Università degli studi di Trieste

Introduzione al Machine Learning

”Challenge two”

Martinelli Davide

April 21, 2024

0: Introduzione

In the challenge it is shown why it is better (or not) to use kernelised methods on two toy data sets, and find out which kernel is the most suitable for each. There are two main tasks:

1. Ridge Regression vs Kernel Ridge Regression
2. PCA vs Kernel PCA (with linearn SVM)

The toy dataset for the first part is made out of a gaussian distribution; while the second toy dataset is made using a sklearn function called "make_circle()" that make a large circle containing a smaller circle in 2d. Additionally, it will be present a third point that give the hint to play with "make_classifier()", from sklearn, that generate a random n-class classification problem and put into practice what learned during the first two points.

1: Ridge Regression

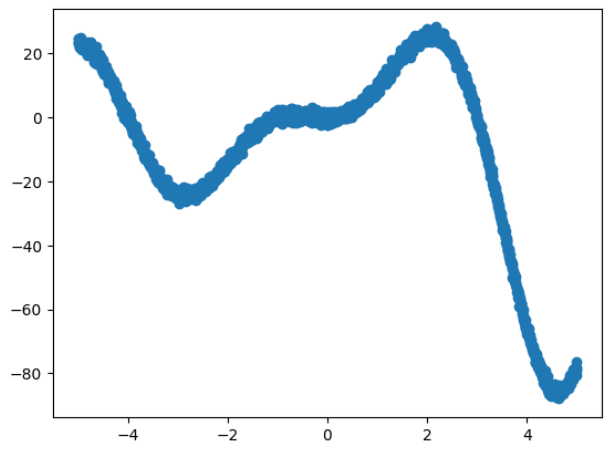
The first point quote:
Using the training and test data sets created in the following cell, what you have to do is:

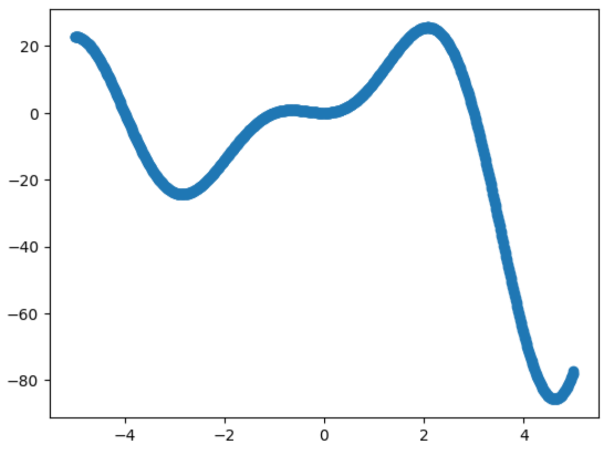
- *Fit a linear Ridge Regression model;*
- *Fit a Kernel Ridge Regression model with a Gaussian kernel and one with a Polynomial kernel. Through a grid search, see how different values for gamma for the Gaussian, and different degrees and values for the regularisation term for the Polynomial, change the line that gets fitted by the model;*
- *Lastly, fit one last KRR with the best kernel and best parameters that you found (the ones that minimise the test error).*

For each subtask, calculate the RMSE on the test data set and plot a visual representation of each fitted line, also when trying different values for the parameters.

Her's the code for the first toy dataset:

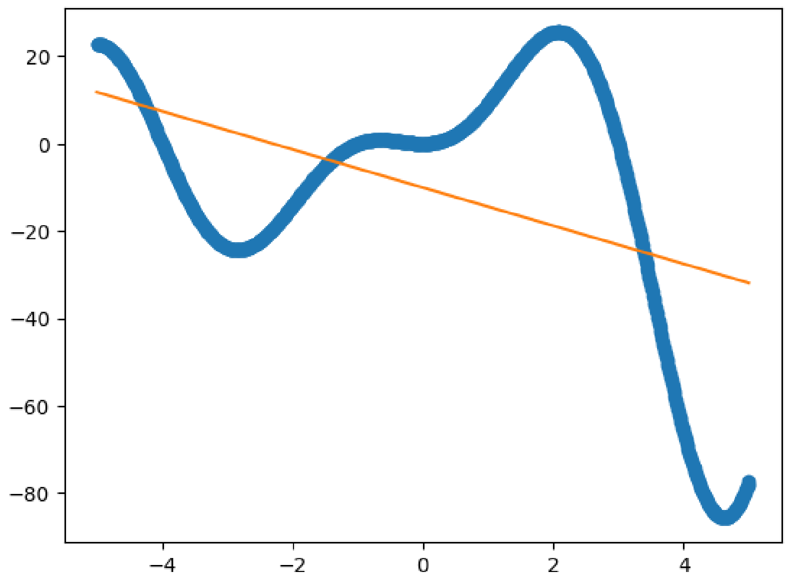
```
# create training dataset
eps = np.random.normal(0, 1, train_points)

X_train = np.linspace(-5, 5, train_points)
y_train = (X_train+4) * (X_train+1) * (np.cos(X_train)-1) * (X_train-3) + eps

# create testing data set

X_test = np.linspace(-5, 5, test_points)
y_test = (X_test+4) * (X_test+1) * (np.cos(X_test)-1) * (X_test-3)
```





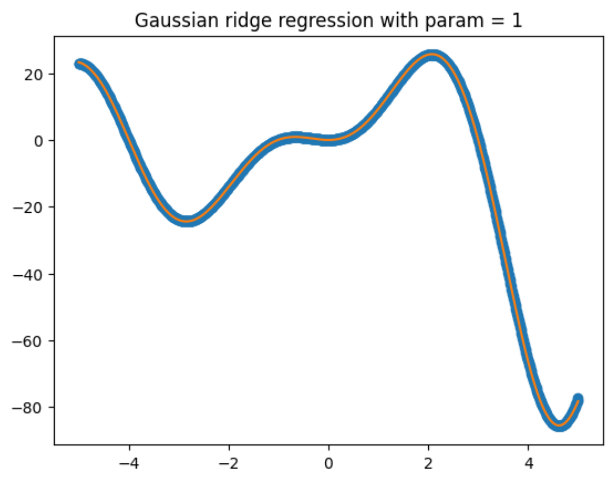
Her's the plot of how the train (left) and test (right) set look like. The train test is affected by some eps-noise.

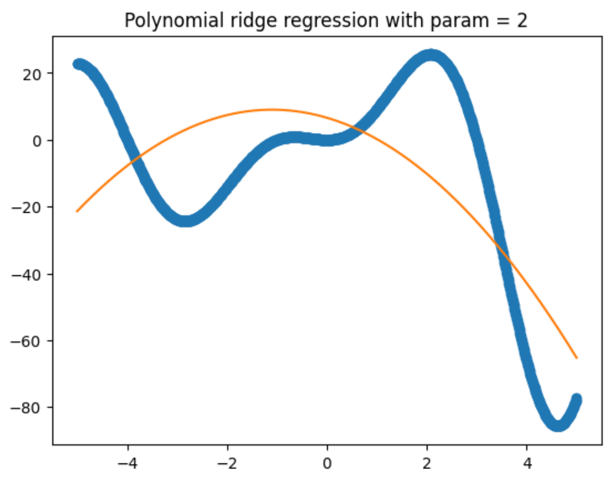
Let's try to fitting a simple ridge regression:



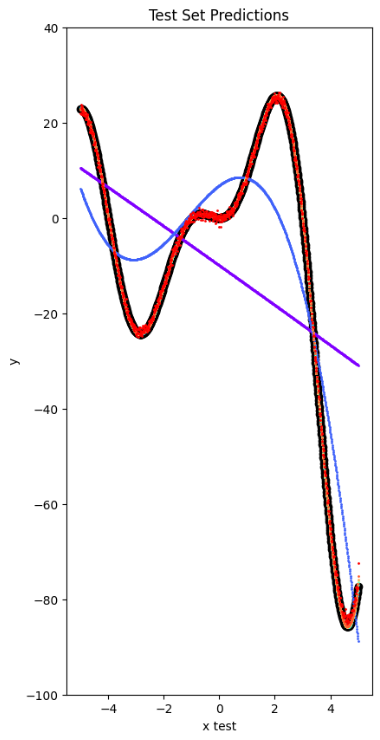
Notice how a simple linear ridge regressionhe is unable to follow the trend very well. In fact, we obtain an RMSE of 26.75.
I even tried implementing two models for ridge regression: using sklearn and doing the calculations by hand. Both gave the same results.

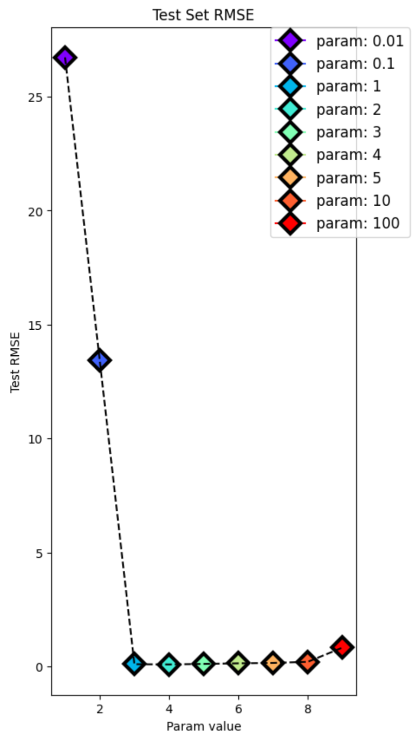
As asked, we try to use a kernel ridge regression (I opted for Gaussian, polynomial, or linear, but there are many others). Let's see the result of a gaussian (left) and then polynomial (right) kernel:



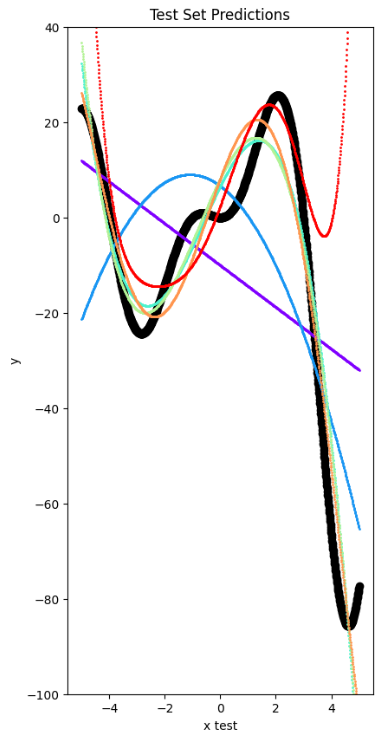


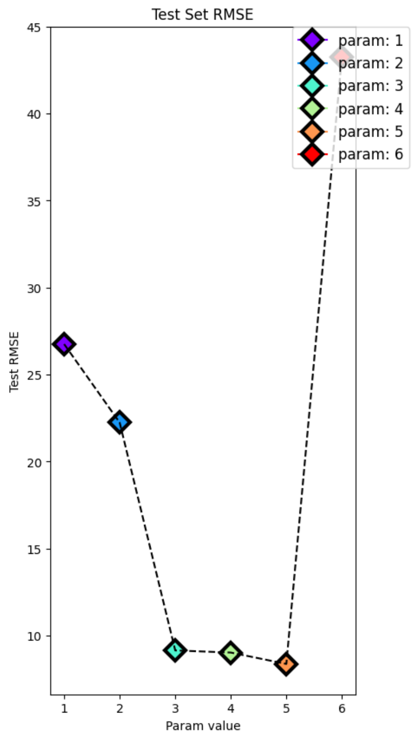
So, the gaussian ridge regression did a pretty good job with an RMSE below 0 at te first try and the polynomial achive a 22.244. Not bad for a first use.
But, even if the Gaussian kernel obtained an excellent value for the RMSE, the polynomial kernel needs to calibrate the parameter. Let's try to perform a gredy search to find the best parameter. Regarding the Gaussian kernel I chose to run the following list of values [0.01, 0.1, 1, 2, 3, 4, 5, 10, 100] for the gaussian, while for the polynomial [1, 2, 3, 4, 5, 6].





All RMSE obtained by the gridy search: [26.72369213977438, 13.43496678148916, 0.10587295795498074, 0.09540717686153567, 0.11862871066809258, 0.14373239361219659, 0.16167997367073278, 0.21065368157323447, 0.8330338734413484] The best value is: 0.09540717686153567 obtained with the parameter 2





All value obtained by the gridy search: [26.754441197054053, 22.244156592589746, 9.150339457987391, 9.028733335172165, 8.37436296567974, 43.26123133354297] The best value is: 8.37436296567974 obtained with the parameter 5

As we expected, we came to the conclusion that gaussian kernel is the best choice for the toy dataset. Overall, the RMSE are very likely to be small, unlike the polynomial kernel. The best param obtained from the gaussian is 2, with an $RMSE < 10^{-1}$. On the other hand, polynomial with a param equal to 5 and an $RMSE \sim 8$.
Note: I choose to stop with a pram = 6 with the polynomial cause the RMSE was skyrocketing to high value and with a param = 100 for the gaussian because the RMSE was not decreasing but slightly increasing.

2: PCA

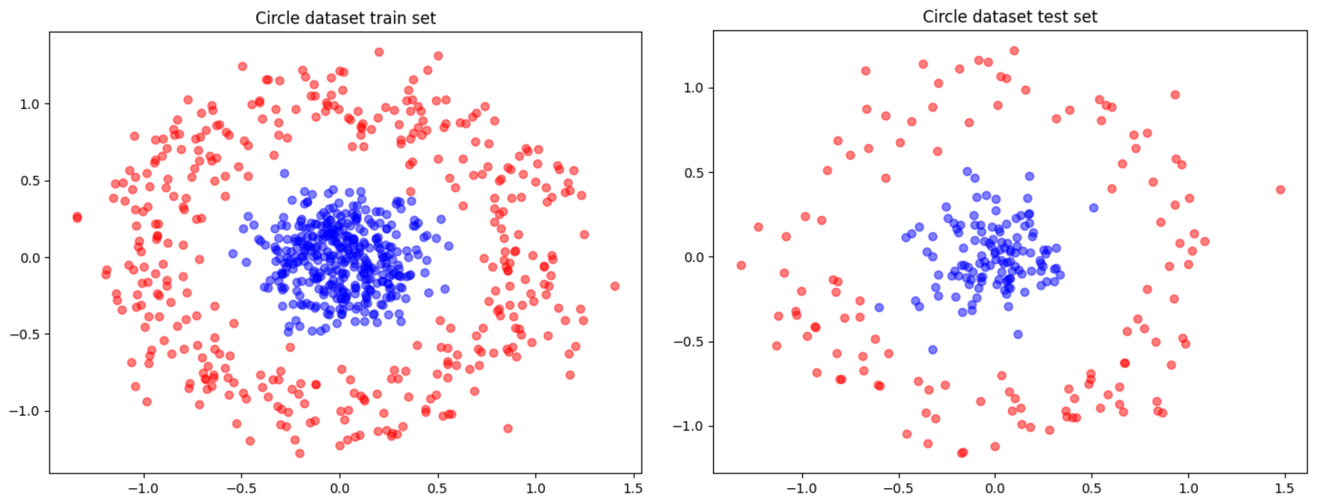
The second point quote:
Using the training and test data sets created in the following cell, what you have to do is:

- *Fit a PCA model;*
- *Fit a Kernel PCA model with a kernel of your choice.*

For each subtask, plot a visual representation of the projections and verify the accuracy of that kernel on the test data set using SVM.

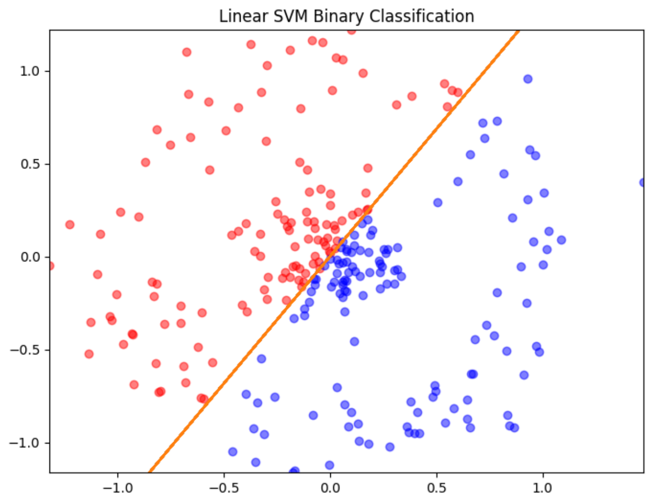
Her's the code for the second toy dataset:

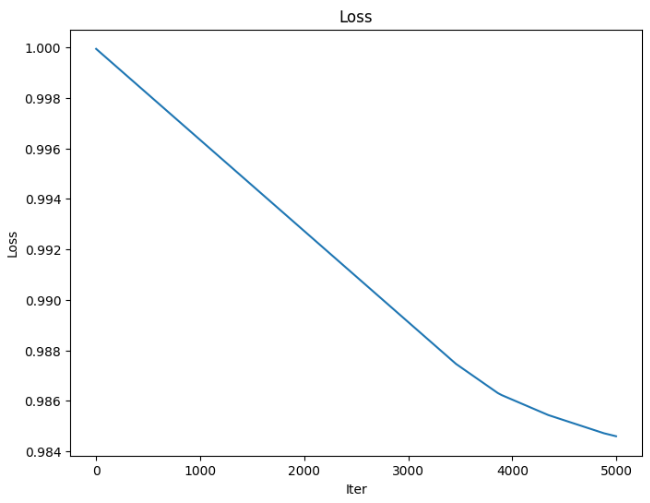
```
X, y = make_circles(n_samples=1000,
                    noise=0.15,
                    factor=0.2,
                    random_state = 0)
X_train, X_test, y_train, y_test = train_test_split(X, y,
                                                    stratify = y,
                                                    random_state=0)
```



Her's the plot of how the train (left) and test (right) set look like. Both the dataset are effected by some noise.

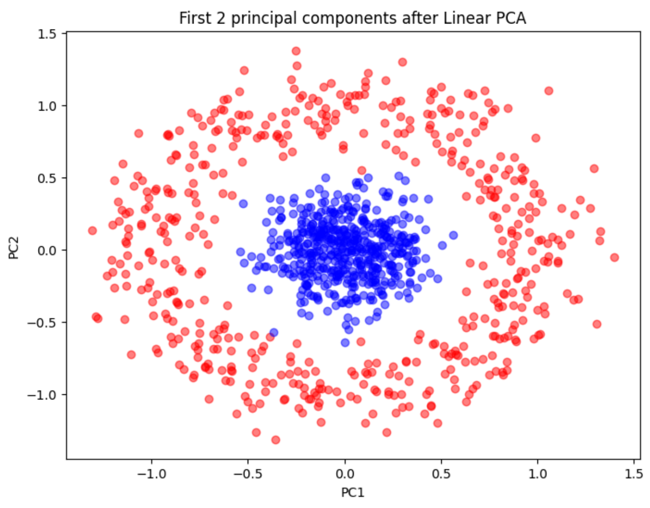
As asked, let's try to train a linear SVM and predict the test set.

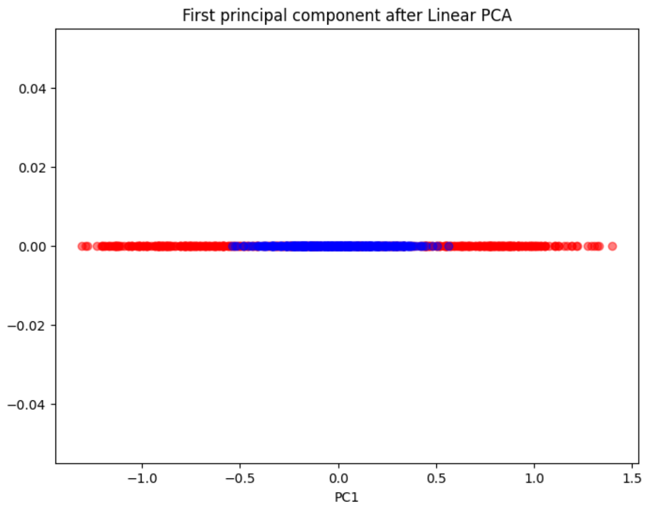




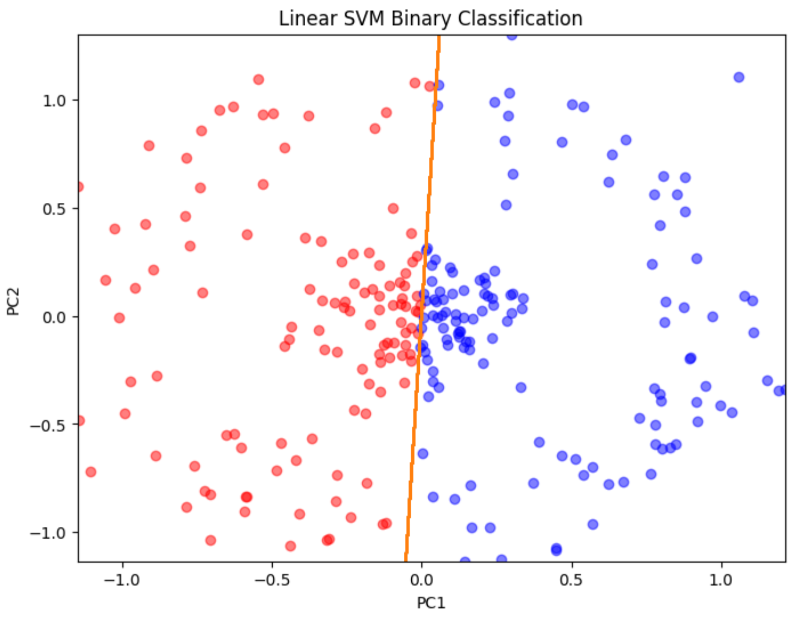
Similar with the ridge regression, these are results that we expect. A linear model cannot follow a dataset with this shape. This can be seen from the hyperplane that divides the test set in half. The number of iteration make no difference. In 5000 steps the loss decreased by 2% and end with an accuracy of 50%.

We just need to organize the shape to make it easier to predict for a linear model. So, we use PCA to reprocess the first 2 components of the dataset.



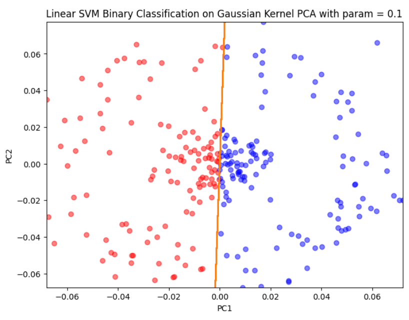


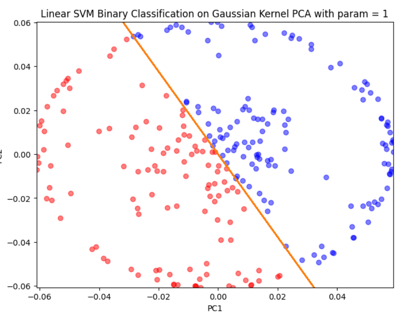
Her's the plot of how the first two (left) and just the first (right) component look like. Nothing has change, that's because PCA is also a linear method. Without this important peculiarity, PCA simply centers the dataset towards the intersection of the x and y axes.

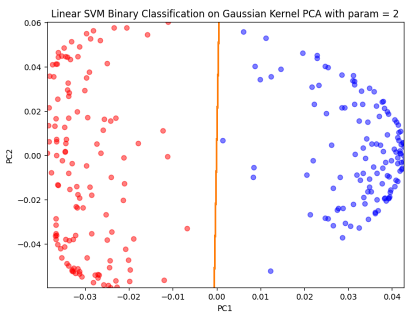


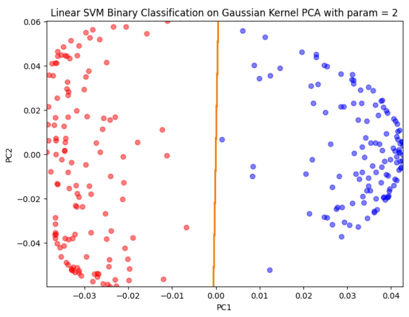
Since no changes were made with PCA, training a linear SVM model will lead to the same results, with the same loss and accuracy.

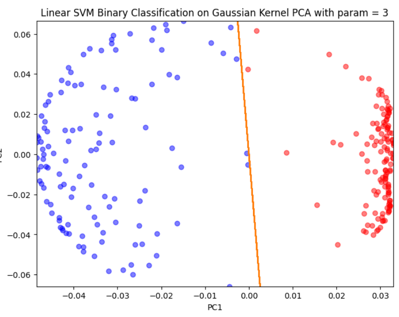
Why don't we try implementing a PCA kernel version, similar to how we did kernel ridge regression? Once I have made the right adjustments to the code, which only consist of doing dimensional transformations on the kernelized matrix instead of directly on the dataset; then, I'm going to do something similar to what I did with gready search: run a PCA kernel, first with Gaussian and then with polynomial, using different parameters. As done for the other points, I train an svm on top of the data obtained from the PCA kernel

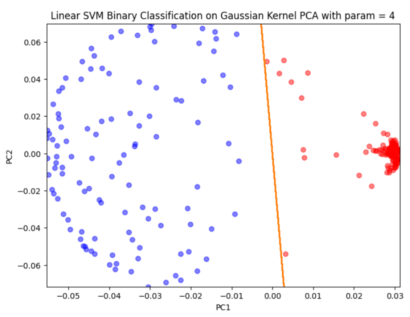








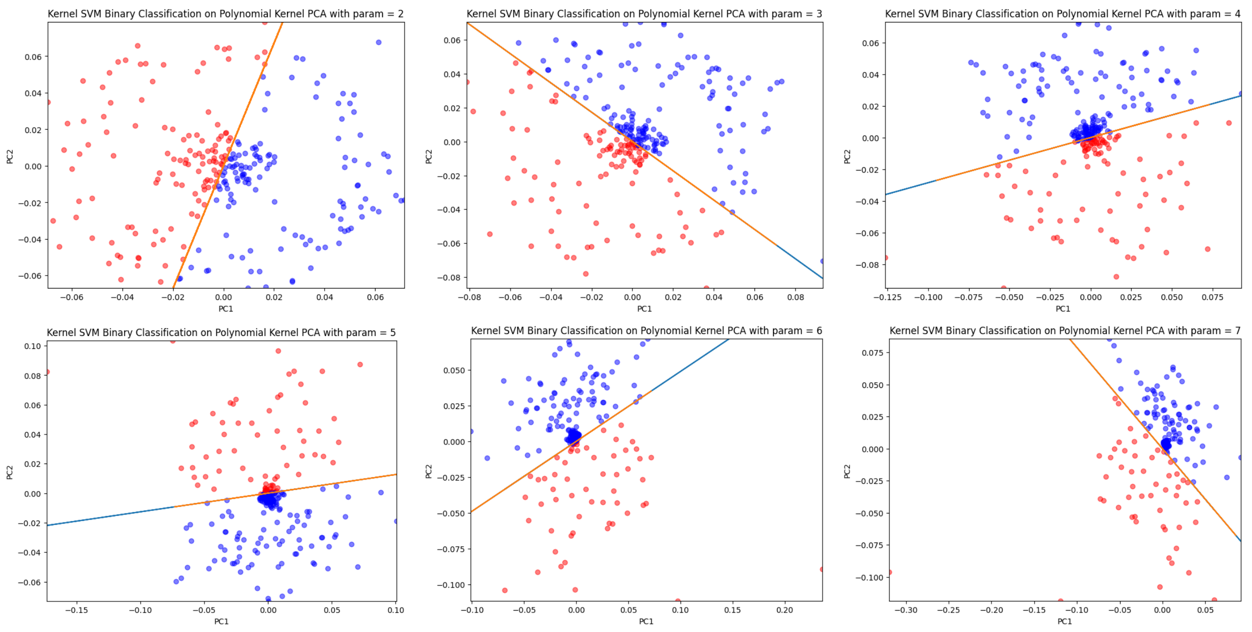




The first image with a param = 0.1 makes the Gaussian kernel linear (i.e. as if no transformation had been applied); the second image with a param = 1 gives us a glimpse of the beginning of a transformation, which however does not give good results from SVM; the third image with a param = 2 makes us start to see interesting transformations with an egg shape and a clear separation between the two clusters; by increasing the parameter to 3 and 4, we notice that the dataset begins to sharpen in the right region of the points. Now, let's take a look at the accuracies to understand which param value obtained better accuracy in the test set.

- param = 0.1, acc = 48%
- param = 1, acc = 48,4%
- param = 2, acc = 99.2%
- param = 3, acc = 98%
- param = 4, acc = 95.2%

Now let's try with polynomial kernel.



Unlike the Gaussian kernel, the polynomial one maintains the circular structure of the dataset and squashes all the points towards the center, keeping 1 or 2 points as extremes. Linear SVM is therefore not effective for clustering this type of dataset, since even with the best parameter for PCA, the accuracy increases very little.

- param = 2, acc = 47%
- param = 3, acc = 50%
- param = 4, acc = 51.6%
- param = 5, acc = 57.2%
- param = 6, acc = 66.4%
- param = 7, acc = 69.2%

In conclusion, Gaussian PCA is more suitable for allowing linear separation with SVM, while polynomial PCA has some problems but still manages to give some good results.

3: PCA pt. 2

I won't stop to analyze this point too much since it is like the previous one but with more freedom. What we've done is take the "make_classifier()" function and play around with it to see what the PCA kernel can do. In particular on two datasets: a simpler one with well-separated clusters, the other with noise and overlapping point clouds.