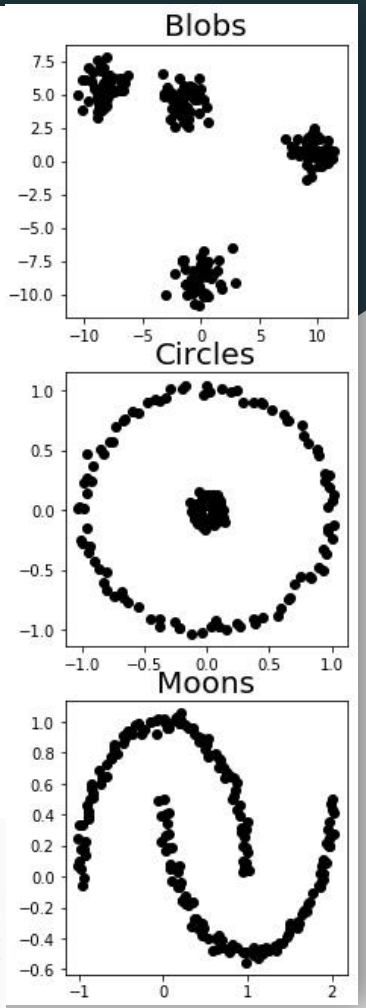
Kernel k-means

Data sets, kernels used and implementations

- The data sets used are the ones seen to the right, with 200 data points each. They were generated using “blobs”, “circles” and “moons” in python’s sklearn.
- The kernels used are the examples used in lecture 12: (image from lecture 12).
- The assignment matrix Z initially assigned the data points to random clusters, with the top of Z being a unit matrix to ensure that all clusters had at least one data point, and then placing one 1 on each row on a random column.
- Everything was done in python.
- Seed was used to get the same data sets each time.
- In laplacian the l1-norm is used.



Linear kernel $k(\mathbf{x}, \mathbf{y}) = \mathbf{x}^\top \mathbf{y}$

Polynomial kernel $k(\mathbf{x}, \mathbf{y}) = (\gamma \mathbf{x}^\top \mathbf{y} + r)^m$

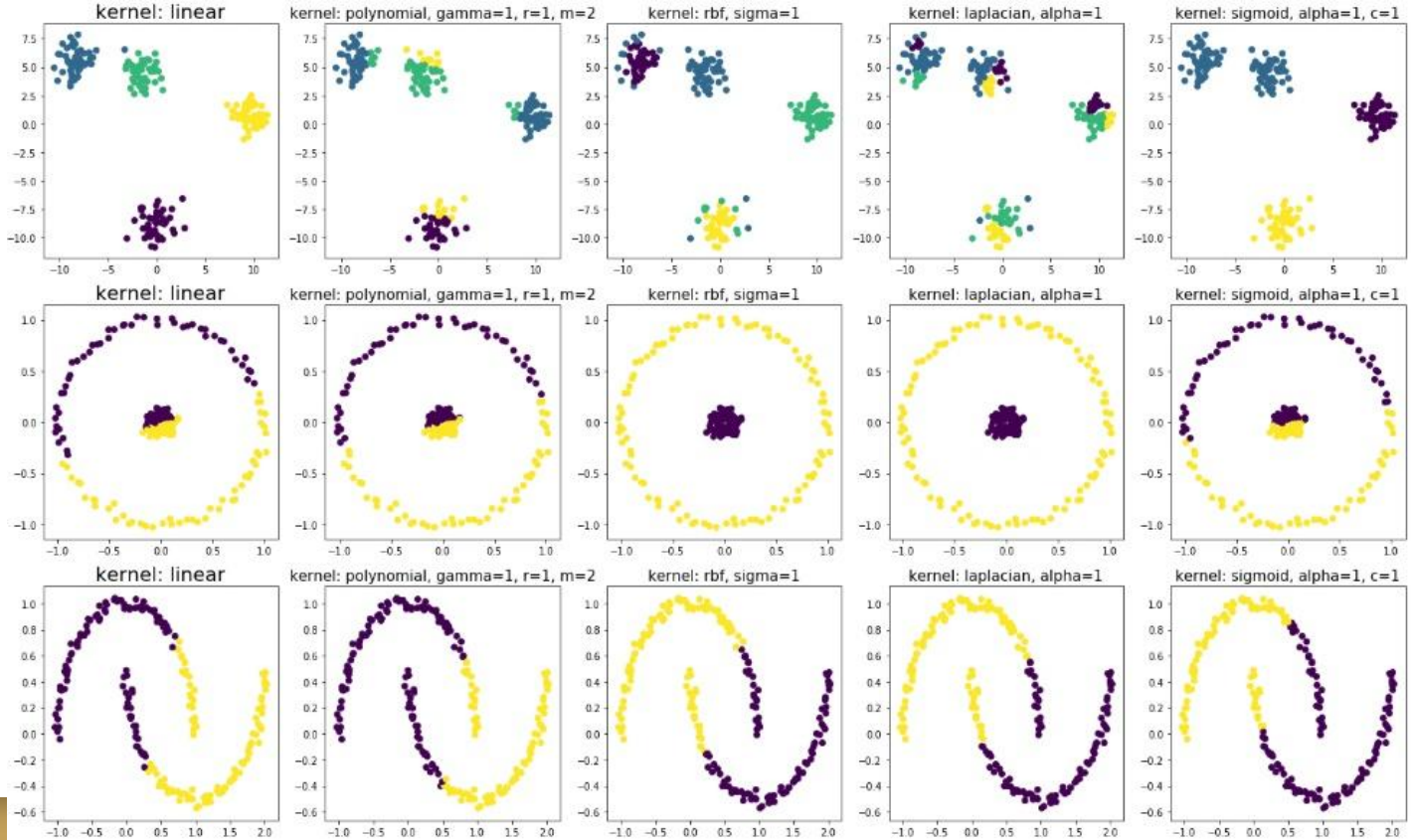
Radial basis function (RBF) kernel $k(\mathbf{x}, \mathbf{y}) = \exp\left(-\frac{\|\mathbf{x} - \mathbf{y}\|_2^2}{2\sigma^2}\right)$

Laplacian kernel $k(\mathbf{x}, \mathbf{y}) = \exp(-\alpha \|\mathbf{x} - \mathbf{y}\|_2^2)$

Sigmoid kernel $k(\mathbf{x}, \mathbf{y}) = \tanh(\alpha \mathbf{x}^\top \mathbf{y} + c)$

Results for different kernels with arbitrarily chosen parameters

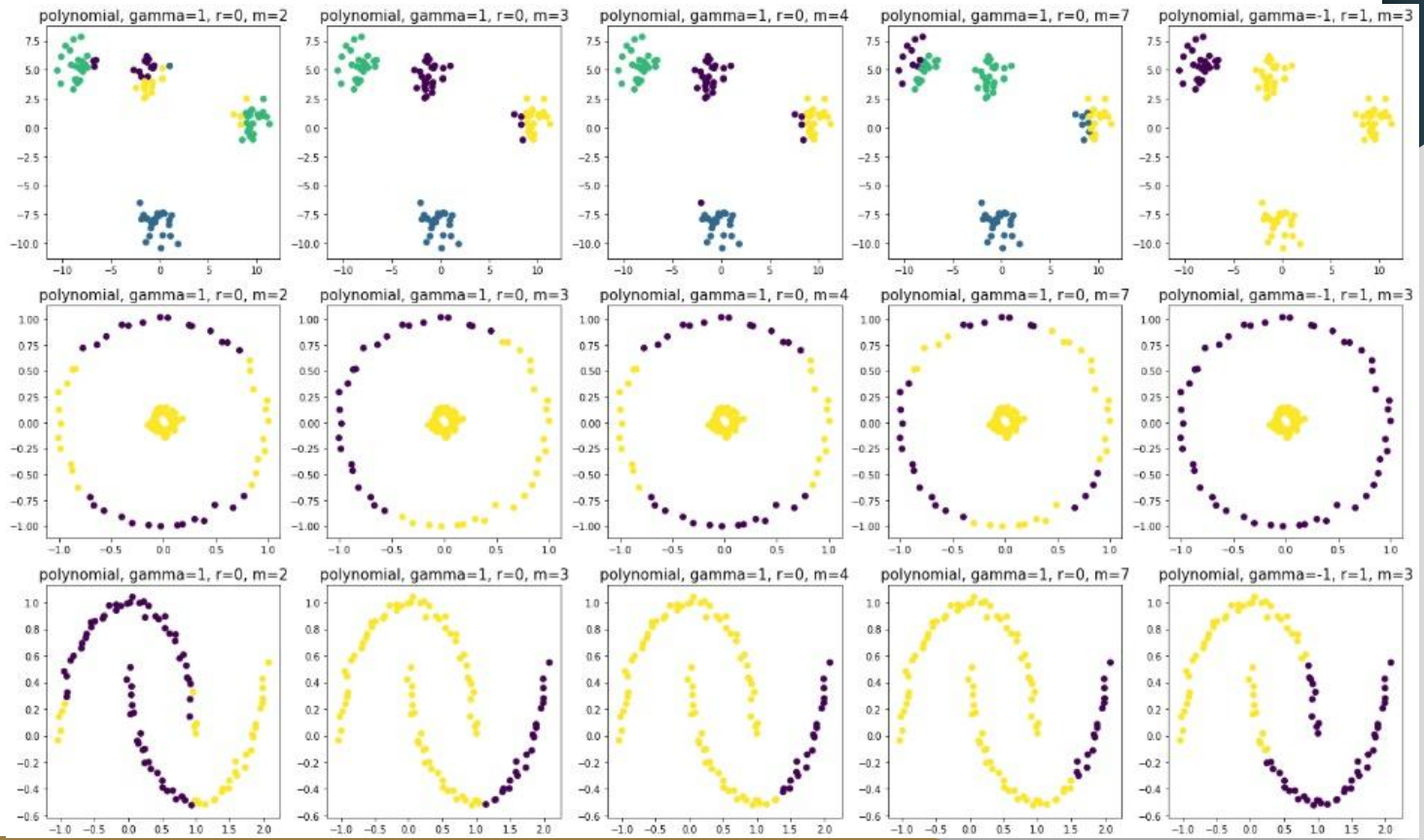
The kernels with a norm (rbf l2-norm, laplacian l1-norm) could cluster the circles, the others split it in two halves. The sigmoid kernel generated homogenous clusters but only found three clusters for the blobs, while the laplacian, polynomial and rbf-kernel had mixes of classes in their clusters.



Polynomial kernel clustering exploration

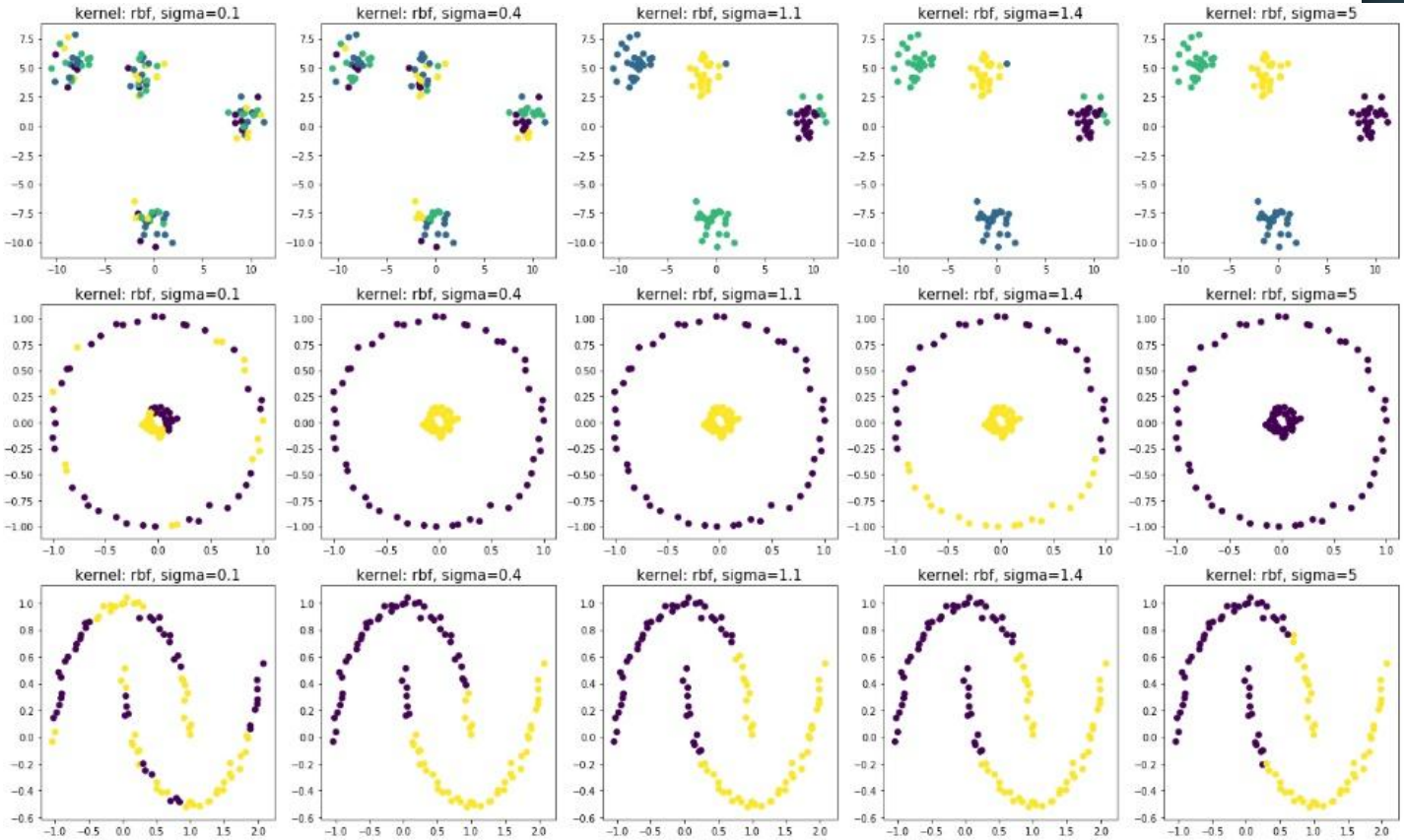
The outer circle seems to be divided in more clusters for increasing the exponent.

For some reason when gamma, r and m were set to -1, 1 and 3 it clustered the outer ring as one cluster and the centre as one cluster, but it only assigned two classes for the blobs.



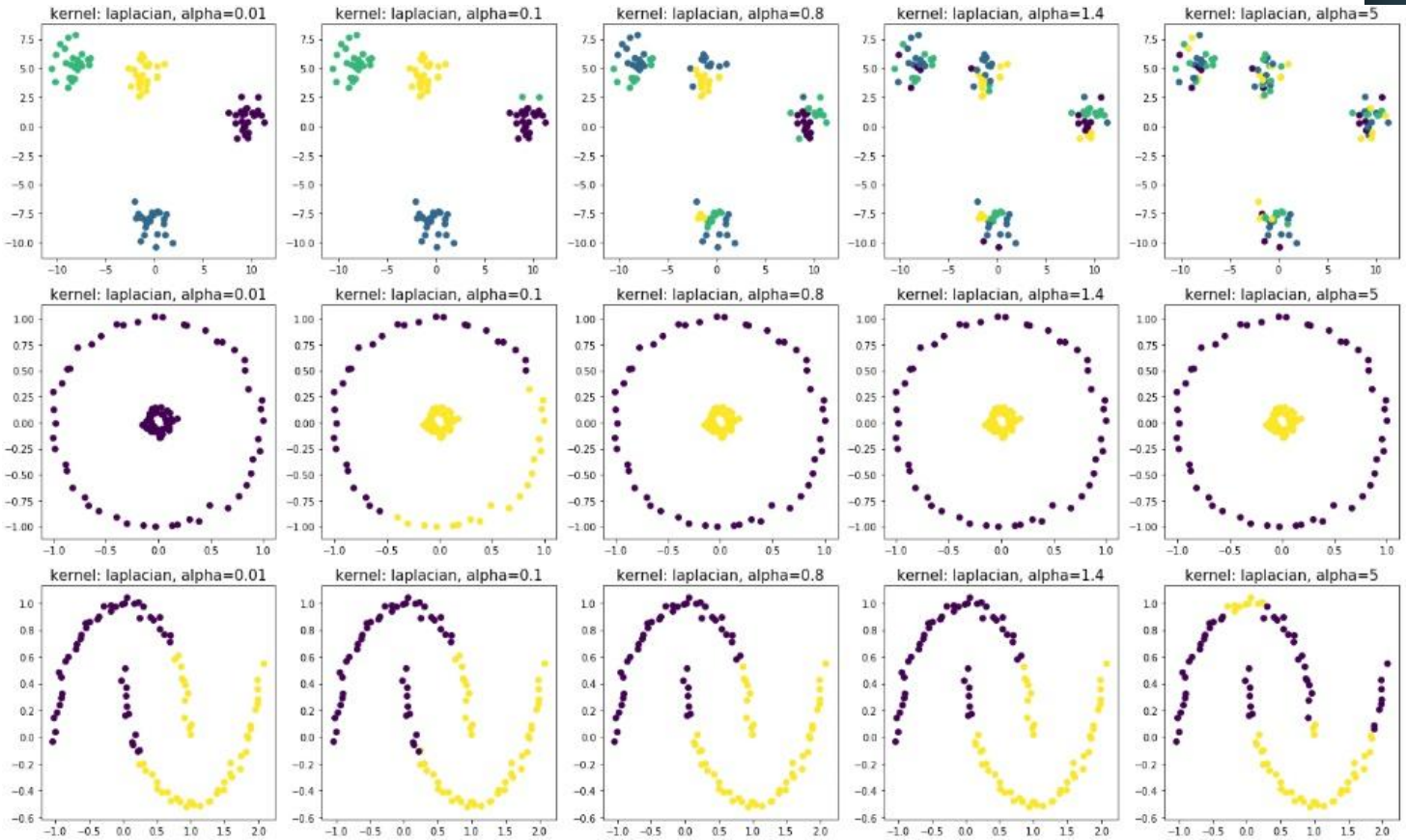
RBF kernel clustering exploration

See that for increasing sigma the blob clusters become more homogenous while the circles eventually turns into one cluster. There is a sweet spot for sigma when classifying the circles



Laplacian kernel clustering exploration

For low alpha we see that the blobs get one class each, but that the circles are assigned the same class. With increasing alpha the clusters of the blobs become more mixed and the circles become more homogene.



Results and discussion part 1

- For the circles-set, the linear, polynomial and sigmoid kernels tended to split the the center circle in half, all three of them uses the operation $x^T y$. The laplacian and rbf kernels tended to assign the center one class, both of them uses 1/(e to the power of a norm), with the difference being l1- and l2-norm.
- For some reason the polynomial kernel classified the circles such that the centre was one class and the outer circle was another when gamma, r and m were set to -1, 1 and 3.
- The rbf and laplacian kernel had very similar behaviours while varying their parameters (taking into account that the parameter is in the numerator for laplacian and denominator for rbf), which makes sense since both of them is 1/(e to the power of a norm). The value the kernel spits out is close to 1 when the data points computed are close, and goes to 0 quickly with increasing the distance. This behaviour can be either strengthened or weakened by changing the parameters. With rbf, a large sigma would make points far away from each other to generate higher values than if sigma was small, this increases the penalty when points further away from each other are assigned to the same cluster and when clusters are spread over a large area. With a small sigma the kernel is influenced more by close points than points far from each other, this could explain why the circles are classified as they are for smaller sigma and why the blobs are classified as they are for large sigma. The same reasoning can be applied to the laplacian kernel.

$$\arg\min_{i=1,\dots,K} \|\phi(\mathbf{x}_l) - \mathbf{m}_i\|^2 = \arg\min_{i=1,\dots,K} -\frac{2}{n_i} \sum_{C(\phi(\mathbf{x}_r))=i} k(\mathbf{x}_l, \mathbf{x}_r) + \frac{1}{n_i^2} \sum_{C(\phi(\mathbf{x}_r))=i} \sum_{C(\phi(\mathbf{x}_s))=i} k(\mathbf{x}_r, \mathbf{x}_s)$$

Results and discussion part 2

- To measure the quality of the clustering maybe one could look at the two sums when overfitting and when “successfully” labeling to see if there is a connection between them when overfitting and clustering correctly, or maybe train an AI on “good outcomes” and “bad outcomes” to find these connections for us.
- Maybe there is value in introducing some stochasticity/noise when converging the Z-matrix to try to avoid the sensitivity of initialization, since the outcome is deterministic from a given initialization. The “noise” could be something like randomly relabeling data points while running.
- From experience I can also say that it took some time to run the code, which implies that the method is computationally expensive (unless I made an inefficient code)
- Lastly, the clustering is very sensitive to the hyperparameters of the kernels, fine tuning them is essential.