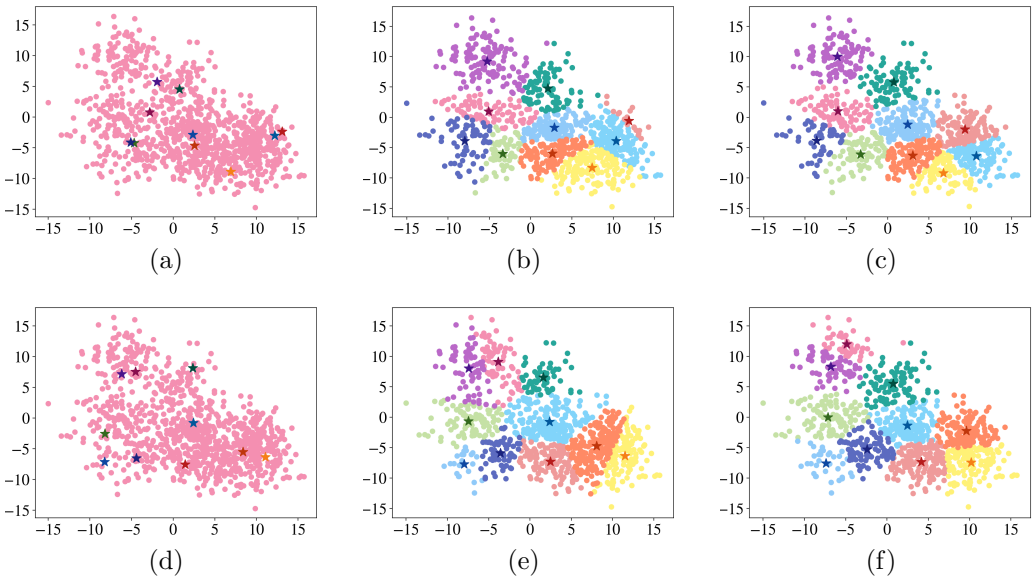


Figure 2: Clustering of a simple data set with $k = 10$. *Top:* Initial centroids chosen at random (a). No data points have been assigned to a cluster yet. Centroids and clusters after the first iteration (b). Final configuration (c). *Bottom:* Initial centroids from the `k-means++` method (d). No data points have been assigned to a cluster yet. Centroids and clusters after the first iteration (e). Final configuration (f).

with speeding up the Scikit-Learn model. The `n_init` argument specifies the number of times the algorithm is run with different centroid seeds, where the final result is the centroid providing the lowest WCSS score. The `n_jobs` argument specifies the number of parallel jobs to use for computation. Based on this information, it is intuitive to set `n_init = 1` and vary `n_jobs` in the range of 1-6. However, when we did this, we got a severe increase in time as the number of parallel jobs increased. Due to time restrictions, we have not been able to investigate this problem, but a possible reason might be that Scikit-Learn is not optimised for the large data. Because of this, the Scikit-Learn performance curve is from default runs where `n_init = 10` and `n_jobs` varies between 1-6.

With the drawback described above, we can not compare our method to Scikit-Learn directly in terms of model performance. However, we see from the figure that an increase in the number of processors decreases the mean

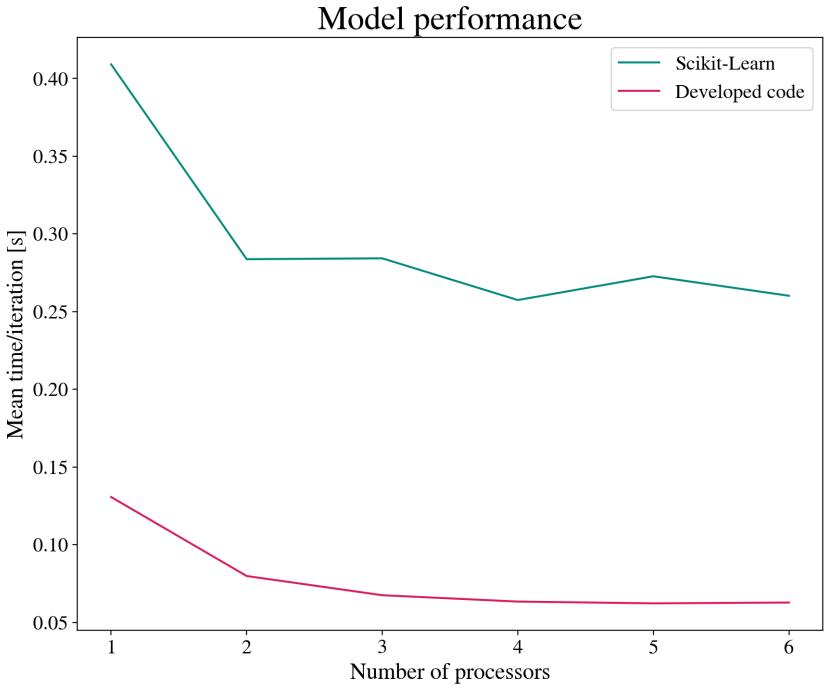



Figure 3: Time per number of iterations for the developed model and Scikit-Learn.

computation time of the developed code. For the reduced observational data set, there is no significant speed-up when the number of processors is larger than 4. Instead, we investigate the performance of the developed code for different number of clusters. Figure 4 shows the mean time per iteration for the different number of processor when the model was run for 10, 12, 14, 16, 18 and 20 clusters. We see that when $k$ increases, the computation time increases as well for all number of processors.

## 3.3 Training the model

The model was trained on 5 scans including a pre-flare phase, a flare event and a post-flare phase. The observational data has a cubic shape `data[xpix, ypix, scans, wav]`, where each parameter is explained in Section 2.1. We preprocess the data by reshaping it into a 2D array of dimension (xpix $\cdot$ ypix $\cdot$ scans $\times$ wav), and send it as input to the model function. The first task is to find the optimal number of clusters. There exists a variety of methods

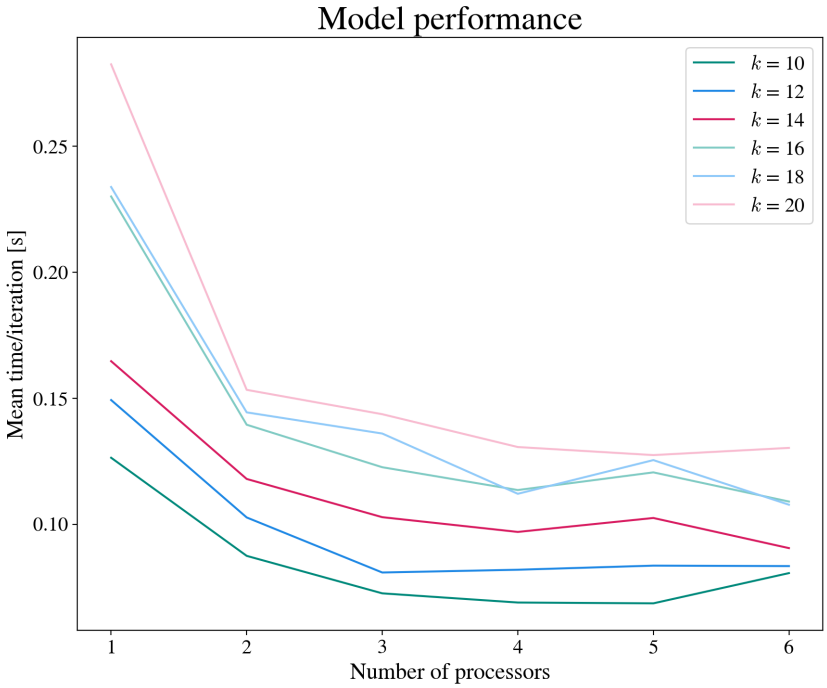



Figure 4: Performance of developed code for different cluster values.

for choosing this number, such as the average silhouette method and elbow method. For data sets where a small amount of clusters are sufficient, the average silhouette method is a visually good aid (see Appendix B for an example using a simple data set). However, for large data sets that might require many clusters, the elbow method is easier to illustrate. The model is run for values of $k \in [20, 100]$, and for each cluster the mean standard deviation is calculated (Bose et al., 2019). It is inferred from Figure 4 that the computation time increase with $k$, hence we know that the 80 runs will take a lot of time. Figure 5 shows the elbow method for the model runs. The optimal number of clusters are found "in the elbow", and we have decided to set this number to 50 based on the figure.

The model was then trained on 50 clusters by running it 10 times. For each run, the final centroids were stored such that we could calculate the within-cluster sum of squares (WCSS). The run with the lowest WCSS score has the lowest heterogeneity, and is the model we will use for clustering of solar flares in the entire time series.

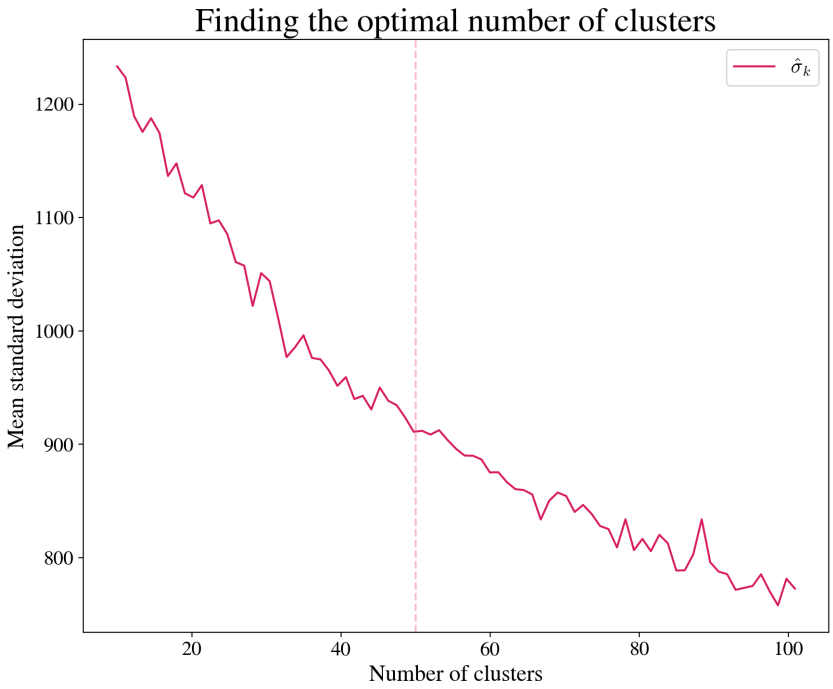



Figure 5: Elbow method used as visual aid to decide the optimal number of clusters.

Figure 6 shows a selection of cluster centroids for the developed model (left) and Scikit-Learn (right). The k-means clustering model by Scikit-Learn has been trained using $k = 50$, such as the developed model. This is because we did not have time to decide the optimal number of clusters in the same way as the developed code, hence we use the same result. We see from the figure that selected cluster centroids are similar. We are mainly interested in centroids showing emission, meaning that the centre of the line has a higher intensity than the line wings. We have selected the cluster centroids showing the highest peak intensities, as well as a few centroids with low peak intensities., From the figure, we clearly see that both models include cluster centroids of equally high peak intensitites. This is an additional verification of the developed code.

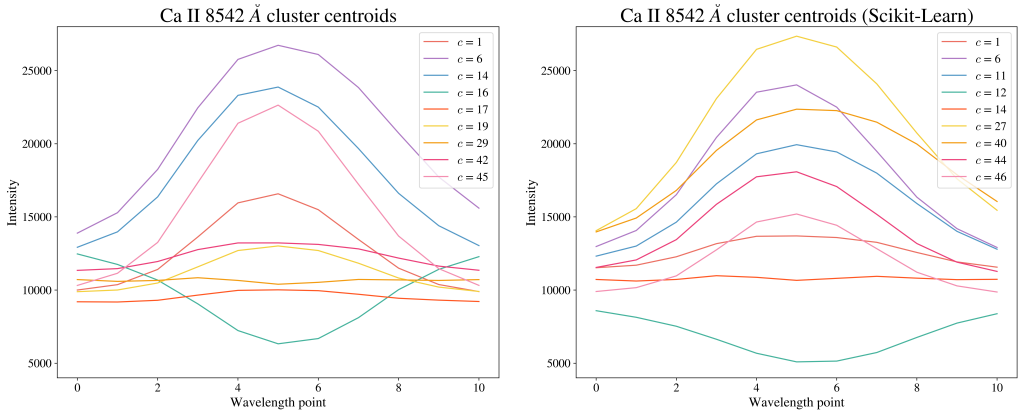


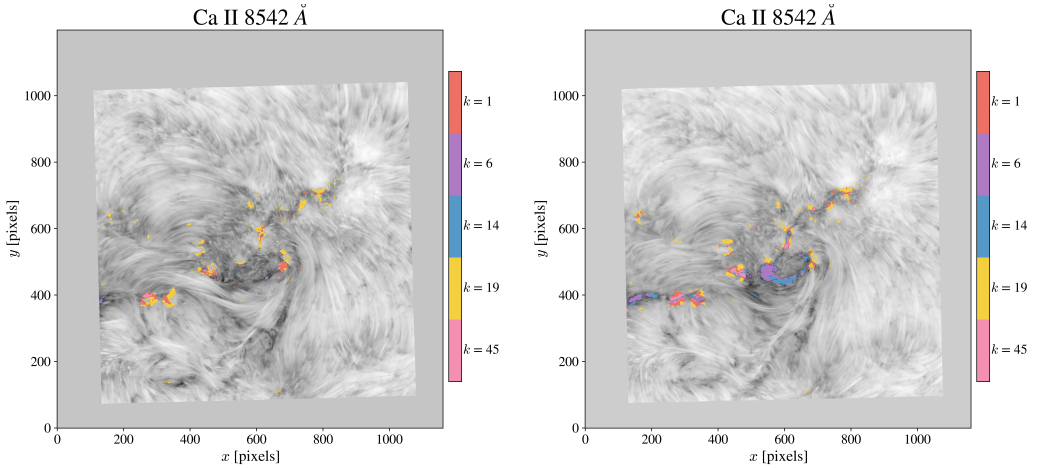

Figure 6: Selection of cluster centroids. *Left:* Developed code. *Right:* Scikit-Learn.

## 3.4 Solar flare clustering

The entire time series was clustered using the best centroids obtained from training the k-means model. We chose the cluster centroids that provided the highest peak intensity, as seen in Figure 6 (left). The observational data included two B-class flares and a microflare, where the aim was to cluster all the events.

Figure 7 shows the first B-class flare event. The clusters of centroids showing the highest peak-intensities are highlighted, and the colorbar gives the respective cluster. Panel (a) is a frame from the pre-flare phase, panels (b) and (c) are frames from the flaring phase and panel (d) is a frame from the post-flare phase. The purple, blue and pink clusters have centroids of high peak intensity, meaning that flaring events should gather in those clusters. In the pre-flare phase, pixels located in these clusters are starting to appear in the active region. When the flare occurs, additional pixels are clustering where the intensity is high. In the post-flare phase, the high-intensity pixels centred in the middle start to disappear.

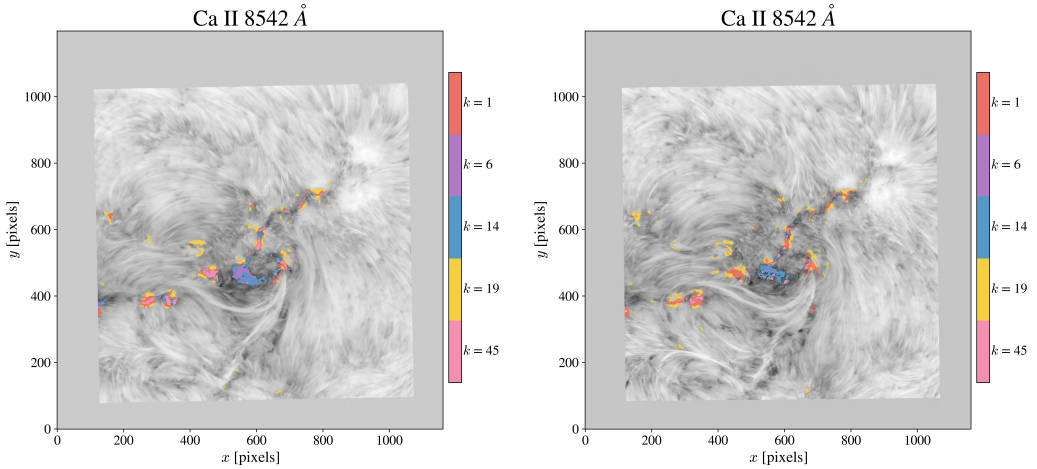
The figure also shows intensities in some of the active region pixels. On the Sun, an active region is an area where the magnetic field is strong. As explained in Section 2.1, flares are brightenings caused by a release of energy stored in the magnetic fields. When the field lines *reconnect*, particles are accelerated as the magnetic energy is converted into heat and kinetic energy. The energy released during the flare is carried to the lower atmosphere by




(a) Pre-flare phase




(b) Flare phase




(c) Flare phase




(d) Post-flare phase

Figure 7: Observational data frames from the first B-class flare.

high-energy particles or thermal conduction, where it is deposited. During the energy deposition, the ambient surroundings are heated. When this happens, brightenings can be seen in certain spectral lines depending on what height of the solar atmosphere the energy is deposited. The red ($k = 1$) and yellow ($k = 19$) clusters have lower peak-intensity centroids, and the reason they appear at a relatively constant rate throughout the time series can be because smaller amounts of energy are constantly being released. The energy

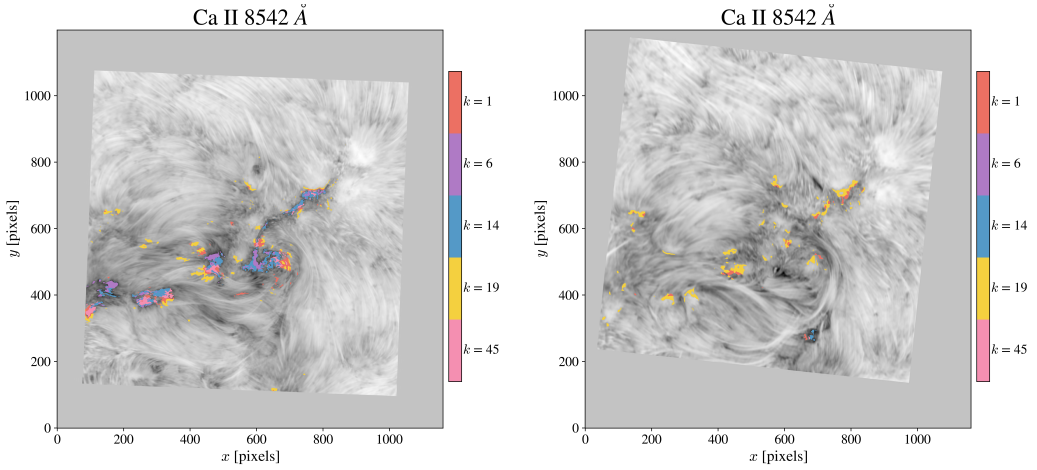



Figure 8: Observational data frames from two more events. *Left*: Second B-class flare event. *Right:* Microflare event.

is deposited along the flare ribbon, as seen from the red and yellow clusters in all panels (with the exception of a few brightenings outside the scope of the flare ribbon). This effect is seen in the entire time series, which can be found at http://folk.uio.no/hellmb/FYS-STK4155/full_run_wav3.mp4.

Figure 8 shows the the second B-class flare (left) and the microflare (right). The second B-class flare affects a larger part of the field of view, as it is seen stretching over a wider area as compared to the first flare. The microflare is a bit difficult to spot, but can be seen around $x = 700$, $y = 280$. It is not related to the flare ribbon, and occurs independently of the B-class flare. The pixels of the event are gathered in two clusters of high peak-intensity centroids ($k = 14$ and $k = 45$).

# 4 Discussion

We have developed a k-means clustering code that runs in both serial and parallel. Parallelisation of the code results in a speed-up in terms of computation time, which is needed in order to train the method on the large observational data set. Figure 4 infer that the computation time increases with the number of clusters, which is valuable knowledge when deciding how many clusters are realistic to include in the elbow analysis. From the anal-

ysis, we find that the optimal number of clusters is $k = 50$. After training the model, we apply the best centroids based on the within-cluster sum of squares to the entire time series. Figures 7 and 8 show that the number of clusters selected is sufficient in order to cluster both B-class flare events as well as the microflare event. The trained model can further be used on other observational data in Ca II 8542, where flares of relatively equal intensities are present. The model has applications outside the scope of this project, as it works as a solar flare diagnostic tool in the Ca II 8542 Å channel.

By comparing the training of the developed model to Scikit-Learn, we find that the peak-intensities of the cluster centroids are relatively equal. We keep in mind, however, that no training using the k-means clustering of Scikit-Learn was performed due to time restrictions. Therefore, we used the optimal number of clusters found with the developed code in the Scikit-Learn method. Preferrably, we would like to train both models independently in order to make a more direct comparison. With the model performance presented in Figure 3, we suspect that Scikit-Learn uses less computation time overall compared to the developed method. For the developed method, we will need to optimise every line of code in order to obtain optimal speed-up in terms of computation time. Regardless, we have developed a method that works well on the observational data in terms of clustering.

# 5 Conclusion

We have developed a k-means clustering algorithm for detecting solar flare events in observational data from the Swedish 1-m Solar Telescope (SST). After verifying the model on a simple data set, we selected a training sample from the observational data and ran the clustering method 80 times for different values of $k \in [20, 100]$. The optimal number of clusters was selected using the elbow method, and was found to be $k = 50$. We trained the model by running it 10 times with the optimal number of clusters, and found the best centroids by computing the WCSS. The results showed that the developed method was able to cluster both the B-class flare events as well as the microflare event for the entire time series. The results obtained from the developed method further proved that k-means clustering of Ca II 8542 works as a solar flare diagnostics tool.

Future improvements include running the algorithm several times for the same number of clusters, and calculating the the WCSS score to get the

best centroid. There are several ways of automating the method so that is it more similar to Scikit-Learn. However, with the limited time we had to develop the code, we set our focus to make the algorithm as fast as possible by parallelising it with OpenMPI. There still exists parts of the code that can be optimised in terms of parallelisation, in order to speed up the calculations even more.

# A GitHub

Relevant programs developed for this project can be found at the GitHub address

https://github.com/hellmb/FYS-STK4155/tree/master/Project_3

# B Average silhouette method

We apply the average silhouette method to the simple data used for model verification in Section 3.1. Figure 9 shows the silhouette analysis for $k = 5$ (left) and $k = 10$ (right) clusters. The height of each silhouette represents the fraction of the data sampled in the respective cluster. When $k = 5$, the data is more evenly distributed in the cluster in comparison to clustering with $k = 10$. A large silhouette coefficient indicates if the sample is far away from neighbouring clusters, while a silhouette coefficient close to 0 indicates if the sample is on or close to the decision boundary between neighbouring clusters. Negative coefficients implies that the sample is wrongly assigned, and rather belongs to a different cluster.

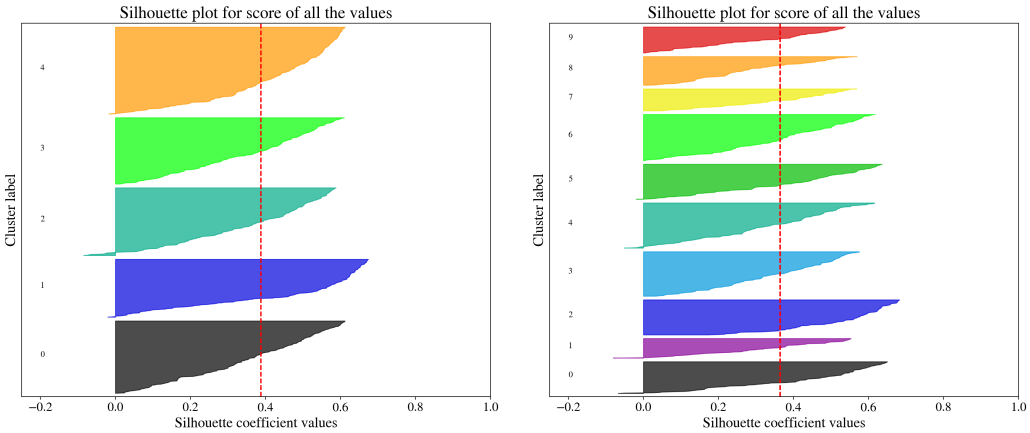



Figure 9: Average silhouette analysis on the simple data set for $k = 5$ and $k = 10$ clusters.

# References


Arthur, D. and S. Vassilvitskii
2007. k-means++: The advantages of careful seeding. In *Proceedings of the eighteenth annual ACM-SIAM symposium on Discrete algorithms*, Pp. 1027–1035. Society for Industrial and Applied Mathematics.

Bose, S., V. M. J. Henriques, J. Joshi, and L. Rouppe van der Voort
2019. Characterization and formation of on-disk spicules in the Ca II K and Mg II k spectral lines. *Astronomy and Astrophysics*, 631:L5.

Dabbura, I.
2017. K-means clustering: Algorithm, applications, evaluation methods, and drawbacks. https://towardsdatascience.com/k-means-clustering-algorithm-applications-evaluation-methods-and-drawbacks-aa03e644b48a. Accessed: 2019-15-12.

Priest, E.
2014. *Magnetohydrodynamics of the Sun*.

Scharmer, G. B., K. Bjelksjo, T. K. Korhonen, B. Lindberg, and B. Petterson
2003. *The 1-meter Swedish solar telescope*, volume 4853 of *Society of Photo-Optical Instrumentation Engineers (SPIE) Conference Series*, Pp. 341–350.