Unsupervised Learning and Large Scale Problems

James O'Reilly
james.oreilly@student.kuleuven.be

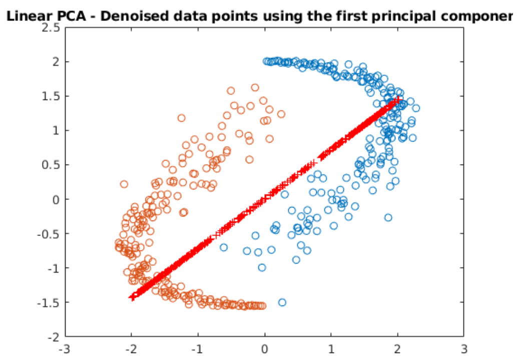
1 Kernel Principal Component Analysis

Describe how you can do denoising using PCA. Describe what happens with the denoising if you increase the number of principal components.
PCA can be used to reduce the noise present in the data, or more specifically, increase the signal-to-noise (SNR) ratio of the data. Using PCA, we can identify the dimensions of the data which have the most variance, as measured by the eigenvalues associated with the covariance matrix of the data. These dimensions with the most variance are also the dimensions which contain most of the signal that we are interested in. In most cases, signal is not evenly distributed across the dimensions of the data. Noise, however, is most often distributed across all dimensions of the data. Therefore, by using only the dimensions with the highest eigenvalues (the most signal), we can increase the signal-to-noise ratio, and the influence of noise is reduced. is based on linear functions and then they are good only in the presence of additive noise. Moreover, if the variables of the signal are non-linearly correlated, PCA won't be able to remove additive noise. However, as PCA is based on linear transformations this denoising is effective only in the presence of additive noise. Moreover, if the variables of the signal are non-linearly correlated, PCA won't be able to remove additive noise.

As the number of principal components is increased, the amount of ratio of signal to noise added per component decreases. So while adding components increases the amount of signal present, it also decreases the SNR of the data.

Compare linear PCA with kernel PCA. What are the main differences? How many principal components can you obtain?
I will compare linear PCA and kernel PCA at a high level along a number of axes, including data structure, computational cost, reconstruction, number of hyperparameters, and number of PCs.

- **Data structure:** Simply, kernel PCA with a nonlinear kernel can capture non-linear patterns in the data (see Figure 1). Standard PCA cannot. This is the central motivation for kernel PCA in the first place. In the context of dimensionality reduction, this means that kernel PCA can reduce the data more, as it can find non-linear low-dimensional manifolds which PCA can not.



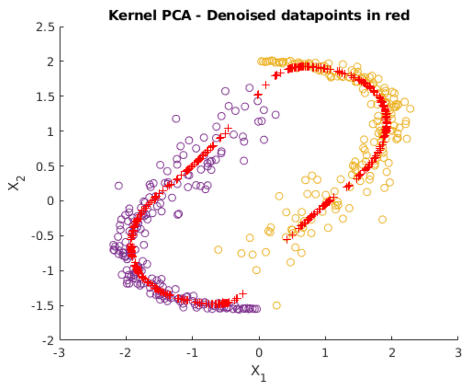


Figure 1: *(left)* Standard on the yin-yang dataset. *(right)* Kernel PCA on the yin yang dataset with 6 principal components and $\sigma = 0.4$.

- **Number of hyperparameters:** Using standard PCA for dimensionality reduction or denoising requires only choosing the number of dimensions to map to. Kernel PCA requires that the kernel function and any associated parameters are also specified. For example a Gaussian RBF kernel and its bandwidth parameter σ. This means that the kernel and kernel parameters often need to be tuned.
- **Reconstruction:** Standard PCA allows for the approximate reconstruction of the original (pre-image) data via an inverse mapping from the feature space back to the pre-image space. In kernel PCA the reconstruction of pre-image data is not possible via an inverse mapping, as this mapping generally does not exist, and only a few elements in the feature space have a valid pre-image in the input space [1]. It is sometimes possible to approximate this mapping, although this can be computationally expensive if the feature space is especially high-dimensional (or infinite dimensional, in the case of Gaussian RBF kernels). I used Kwok's paper *"The Pre-Image Problem in Kernel Methods"* and Mika's paper *"Kernel PCA and De-Noising in Feature Spaces"* as a reference for this [2, 3].
- **Number of PCs:** Given P features and m learning instances, standard PCA can extract a maximum of P principal components, while kernel PCA can extract a maximum of m. This is because in standard PCA, the eigenvalue decomposition is performed on the covariance or correlation matrix Σ, which is $P \times P$, while in kernel PCA, eigenvalue decomposition is performed on the kernel matrix K, which is $m \times m$.

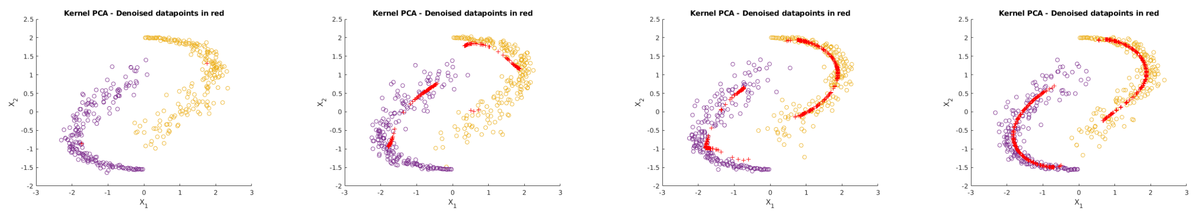


Figure 2: kPCA with 1, 2, 3, and 4 principal components (from left to right).

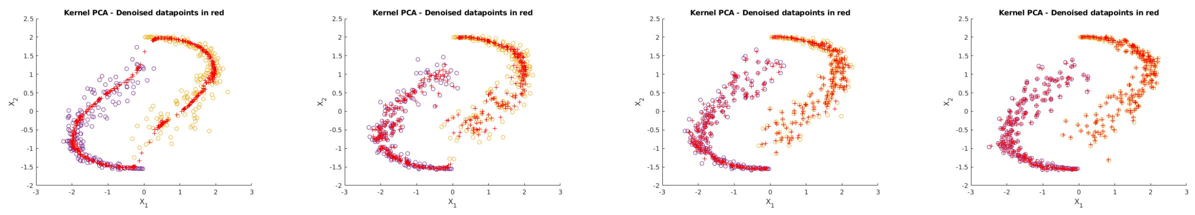


Figure 3: kPCA with 10, 15, 20, and 30 principal components (from left to right).

From Figures 2 and 3 below, we can see that too few PCs results in poor performance and reconstruction error will be much higher. Choosing too many PCs will give low reconstruction error, but poor dimensionality reduction (thus defeating the purpose). The optimal number of PCs will be the smallest number which successfully captures the non-linear trend and minimises the reconstruction error. My estimate would be four or five.

For the dataset at hand, propose a technique to tune the number of components, the hyperparameter and the kernel parameters.
The optimal kernel, kernel parameters, and number of components used in kernel PCA can be selected via cross validation. In order to do this cross validation, we need some metric for model performance in kPCA. In standard PCA, the reconstruction error on the test set is used as a performance metric, as measured in the target space. However, this method is not straightforward for kPCA as different kernels give different target spaces and so the reconstruction errors are not directly comparable.

The solution is to compare the reconstruction error in the original input space, and not in the target space. The data point(s) used for testing in cross-validation exist in this original space, but its kPCA reconstruction lives in the subspace of the target space spanned by the principal components. We must therefore use an approximate pre-image mapping to find the pre-image in the original space that would be as close as possible to this reconstruction point, and then measure the reconstruction error as the distance between the test point and this pre-image.

As a reference, I used Fukumizu and Alam's paper *"Hyperparameter Selection in Kernel Principal Component Analysis"*, in which they outline the difficulties of measuring reconstruction error for kPCA and discuss the method given above for choosing hyperparameters, optimal kernel and the number of kernel principal components using cross-validation on the reconstruction errors of pre-images [4].

2 Spectral Clustering

Explain briefly how spectral clustering works.
Before I continue, I should note that this explanation borrows heavily from this article. Spectral clustering is an unsupervised clustering technique based graph theory, in which communities of nodes in a graph are clustered based on the edges connecting them. Spectral clustering uses the eigenvalues of the Laplacian matrix of the graph. The Laplacian matrix is first constructed by subtracting the adjacency matrix from the degree matrix. The diagonal of the Laplacian is the degree of the nodes in the graph, and the off diagonal is the negative edge weights. The eigenvalues of the Laplacian has some interesting properties which can be used to identify communities in the graph.

The first nonzero eigenvalue is called the **spectral gap**. The spectral gap gives information on the connectivity of the graph. If this graph is fully connected (all pairs of nodes share an edge), then the spectral gap would be the number of nodes in the graph. The second eigenvalue of the Laplacian is the **Fiedler value**, and the corresponding vector is the Fiedler vector. The Fiedler value approximates the minimum graph cut needed to separate the graph into two connected components. Each value in the Fiedler vector gives information about the side of the cut to which each node belongs. In general, the first large gap between eigenvalues indicates the number of clusters expressed in the data: having four eigenvalues before the gap indicates that there is likely four clusters. The vectors associated with the positive eigenvalues before this gap should give information about which three cuts need to be made in the graph to assign each node to one of the approximated clusters. A matrix of these vectors is then constructed and k-means clustering is performed to determine the assignment of each node in the graph.

This technique can be extended to arbitrary data by constructing some graph formulation of the data, for example by constructing a graph of nearest neighbours or defining some affinity matrix between all the points. However, Spectral clustering has some limitations. For instance, it cannot handle big data without using approximation methods like the Nystrom algorithm and the generalisation to out-of-sample data is only approximate [5]. These issues prompted the development of kernel spectral clustering (KSC), which is a LS-SVM model used for clustering instead of classification [5]. KSC allows for parameters such as the number of clusters to be tuned is the LS-SVM framework, and gives the option to select application-specific kernels. KSC also learns an embedding in the Laplacian eigenspace during training, which allows for accurate prediction of cluster membership by projecting data into this embedding. Lastly, by enforcing sparsity using incomplete Cholesky decomposition and L_1 and L_0 sparsity penalties, the KSC model can scale well to large scale data.

What are the differences between spectral clustering and classification?
There is the basic difference that one method is unsupervised and the other is supervised or semi supervised.

What is the influence of the sig2 parameter on the clustering results?
The kernel matrices and clustering results for different values of σ are given in Table 1. It has been the case throughout these exercises, in both the classification and function estimation applications, that the sigma parameter determines the range of influence of the data points by determining the bandwidth of the kernel. The formula in the case of RBF is given by

$$K\left(x_i, x_j\right) = \exp\left(-\left\|x_i - x_j\right\|_2^2 / \sigma^2\right) \tag{1}$$

In the case of clustering, the sigma parameter has a similar function. It determines whether any given points in the dataset are deemed to share cluster identity, based on their distance from each other as evaluated by this kernel function. Increasing sigma increases the distance at which points share a cluster identity, as can be clearly seen in the kernel matrix and clustering results between $\sigma = 0.2$ and $\sigma = 0.5$. This is also evident in the sorted kernel matrix when $\sigma = 0.001$, wherein the bandwidth is so small that very few points share cluster membership. This is also evident in the dimensions onto the 2nd and 3rd principal component for different σ values (see Figure 4).

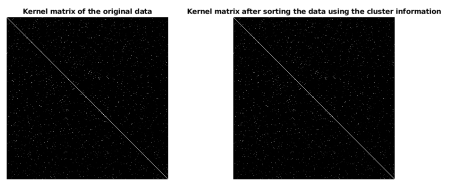
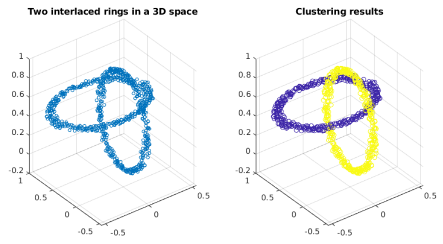
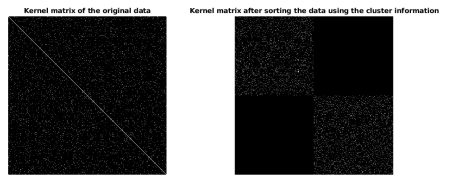
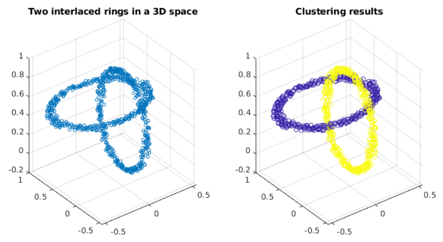
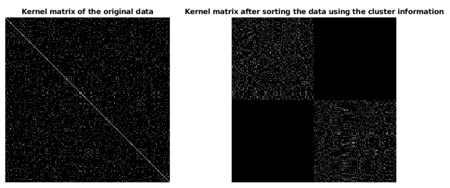
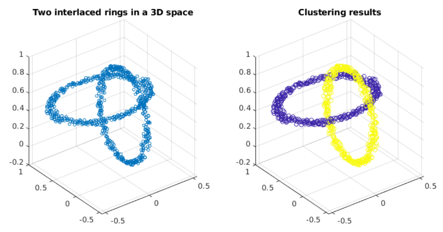
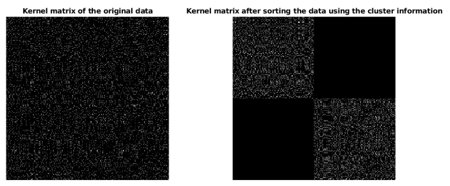
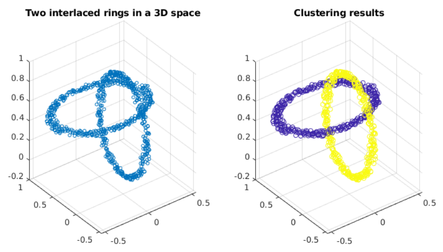
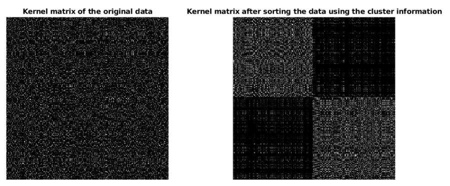
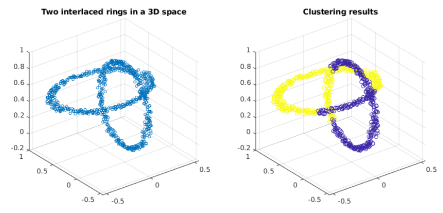
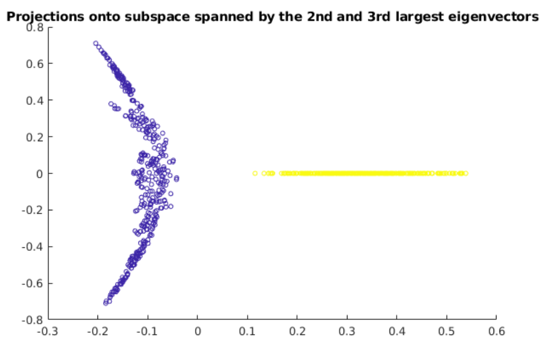
σ	Kernel Matrices	Clustering Results
$\sigma = 0.001$		
$\sigma = 0.005$		
$\sigma = 0.01$		
$\sigma = 0.02$		
$\sigma = 0.05$		

Table 1: Kernel matrices and clustering results for different values of σ



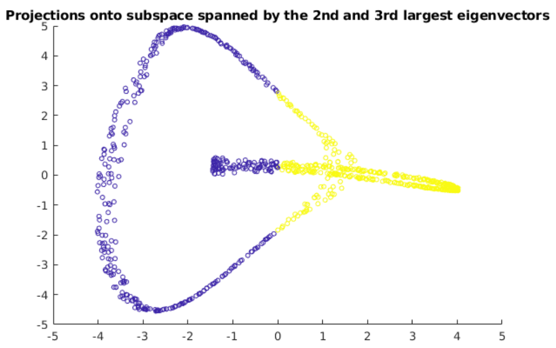


Figure 4: *(left)* Projection for $\sigma = 0.001$ *(right)* Projection for $\sigma = 0.05$

3 Fixed-size LS-SVM

In which setting would one be interested in solving a model in the primal? In which cases is a solution in the dual more advantageous?
To answer this concretely, I will first give the primal and dual formulations, in the context of hard-margin SVMs. The formulations and arguments are similar for soft margins. Given the data points $x_1, \ldots, x_n \in \mathbb{R}^d$ and labels $y_1, \ldots, y_n \in \{-1, 1\}$ the hard margin SVM primal problem is given by

$$\min_{w,w_0}(\tfrac{1}{2}w^T w), \quad \text{such that} \quad \forall i : y_i \left(w^T x_i + w_0\right) \geq 1 \tag{2}$$

which constitutes a quadratic programming problem with $d+1$ variables to be optimised and i constraints. The dual formulation is given by

$$\begin{aligned} &\max_\alpha(\textstyle\sum_{i=1}^n \alpha_i - \frac{1}{2}\sum_{i=1}^n \sum_{j=1}^n y_i y_j \alpha_i \alpha_j K(x_i, x_j)) \\ &\text{such that} \quad \forall i : \alpha_i \geq 0 \quad \text{and} \quad \textstyle\sum_{i=1}^n y_i \alpha_i = 0 \end{aligned} \tag{3}$$

Using the mapping

$$x \to \phi(x), \text{ where } \phi : R^d \to R^D \tag{4}$$

and the kernel function

$$K(u, v) =< \phi(u), \phi(v) > \tag{5}$$

This dual formulation constitutes a quadratic programming problem with $n + 1$ variables to be optimised, n inequality and n equality constraints. If D is too large, then computing this mapping (and solving for W using the primal formulation) is too expensive. The dual formulation with the kernel trick allows one to compute the kernel function K instead of the dot-product in the basic dual formulation. This is not possible in the primal formulation where one must explicitly compute the mapping for each data point.

The advantages and disadvantages of solving a model in each of the formulations are now quite clear. In the case where there is a lot of data (n is large), it is efficient to solve a model in the primal. In the case where the data is high-dimensional (such as with microarray or transcriptome data) it more advantageous to solve the model in the dual.

What is the effect of the chosen kernel parameter sig2 on the resulting fixed-size subset of data points? Can you intuitively describe to what subset the algorithm converges?
For the non-linear case, one can solve the primal formulation by employing the Nyström method to get an approximation to the feature map, assuming a small fixed-size M, where M is the dimensions of the kernel matrix K. The working set of M support vectors is selected by the quadratic Renyi entropy criterion. In this

fixed-size formulation, the kernel parameter σ affects the fixed-size subset of support vectors. By observing the distribution of support vectors after convergence with different values of σ, it is clear that σ determines the dispersion or distance of the support vectors away from the mean of the data (see Figure 5. Large values of σ give a dispersed set of support vectors, and small values of σ give a tightly clustered set of support vectors toward the center of the data.

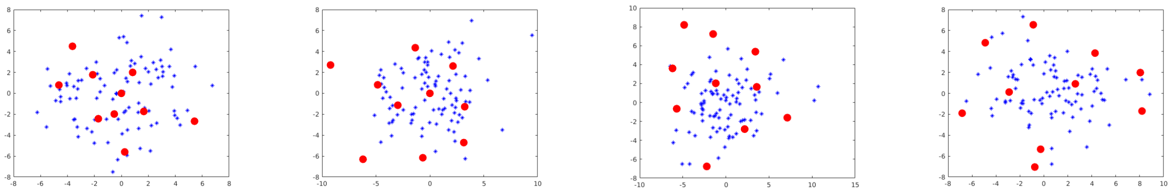


Figure 5: Determining support vector subsets of size 10, with $\sigma = 0.01$, 0.1, 1, and 10 (from left to right).

The question is, why use quadratic Renyi entropy to find a subset of size m, such that they best approximate the kernel matrix? The goal is that the selected support vectors should represent the main characteristics of the whole training data set [6]. By maximising the entropy criterion

$$H_R = -\log \frac{1}{M^2} \sum_{ij} \Omega_{(M,M)_{ij}} \tag{6}$$

it is most likely that these support vectors accurately represent the training data. It also ensures that the support vectors are used efficiently, with no two vectors representing the same characteristic of the data, as we have sparsity in mind, it wouldn't be logical to waste our precious support vectors.

Compare the results of fixed-size LS-SVM to ℓ_0-approximation in terms of test errors, number of support vectors and computational time.
A second level of sparsity can be introduced into the PFS-LSSVM approach, resulting in a further reduced set of support vectors which give a highly sparse solution, allowing the model to scale to large datasets due to improved memory constraints and computational costs. Figures 6 and 7 show the results comparing test errors, number of support vectors, and computational time for these two approaches. From the figure we see that the ℓ_0-approximation results in a similar error estimate which is also more stable. Both the standard FS-LSSVM and the ℓ_0-approximation result in the same number of support vectors (18), which is interesting considering that the ℓ_0-approximation is supposed to lead to a further reduced set of support vectors and a more sparse representation (though perhaps my understanding is incorrect here) [7]. Lastly, the ℓ_0-approximation has a worse estimate for computational time taken. The sparsity-vs-error trade-off of the ℓ_0-approximation trades increased error for decreased memorisation and computational cost, by inducing sparsity. This does not seem to be the case here.

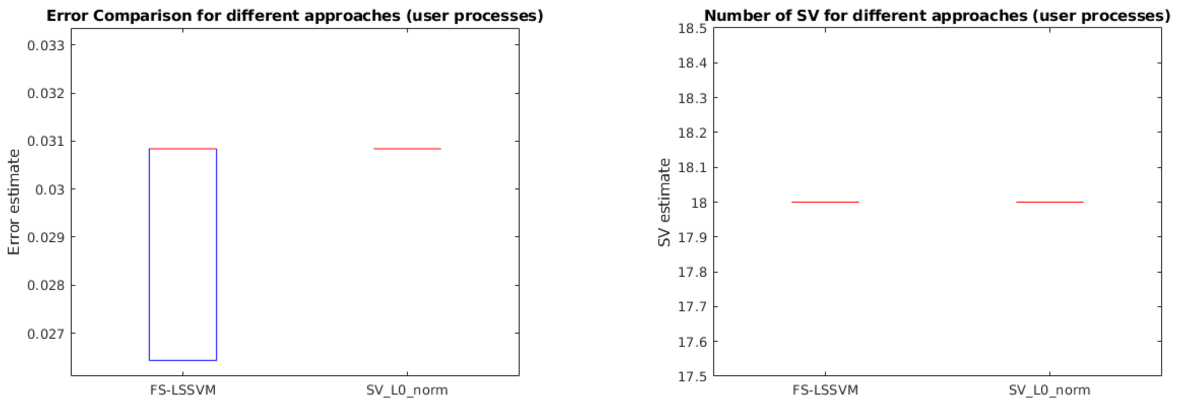


Figure 6: *(left)* Error comparison for FS-LSSVM and ℓ_0-approximation. *(right)* Number of support vectors.

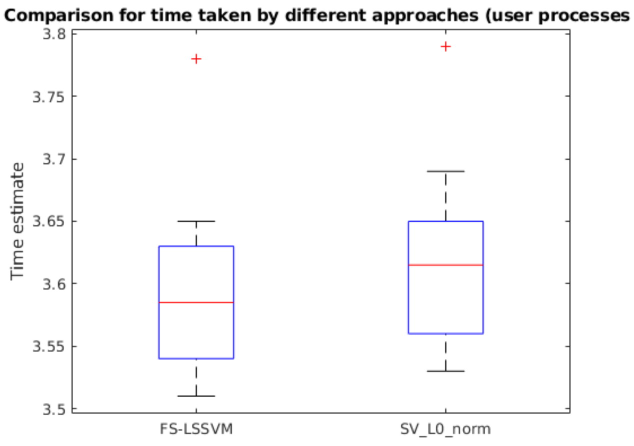


Figure 7: Time estimate for FS-LSSVM and ℓ_0-approximation

4 Homework Problems

4.1 Kernel Principal Components Analysis

In this homework problem, we do image denoising using kernel PCA. The dataset used in this exercise consists of images of handwritten numerals (0 to 9), extracted from a collection of Dutch utility maps. `sig2` is calculated as the mean of the variances of each dimension times the dimension (number of features) of the training data.

Illustrate the difference between linear and kernel PCA by giving an example of digit denoising for noisefactor = 1.0. Give your comments on the results (based on visual inspection).
The results are given in Figure 8

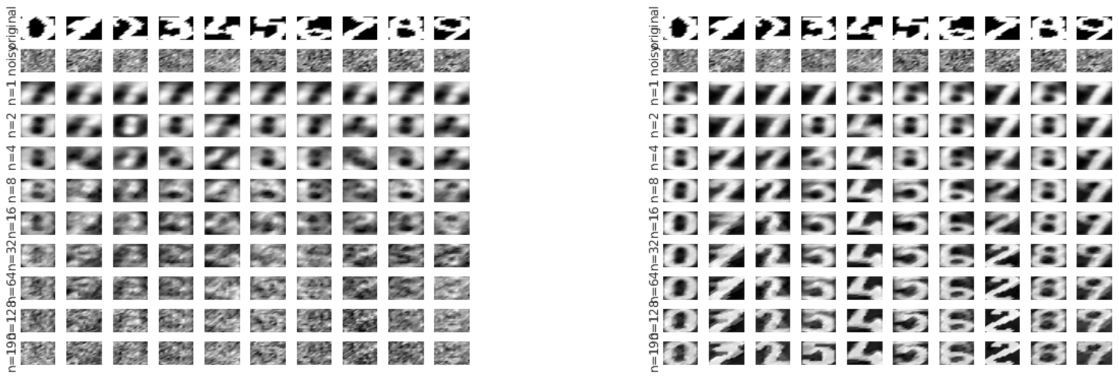


Figure 8: Denoising with PCA *(left)* and kPCA *(right)* with a noise factor of 1.

What happens when the sig2 parameter is much bigger than the suggested estimate? What if the parameter value is much smaller? In order to investigate this, change the sigma factor parameter for equispaced values in logarithmic scale.
The results are given in Figures 9 and 10.

Investigate the reconstruction error on training (Xtest) and validation sets (Xtest1 and Xtest2), as a function of the kernel PCA denoising parameters. Select parameter values such that the error on the validation sets is minimal. Can you observe any improvements in denoising using these optimised parameter settings?

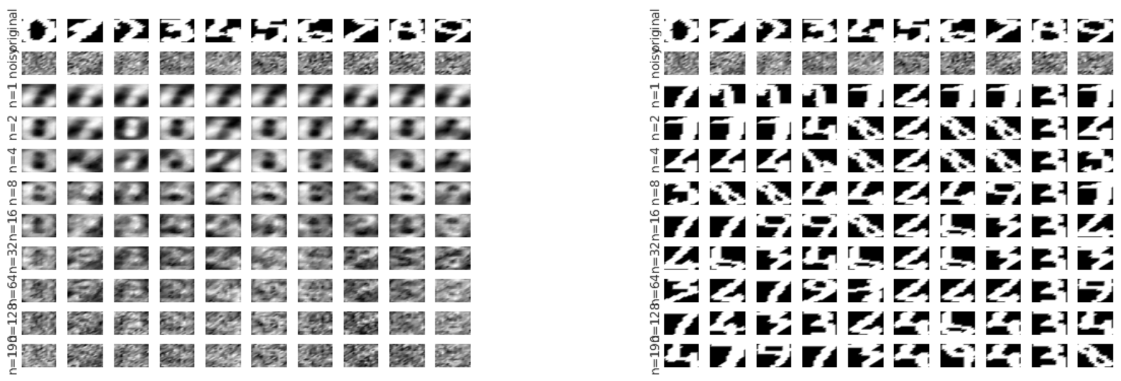


Figure 9: Denoising with PCA *(left)* and kPCA *(right)* with a sigma factor of 0.01.

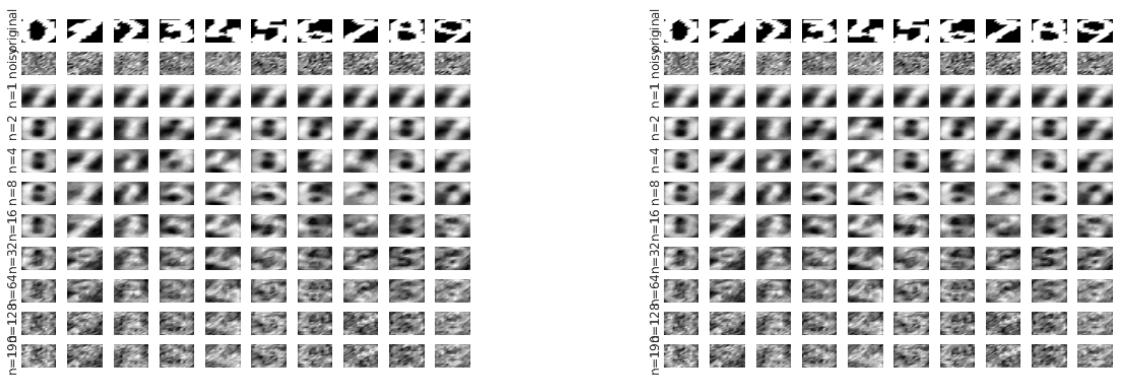


Figure 10: Denoising with PCA *(left)* and kPCA *(right)* with a sigma factor of 10.

4.2 Fixed-size LS-SVM

4.2.1 Shuttle

Explore and visualise (part of) the dataset. How many datapoints? How many and meaning of attributes? How many classes? What is to be expected about classification performance?
The dataset has 58,000 datapoints. There are nine attribute columns all of which are numerical and one class column, with seven classes. The frequency of the classes is given in Figure 11. The frequencies indicate that it might be difficult to reliably predict classes 2, 3, 6, and 7, given the limited amount of data that we have for these classes.

Visualise and explain the obtained results.
The results from the LS-SVM classification model and ℓ_0-approximation are given in Figure 12. The ℓ_0-approximation has lower error, the same number of support vectors, and slightly increased cost compared to the standard LS-SVM. I believe this indicated that a more sparse model can be afforded, sacrificing error, increasing sparsity, and increasing the computational efficiency of training the model on this large dataset.

4.2.2 California

Explore and visualise (part of) the dataset. How many datapoints? How many and meaning of attributes?
The dataset has 20,640 datapoints with nine numerical independent variables and one numerical dependent variable (ln(median house income)).

Visualise and explain the obtained results.
The results from the LS-SVM classification model and ℓ_0-approximation are given in Figure 13. The error and number of support vectors are the same using both approaches. Again, the sparsity is the same, despite the

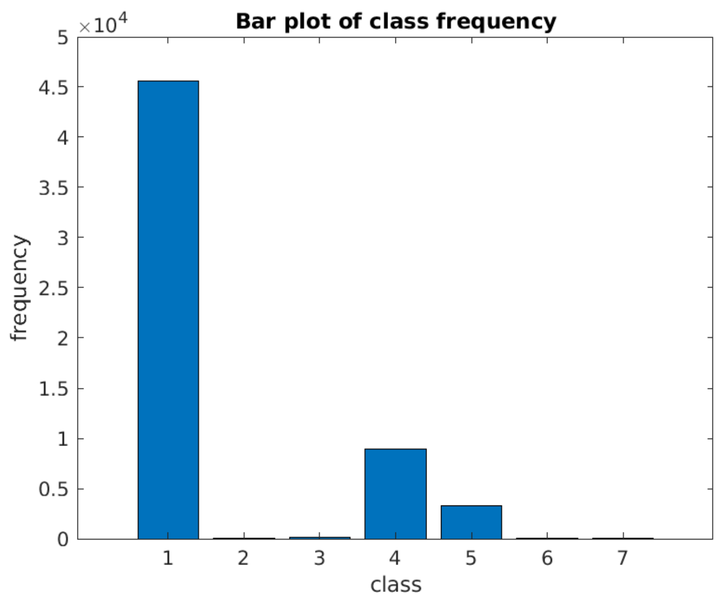


Figure 11: Bar plot showing class frequency for the shuttle dataset.

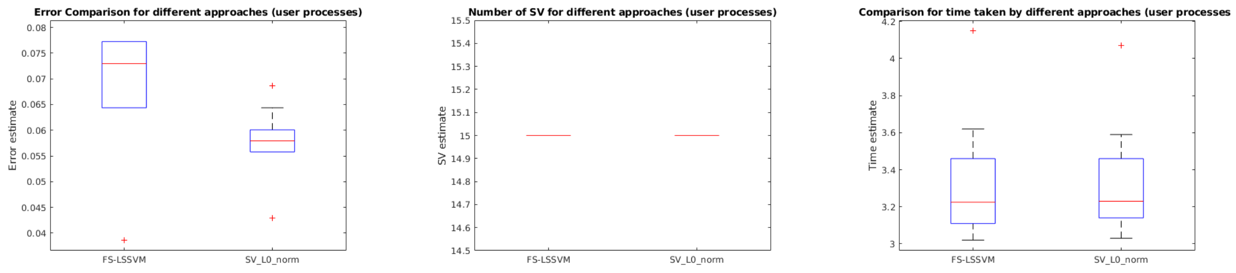


Figure 12: *(left)* Error comparison for FS-LSSVM and ℓ_0-approximation. *(middle)* Number of support vectors. *(right)* Computational cost.

second level of sparsity impose by the ℓ_0-approximation. The ℓ_0-approximation also has increased computational costs when training the model. My understanding of the goal of ℓ_0-approximation was to increase computational efficiency on large datasets like this by inducing sparsity in the kernel matrix, but this is not the case here [7].

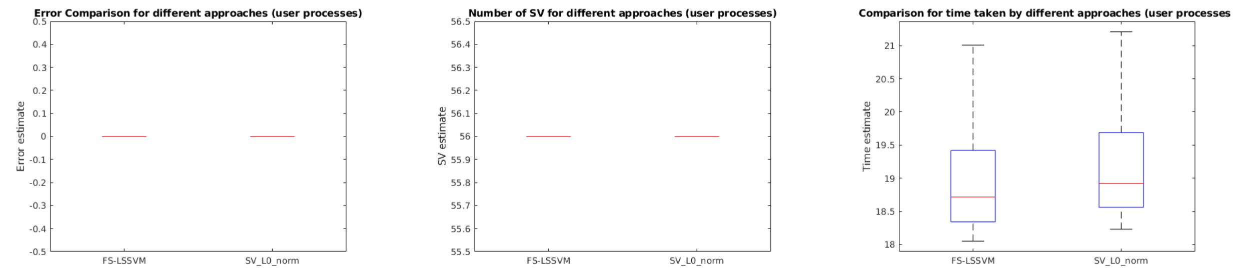


Figure 13: *(left)* Error comparison for FS-LSSVM and ℓ_0-approximation. *(middle)* Number of support vectors. *(right)* Computational cost.

References

[1] Paul Honeine and Cedric Richard. Preimage problem in kernel-based machine learning. *IEEE Signal Processing Magazine*, 28(2):77–88, 2011.

[2] JT-Y Kwok and IW-H Tsang. The pre-image problem in kernel methods. *IEEE transactions on neural networks*, 15(6):1517–1525, 2004.

[3] Bernhard Schölkopf, Sebastian Mika, Alex Smola, Gunnar Rätsch, and Klaus-Robert Müller. Kernel pca pattern reconstruction via approximate pre-images. In *International Conference on Artificial Neural Networks*, pages 147–152. Springer, 1998.

[4] Md Ashad Alam and Kenji Fukumizu. Hyperparameter selection in kernel principal component analysis. *SciPub*, 2014.

[5] Rocco Langone, Raghvendra Mall, Carlos Alzate, and Johan AK Suykens. Kernel spectral clustering and applications. In *Unsupervised Learning Algorithms*, pages 135–161. Springer, 2016.

[6] Kris De Brabanter, Jos De Brabanter, Johan AK Suykens, and Bart De Moor. Optimized fixed-size kernel models for large data sets. *Computational Statistics & Data Analysis*, 54(6):1484–1504, 2010.

[7] Raghvendra Mall and Johan AK Suykens. Very sparse lssvm reductions for large-scale data. *IEEE transactions on neural networks and learning systems*, 26(5):1086–1097, 2015.