KU LEUVEN

FACULTY OF SCIENCE

Exercise Session 3

Support Vector Machines

Daniel Gerardo GIL SANCHEZ
daniel.gilsanchez@student.kuleuven.be
MSc Statistics

Supervisor: Prof. Johan Suykens

Academic year 2018-2019

1 Exercises

1.1 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.***

  The idea behind denoising is to minimize the reconstruction error, which is defined as:

$$\sum_{k=1}^{N} \| x_k - h(z_k) \|_2^2 \tag{1}$$

  where x_k is a specific data point in $\mathbb{R}^n$, z_k is the transformation of that point to another space $\mathbb{R}^{n_s}$, and h is mapping function. This is also known as information bottleneck, where the input space is reduced (enlarged) via linear or kernel PCA and then transformed back to the original input space. Figure 1 shows an illustration of this problem, note that $\tilde{x} = h(z)$.

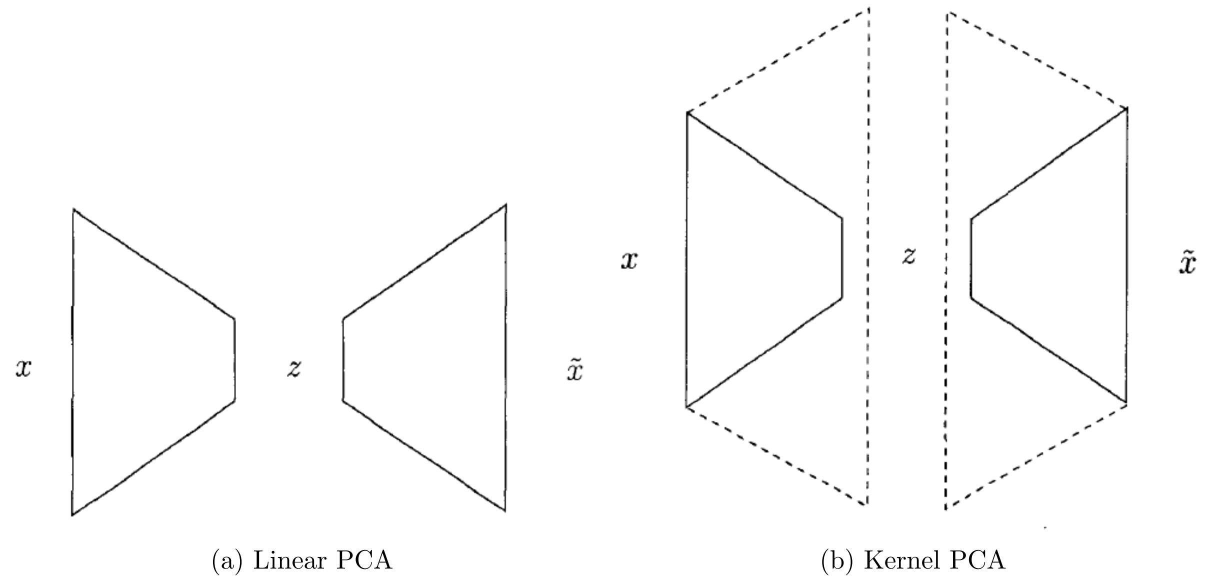


Figure 1: Information bottleneck

  In linear PCA the number of components cannot be greater than the number of input variables; the larger this number, the less denoise is produced. If this number is the same as the number of input variables, then the identity transformation is used, which means that no denoise at all will be produced. On the other hand, in kernel PCA the number of score variables, noted as n_s in equation 1, can be larger that the number of variables in the input space. In this case, one selects as few score variables as possible. Figure 1b shows an illustration of the modified bottleneck.

- ***Compare linear PCA with kernel PCA. What are the main differences? How many principal components can you obtain?***

  Figure 2 shows the solution of the linear PCA retaining only one principal component and the solution of the kernel PCA using six principal components. It is very clear that linear PCA is unable to denoise the shape of the data because all data points are moved to a single line, whereas kernel PCA was able to capture the shape of the data and come up with a good solution. As it was mentioned before, the maximum number of principal components that can be obtained in linear PCA is the number of input variables, which in this case is two, while in kernel PCA this number can be larger than the number of input variables. Therefore, linear PCA will not be the right choice if the shape of the data is nonlinear in nature.

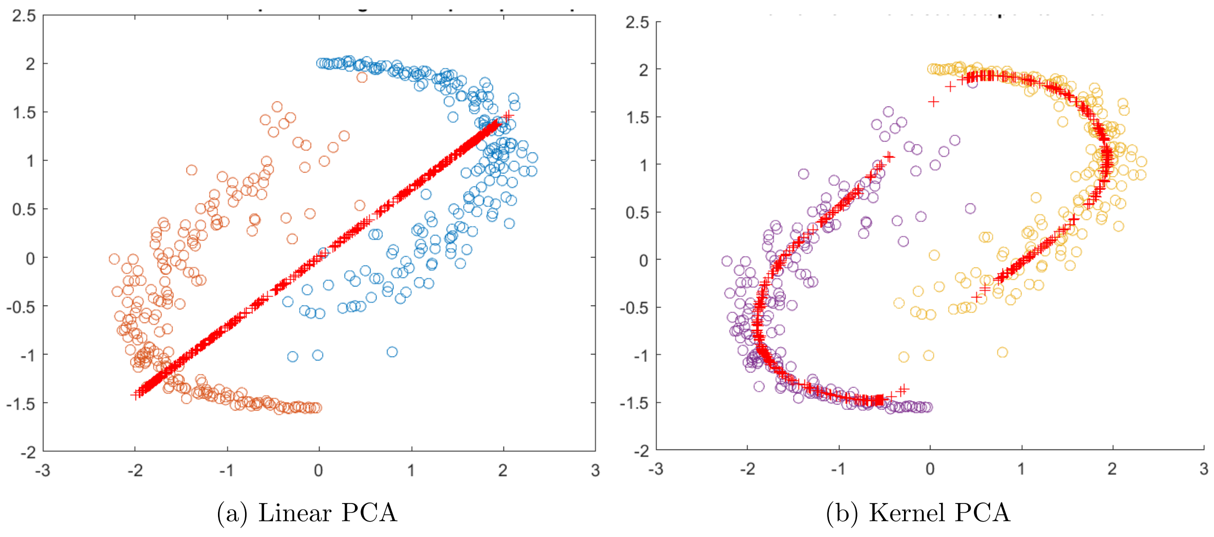


Figure 2: Comparison of linear and kernel PCA

- ***For the dataset at hand, propose a technique to tune the number of components, the hyperparameter and the kernel parameters.***

  An easy way to tune these parameters is by doing an exhaustive grid-search by considering a large number of pairs of parameters, which in this case are the number of components and `sig2` associated to an RBF kernel. Then, take the combination of parameters that minimizes the most the reconstruction error. To build the grid of parameters, Figure 3 illustrates how the reconstruction changes when the number of components is manipulated while keeping constant `sig2`. There, it is easy to see that when the number of components increases, the denoising gets worse and worse (see plot 3d). These plots suggest that a good solution can be found by retaining between 6 to 10 components, hence the grid should consider these values.

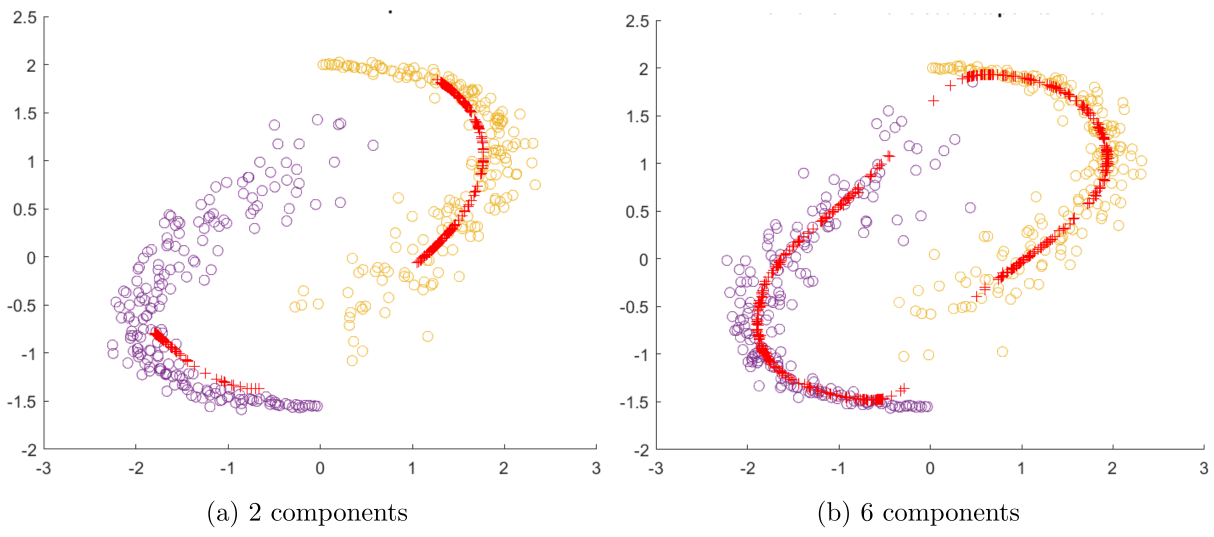
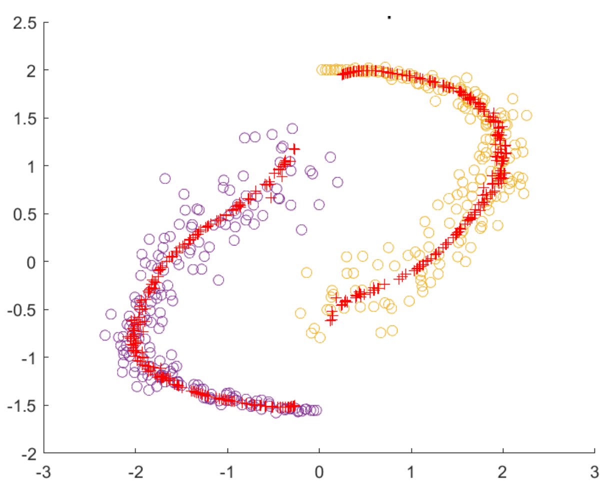


Figure 3: Comparison of solutions considering different number of components


(c) 10 components

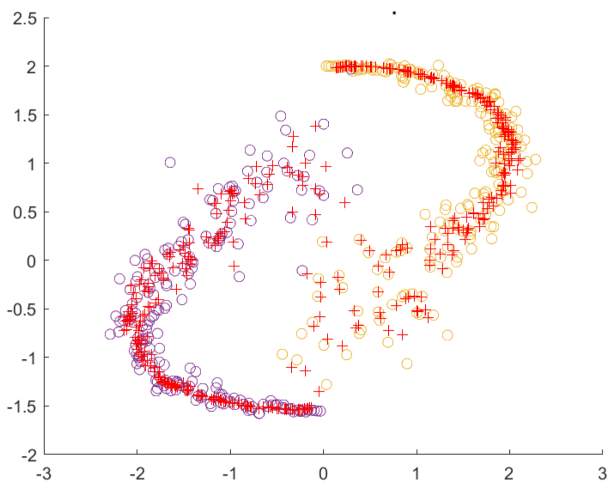

(d) 15 components

Figure 3: Comparison of solutions considering different number of components (cont.)

Another way to get a reasonable number of principal components in the grid for the optimal solution, is by analyzing the scree-plot, i.e., the plot with the eigenvalues obtained. Figure 4 shows the corresponding eigenvalues when `sig2` is fixed at 0.4 as in the plots above. The first big jump is between the first and the second component, the second jump is between the fifth and the sixth component. After this component, the eigenvalues steadily decrease up to the 11th component, where the difference between components becomes smaller and smaller. As a consequence, the grid should consider from 5 to 11 principal components in the search for an optimal solution.

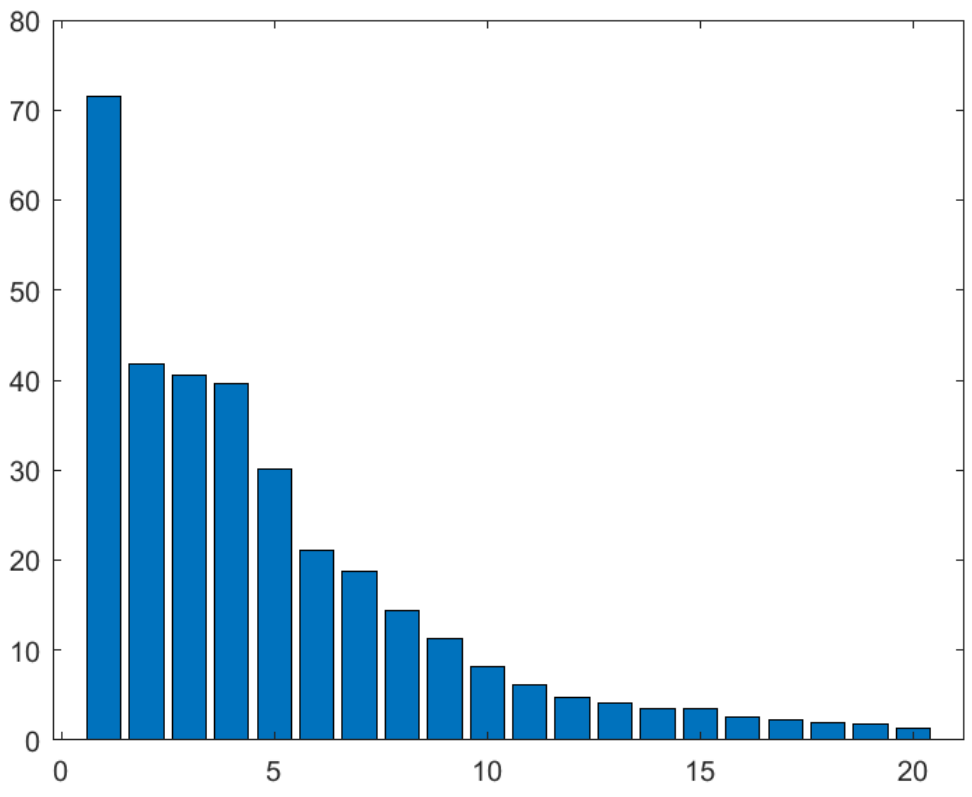


Figure 4: Scree plot

In regard to the `sig2`, it is well know that the final solution highly depends on this parameter. Therefore, the grid should consider different values for `sig2`: from very, very small values to large values. The more exhaustive the search, the better.

1.2 Spectral clustering

- ***Explain briefly how spectral clustering works.***

  Basically spectral clustering can be defined in three steps as follows:

  1. Create a similarity graph between all data points in the dataset.
  2. Compute the first k eigenvectors of its Laplacian matrix to define a feature vector for each object.
  3. Usually the largest eigenvector does not contain clustering information. For binary clustering, the solution is the second largest eigenvector.
  4. (Optional) Instead of step 3, run k-means on these features (step 2) to separate objects into k classes.

  In this case, to create the similarity graph, a kernel matrix is used because a kernel function can be seen a measure of similarity between data points. The Laplacian matrix can be computed as $L = D - W$, with D the degree matrix and W the similarity matrix. The degree matrix is computed as $D = diag(d_1, \ldots, d_N)$, where $d_i = \sum_{j=1}^{N} \Omega_{ij}$ and Ω is the kernel matrix.

  In binary classification, a data point is assigned to a class by the sign of its corresponding value in the second largest eigenvector.

- ***What are the differences between spectral clustering and classification?***

  The main difference is that spectral clustering is an unsupervised algorithm, whereas classification is supervised because there are labels, or true values, that feeds the algorithm to get a better solution.

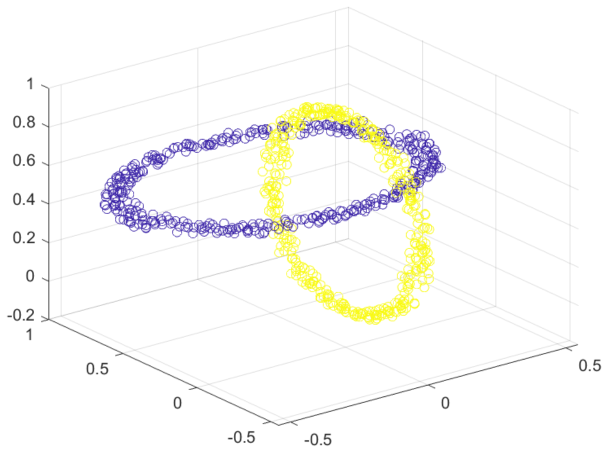
  In this sense, an SVM representation of spectral clustering can be seen as a weighted version of kernel PCA, where the cost function is defined as:

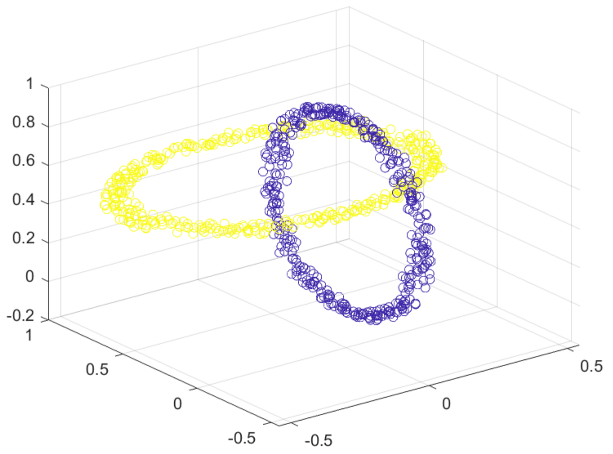
  $$\min_{w,b,e} -\frac{1}{2}w'w + \gamma\frac{1}{2}\sum_{i=1}^{N} v_i e_i^2 \qquad (2)$$

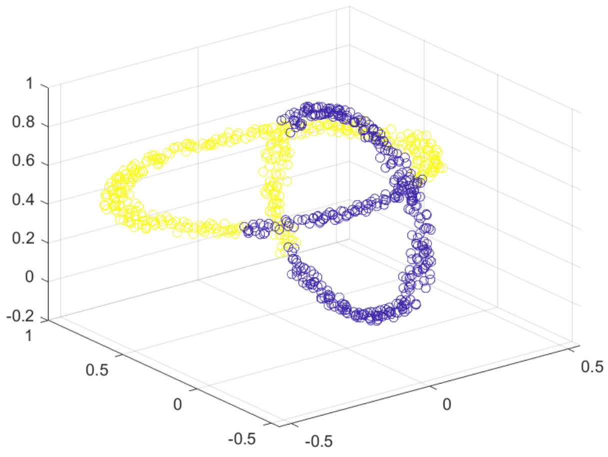
  subject to $e_i = w'\phi(x_i) + b$. Note that v_i are the weights introduced in the cost function, and that e_i is not the difference between the true label and the estimated, but only the estimated value because there are not true values in this scenario.

- ***Edit the script and try different values of $sig2$ (e.g., 0.001, 0.005, 0.01, 0.02). What is the influence of the $sig2$ parameter on the clustering results?***

  Figure 5 shows how the clustering results change when the parameter `sig2` changes. In this case, four different values were considered, namely 0.001, 0.01, 0.05 and 0.1. It seems that this parameter is very sensitive for the final solution because small values tend to give good results, but a small change makes the resulting cluster to be completely wrong, see the difference between plot 5b and 5c. After this value, 0.05, it seems that the solution is wrong as well, as it is presented in plot 5d.


(a) `sig2` $= 0.001$


(b) `sig2` $= 0.01$


(c) `sig2` $= 0.05$

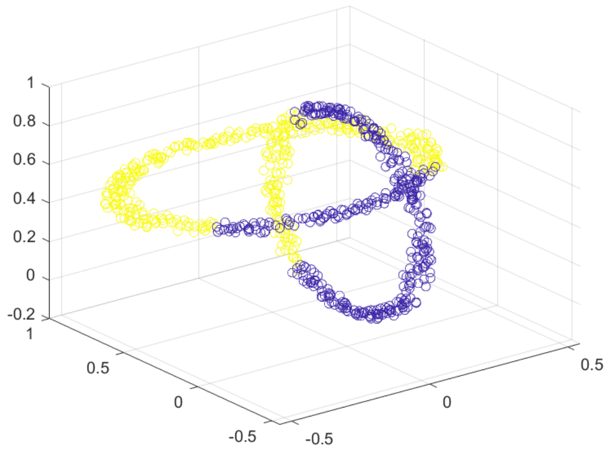

(d) `sig2` $= 0.1$

Figure 5: Comparison of solutions considering different `sig2` values

1.3 Fixed-size LS-SVM

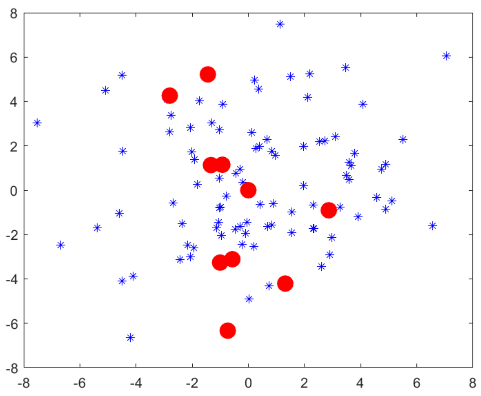
- ***In which setting would one be interested in solving a model in the primal? In which cases is a solution in the dual more advantageous?***

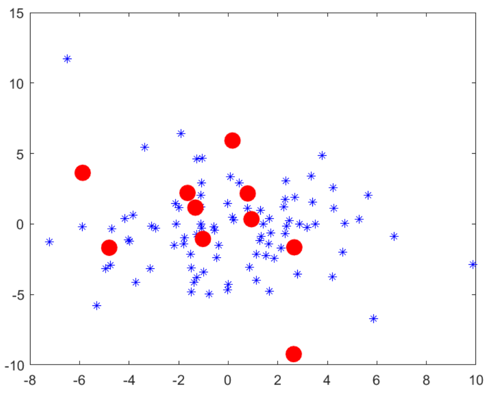
  The primal representation is usually used when the number of data points in the training set is large. This happens because in the dual representation the kernel matrix is of dimension $N \times N$, with N observations. If this number is large, the computations are unfeasible. On the other hand, if the dimensionality of the input space is large, the dual representation is usually used. This happens for the same reason that the kernel matrix does not depend on the number of input features.

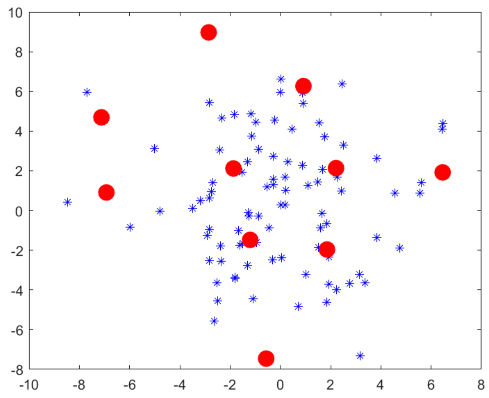
  In addition to that, the selection of the primal or dual representation can be influenced by the kernel function used. If the kernel considers an infinite feature space, like an RBF, then only the dual representation can be used.

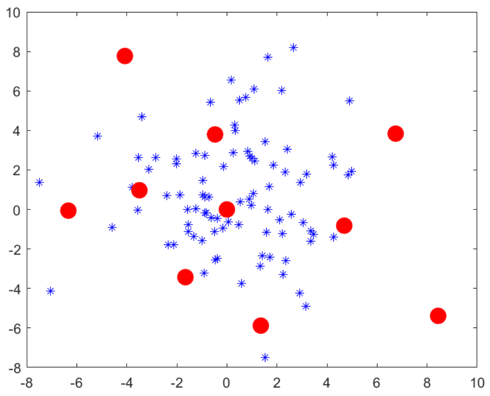
- ***What is the effect of the chosen kernel parameter $sig2$ on the resulting fixed-size subset of data points? Can you intuitively describe to what subset the algorithm converges?***

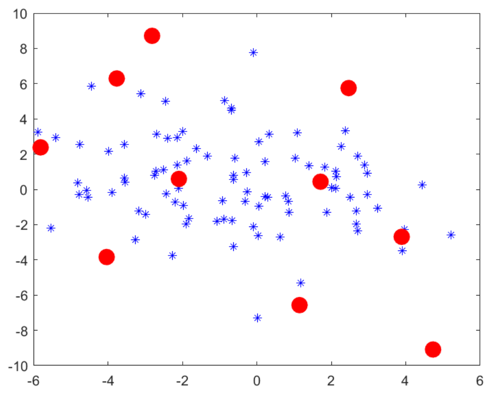
  Figure 6 shows the resulting fixed-size subset of data points when `sig2` is manipulated. Overall, the algorithm ends up choosing those data points that are located in the boundary of the data cloud. The purpose of `sig2` is to determine how close these selected points are to each other. In the case when `sig2` is really small (plots 6a and 6b), the resulting subset is located in the middle of the cloud. But when `sig2` increases, the selected data points are located in the boundary.


(a) `sig2` $= 0.0001$


(b) `sig2` $= 0.001$


(c) `sig2` $= 0.1$


(d) `sig2` $= 1$


(e) `sig2` $= 10$

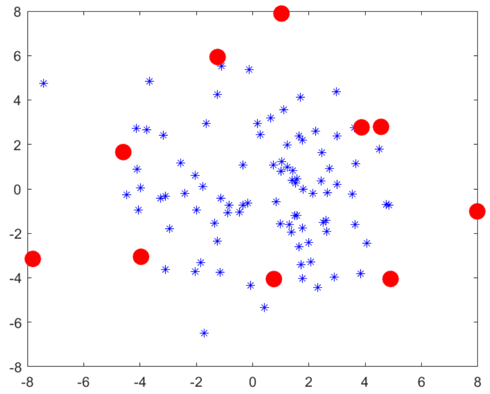
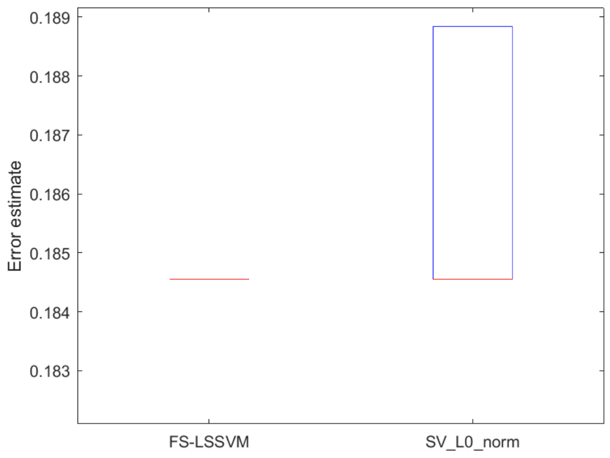

(f) `sig2` $= 100$

Figure 6: Fixed-size subsets with different `sig2` values

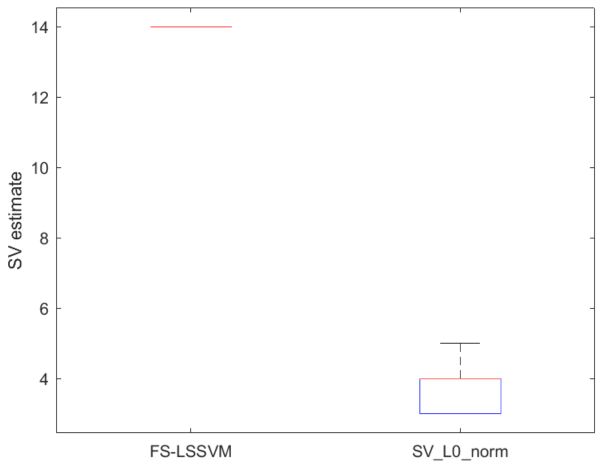
- ***Run*** *fslssvm_script.m*. ***Compare the results of fixed-size LS-SVM to ℓ_0-approximation in terms of test errors, number of support vectors and computational time.***

  Figure 7 shows the respective comparison between Fixed-size LS-SVM and ℓ_0-approximation. In terms of test errors, it can be seen that the results of Fixed-size LS-SVM are constant, whereas the results of the other methodology present more variability and are slightly worse.

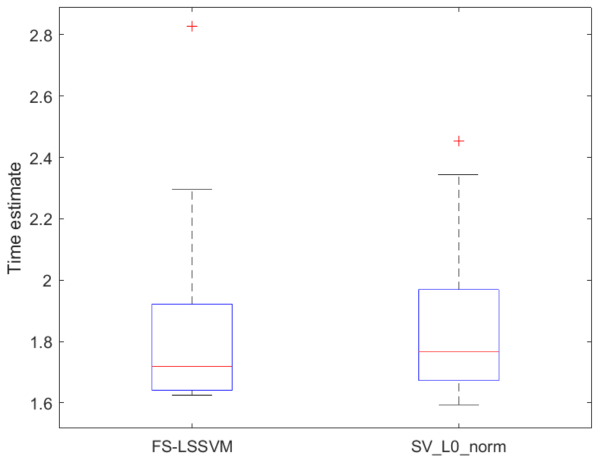
In terms of the number of support vectors, the difference is of course expected due to the fact that ℓ_0-approximation is used to get a sparser solution. In LS-SVM the number of support vectors was 14 and in ℓ_0-approximation was between 3 and 5. In terms of computing time, there is a difference between both methodologies, although it does not seem to be significant. The median computational time is slightly longer for the ℓ_0-approximation. In general, there is a trade-off between the number of support vectors and the test errors, because there is an inverse relationship between both. The sparser the solution, the lower the quality in a test set.



(a) Test errors



(b) Number of support vectors



(c) Computational time

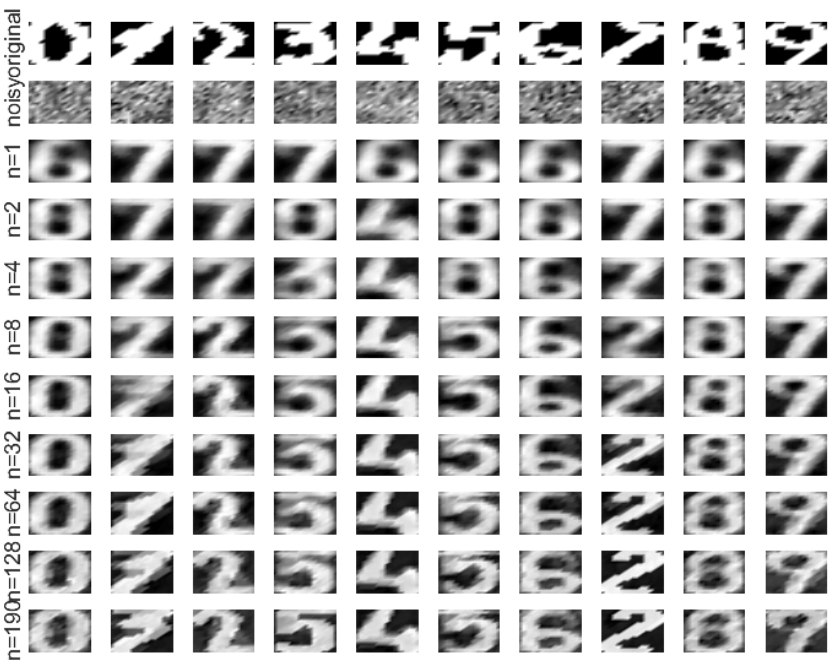
Figure 7: Comparison between Fixed-size LS-SVM and ℓ_0-approximation

2 Homework problems

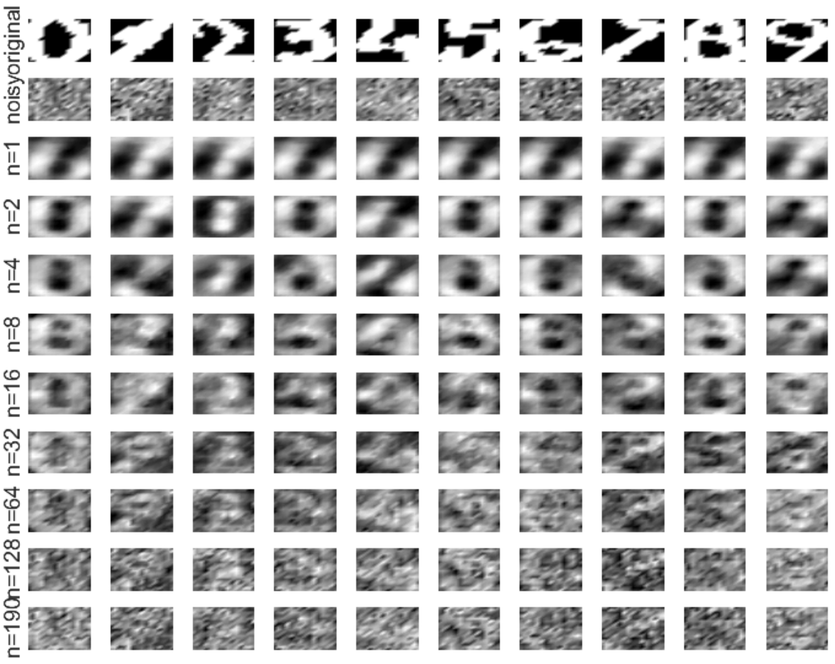
2.1 Kernel principal component analysis

- ***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).***

  Figure 8 shows the difference between kernel and linear PCA in a denoising problem of handwritten numerals. The first two rows show the original image and the noise added. From the third row onwards, the denoising solution is presented by considering different number of principal components. It is very clear that kernel PCA outperforms linear PCA in denoising with high quality the numerals presented because the solution for the latter is a blurred image, regardless the number of components retained. The best solution in kernel PCA is obtained by retaining between 16 to 32 principal components, whereas in linear PCA it is not possible at all to get a decent representation of the original numerals.



(a) Kernel PCA



(b) Linear PCA

Figure 8: Comparison between kernel and linear PCA

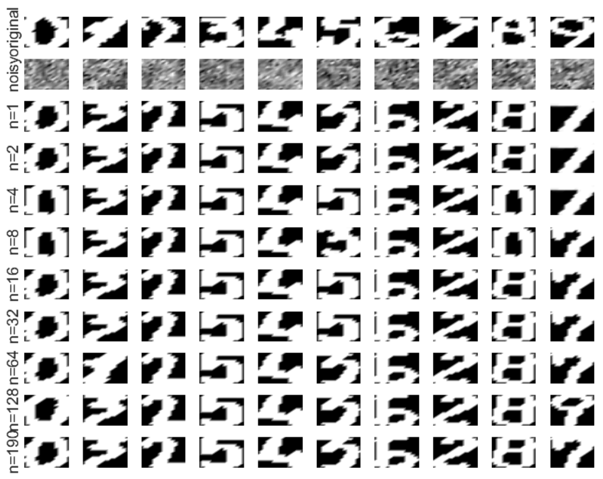
- ***What happens when the `sig2` parameter is much bigger than the suggested estimate? What if the parameter value is much smaller? In order to investigate this, change the***

***sigma_factor* parameter for equispaced values in logarithmic scale.**

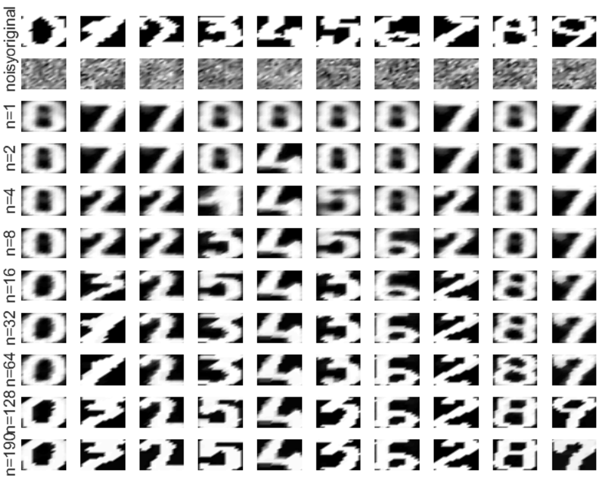
To investigate the effect that `sig2` has on the final solution, six different values for `sigma_factor` on a logarithmic scale were tested. Figure 9 shows how the results change when this parameter increases or decreases. In the first two plots, when `sig2` is really small, the denoising of the images seem to be perfect at a first glance. However, there are problems with some specific numbers: numeral 1 is mixed with numeral 3, numeral 3 is mixed with numeral 5, numeral 8 in some solutions is mixed with numeral zero. Therefore, it can be concluded that when `sig2` is too small, the solution is not good enough.

On the other hand, when `sig2` is too large (plots 9e and 9f), the denoising process did not remove the noise enough, up to a point that it is not possible to identify which number is represented. Thus, it can also be concluded that large values of `sig2` lead to bad results.

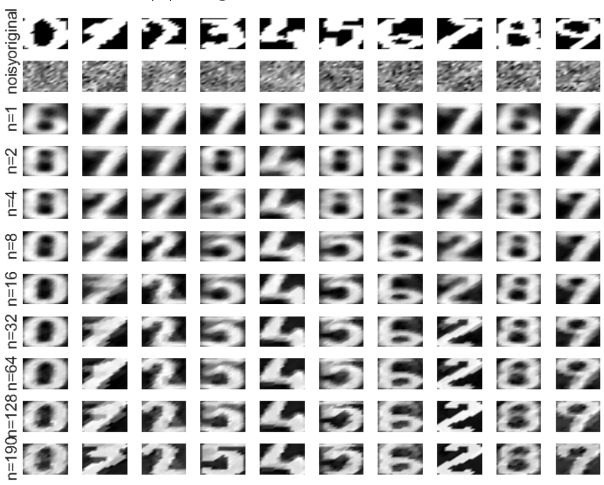
Finally, it seems that when `sig2` is not large, but also not too small, the solution is good enough. Focusing in plot 9c, it is easy to see that the noise is removed and that the original numbers appear correctly. As a consequence, the best way to choose `sig2` is by tuning it with any algorithm and select the one that minimizes the reconstruction error.



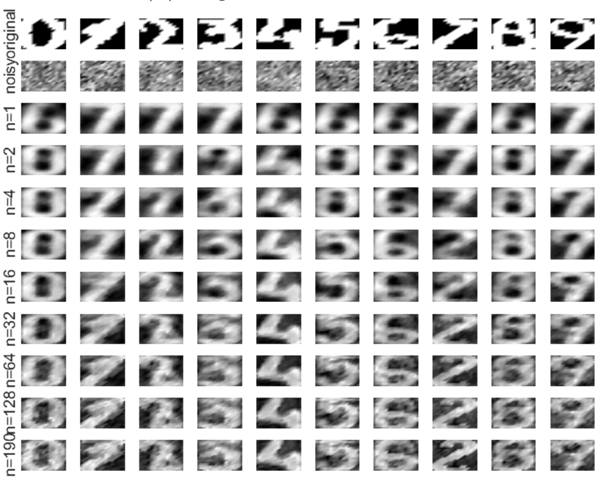
(a) `sigma_factor` = 0.1000



(b) `sigma_factor` = 0.2512

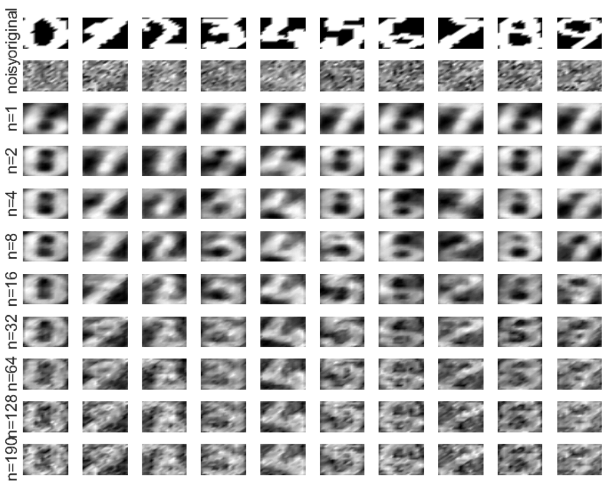


(c) `sigma_factor` = 0.6310

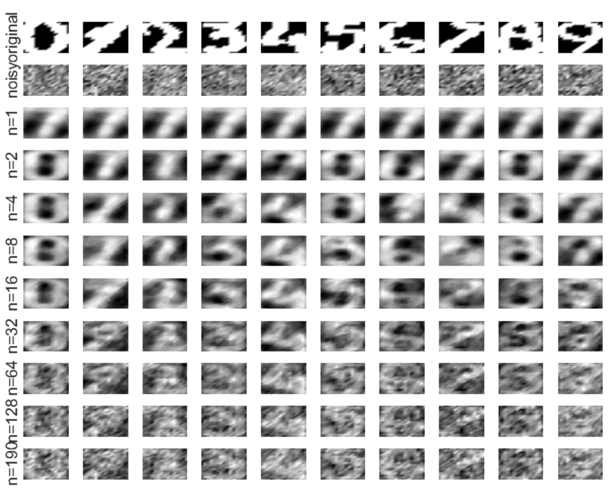


(d) `sigma_factor` = 1.5849

Figure 9: Comparison of solutions when `sig2` is manipulated



(e) `sigma_factor` = 3.9811



(f) `sigma_factor` = 10

Figure 9: Comparison of solutions when `sig2` is manipulated (cont.)

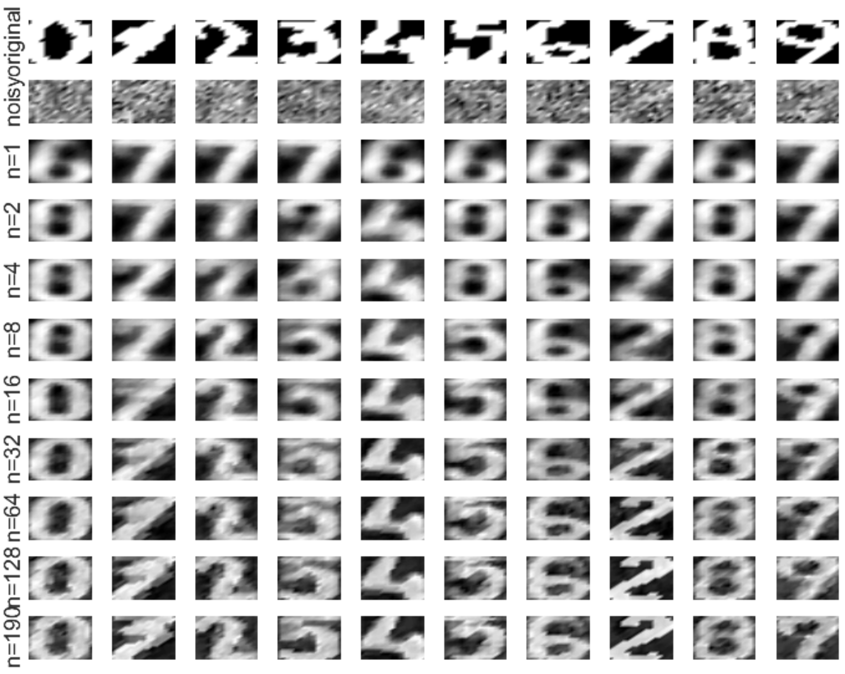
- ***Investigate the reconstruction error on training (`Xtest`) and validation sets (`Xtest1` and `Xtest2`), as a function of the kernel PCA denoising parameters. Select parameter values such that the error on the validation sets is minimal. Can you observe any improvements in denoising using these optimized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

  To get an optimal solution, several steps were taken: First, define `sig2` as the mean of the variances of each dimension times the dimension (number of features) of the training data. Second, define `sigma_factor` as a sequence of 10 values equally spaced on a logarithmic scale. Third, multiply `sig2` with `sigma_factor`, obtain the kernel PCA solution using this updated `sig2` parameter and then compute the reconstruction error on the training, test1 and test2 dataset. Table 1 shows these results, where it can be seen that the reconstruction error is really small in training and test2 datasets. By choosing the `sig2` value that minimizes the most the reconstruction, it can be concluded that 153.89 is the optimal value for training and test2 datasets, whereas 69.40 is the optimal value for test1 dataset.

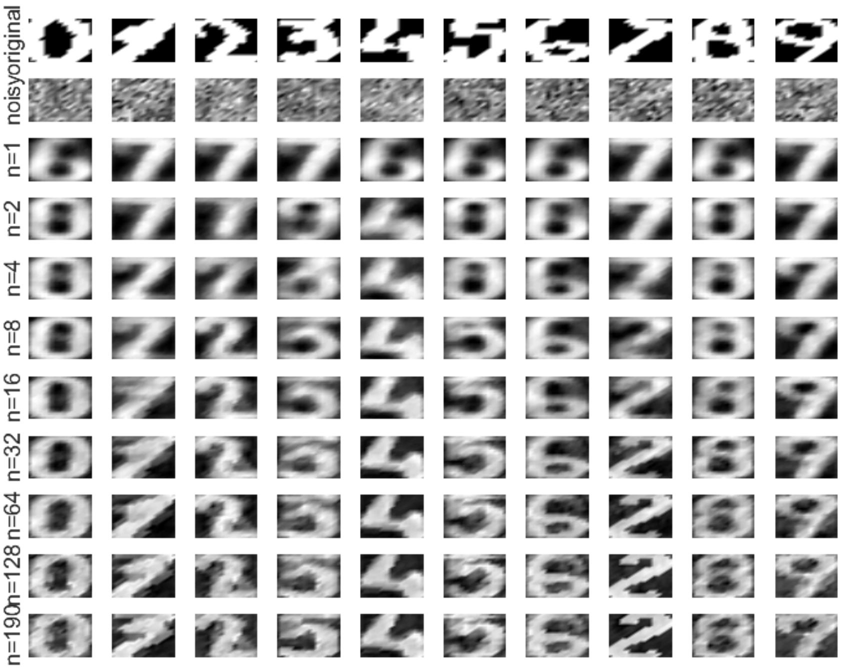
`sigma_factor`	`sig2`	`Training`	`Test1`	`Test2`
0.5	25.65	2.6×10^{-13}	5.64	2.0×10^{-14}
0.61	31.3	1.6×10^{-13}	4.75	-3.9×10^{-15}
0.74	38.19	1.9×10^{-13}	3.93	-1.1×10^{-15}
0.91	46.61	2.1×10^{-13}	3.18	1.6×10^{-14}
1.11	56.87	-3.6×10^{-14}	2.54	-1.6×10^{-15}
1.35	69.4	-1.7×10^{-13}	2.01	-9.8×10^{-15}
1.65	84.69	3.4×10^{-15}	1.57	-1.2×10^{-15}
2.01	103.35	-2.8×10^{-14}	1.21	-2.7×10^{-15}
2.46	126.11	-7.7×10^{-14}	0.93	-6.0×10^{-15}
3	153.89	-1.4×10^{-13}	0.71	-5.1×10^{-15}

Table 1: Reconstruction error using different `sig2` values

Figure 10 shows the solution for each of the optimized parameters. At first glance, it seems that both solutions are the same. Thus, it can be concluded that even though the `sig2` values are different, the overall solution is quite similar. Now, with respect to the comparison with the solutions of the previous item, it seems that there is an improvement in some cases, but is not huge. It is important to mention that this happened because in this item `sigma_factor` was specified to be close to the good solutions obtained in the previous item (see first column in Table 1).



(a) `sig2` $= 153.89$



(b) `sig2` $= 69.40$

Figure 10: Comparison between optimized `sig2` values

2.2 Fixed-size LS-SVM

2.2.1 Shuttle (statlog)

- ***Explore and visualize (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 is composed of 58000 observations and ten variables. There are nine numerical input variables and one categorical target variable with 7 classes. Approximately 80% of the data belongs to class 1. Therefore the default accuracy is about 80%. Unfortunately, the description of each variable was not available so it is not possible to give interpretations in the context of the dataset. Figure 11 shows the distribution of each variable, colored by the corresponding target value, and the scatter plot of all pairs of variables. With respect to the distribution of each variable, it seems that those data points that belong to class 7, colored in black present extreme values in variables 1, 2, 7 and 8. Classes 3 and 4, red and light green respectively, have a similar shape but different frequencies in almost all variables. Regarding the scatter plots, class 5, colored in pink, is always apart from the cloud of the data points that belong to the remain classes.

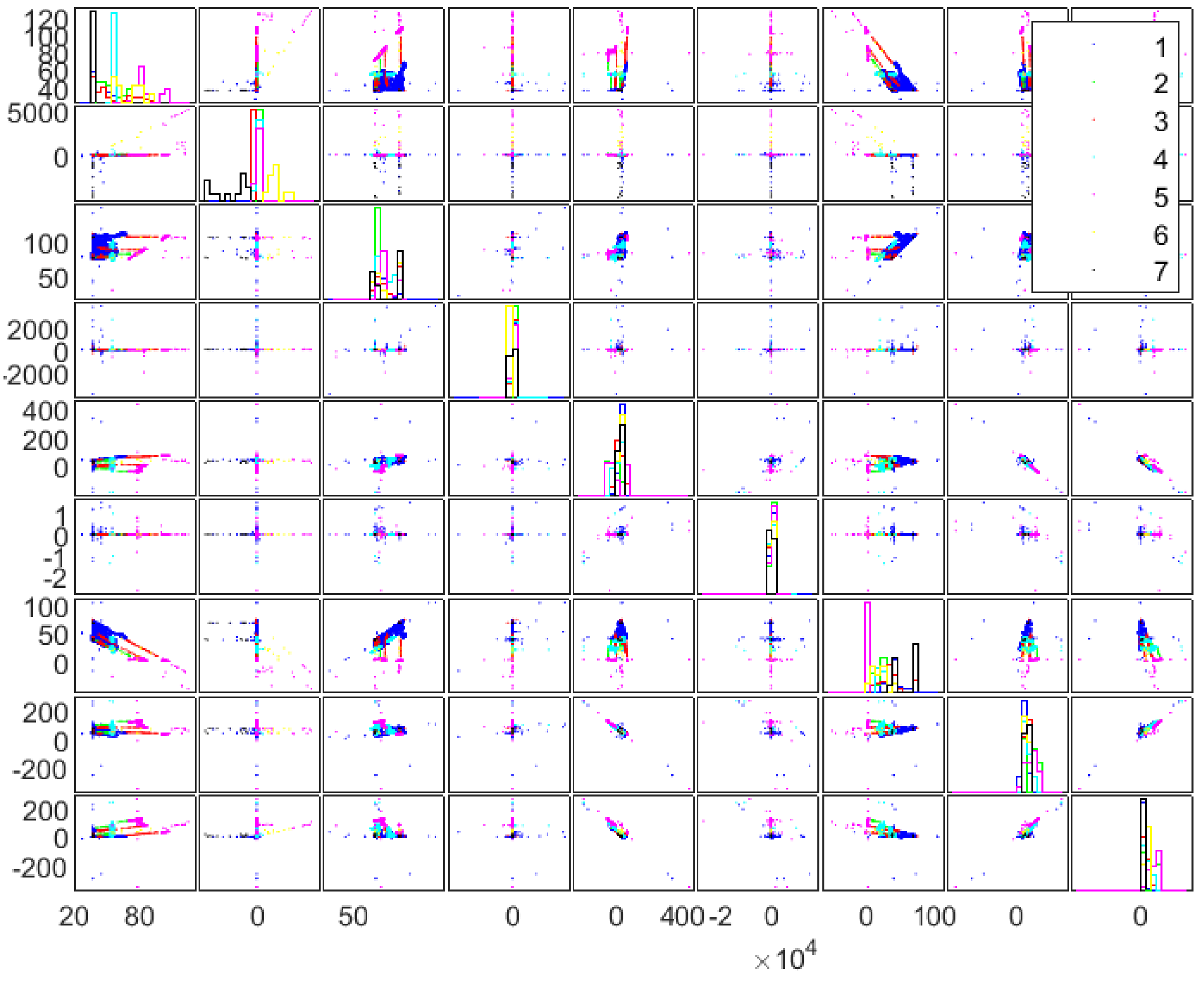
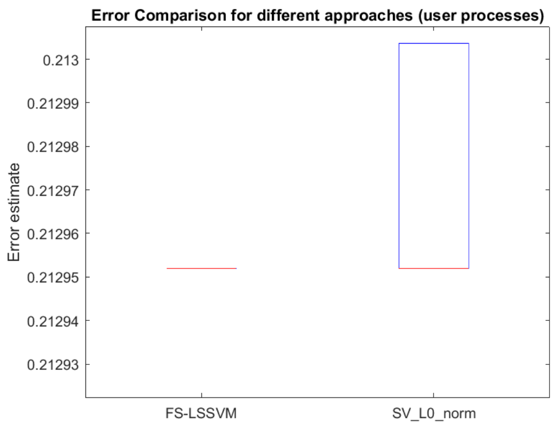


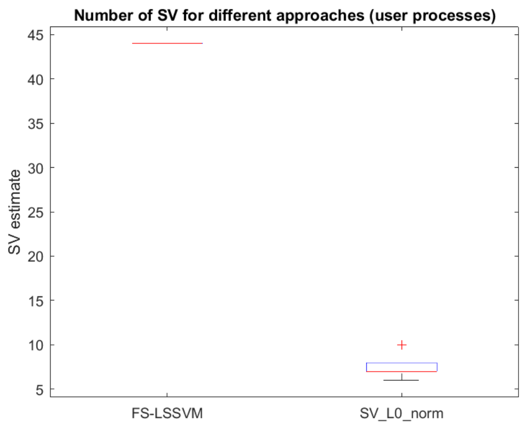
Figure 11: Distribution and scatter plots of each variable

- ***Visualize and explain the obtained results.***

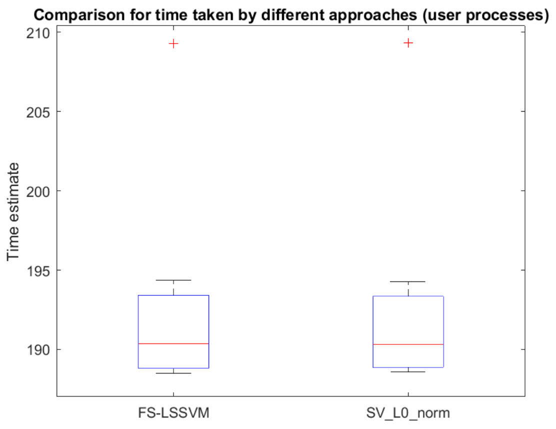
Figure 12 shows the comparison between Fixed-size and ℓ_0-approximation in terms of test errors, number of support vectors and computational time. The difference between these results and the results obtained in the third question of item 1.3 is that here the whole dataset was used to make the comparison, instead of the first 700 observations. Although the dataset is different, the conclusions remain the same: in terms of test errors, ℓ_0-approximation produces slightly worse results than Fixed-size. The number of support vectors is way smaller for the second method because it produces sparse solutions. In terms of computing time, there is a difference in the median, but it is not significant at all. In general, the trade-off between the number of support vectors and the test errors also holds in this dataset. The sparser the solution, the lower the quality in a test set.



(a) Test errors



(b) Number of support vectors



(c) Computational time

Figure 12: Comparison between Fixed-size LS-SVM and ℓ_0-approximation

2.2.2 California

- ***Explore and visualize (part of) the dataset. How many datapoints? How many and meaning of attributes?***

  The dataset is composed of 20640 observations and nine numerical variables. The response variable is the logarithm of the median house value and the input variables are the median income, housing median age, total number of rooms, total number of bedrooms, population, households, latitude and longitude. Figure 13 shows the distribution of each variable and the scatter plot of all pairs of variables, including the response in the last column. Several things can be highlighted, unfortunately the dataset was not labeled so it is not possible to give interpretations with the appropriate context: first, there is a strong correlation between the each pair of variables 4, 5 and 6. Second, there is a strong negative correlation between the first and second variables. Third, the distribution of the response (last column) has a peak in the higher values, which can be explained by a neighborhood with really expensive houses. Finally, the distribution of variables 4 to 7 is skewed to the right.

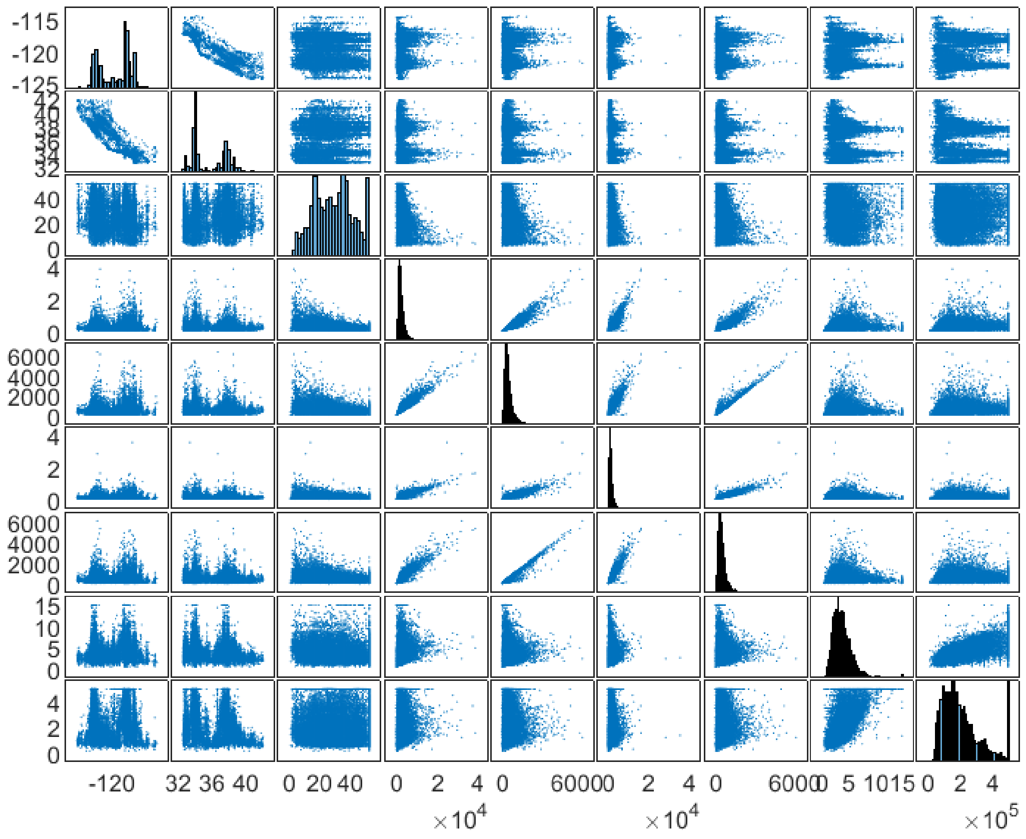
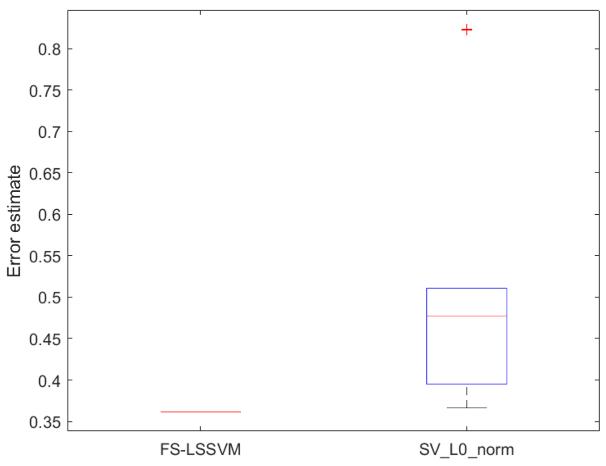


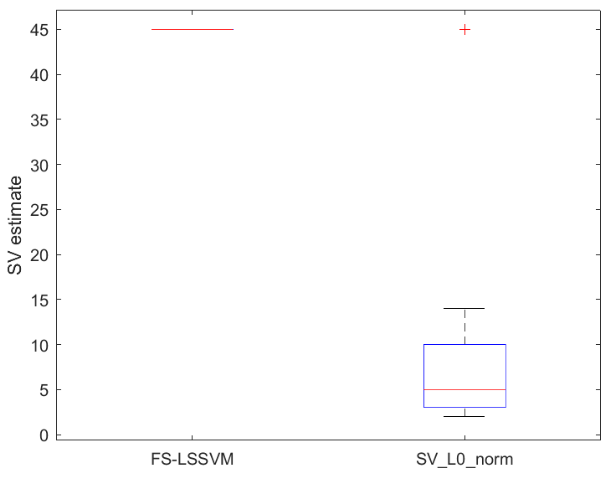
Figure 13: Distribution and scatter plots of each variable

- ***Visualize and explain the obtained results.***

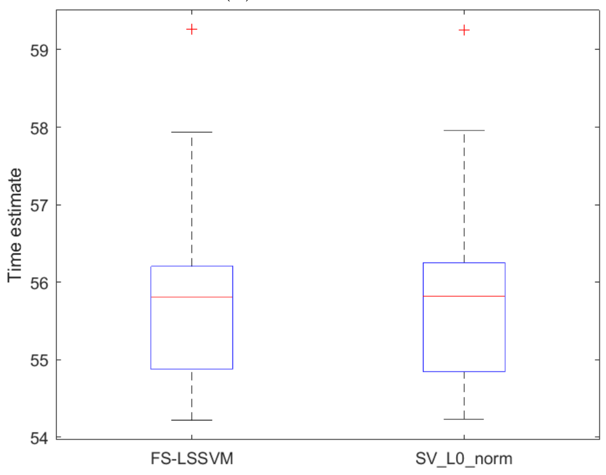
Figure 14 shows the comparison between Fixed-size and ℓ_0-approximation in terms of test errors, number of support vectors and computational time. As in the previous item, the overall conclusion is that ℓ_0-approximation, in comparison with Fixed-size LS-SVM, spends approximately the same computation time to produce sparser solutions with the trade-off between quality of the solution, measured as the error, and complexity, measured as the number of support vectors.



(a) Test errors



(b) Number of support vectors



(c) Computational time

Figure 14: Comparison between Fixed-size LS-SVM and ℓ_0-approximation