Support Vector Machines

Final report on exercise sessions

Marco Salmistraro
marco.salmistraro@student.kuleuven.be
Student number 0874279

Course in Support Vector Machines
Advanced Master's in AI - Academical year 2021/2022
KU Leuven

1. Two Gaussians

In the first exercise two clusters of points are created in a two-dimensional plane. These are defined as Gaussian-type random distributions over the feature space. The first group consists of a series of 50 bi-dimensional points distributed according to a Gaussian curve and shifted above by one unit. Conversely, the other one is constituted by 51 points following the same random assignment, but shifted downwards by one unit. This is an instance of a binary classification problem, where only two classes are available (shown in red and blue in Figure 1). A basic approach could consist in drawing a line over the plane; within such a confined framework, the separation can be considered to be optimal as long as the instances are correctly separated over the entirety of the dataset. However, since the two classes are overlapping, it is impossible to define a boundary that perfectly splits the space into the target classes without any misclassified instances. One could relax the problem by finding a line yielding perfect classification for at least one of the classes; anyway, this would not be an optimal approach. A further alternative would be to define a non-linear decision boundary, which can be done by employing the Mercer theorem; this would allow to map the two-dimensional vectors onto a higher-dimensional space on which a separating hyperplane could be defined. This separation plane would then be projected back onto the initial space as to trace the non-linear decision boundary.

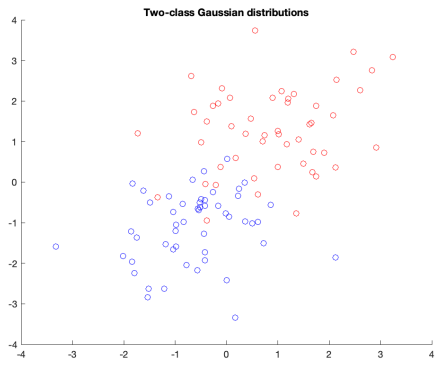
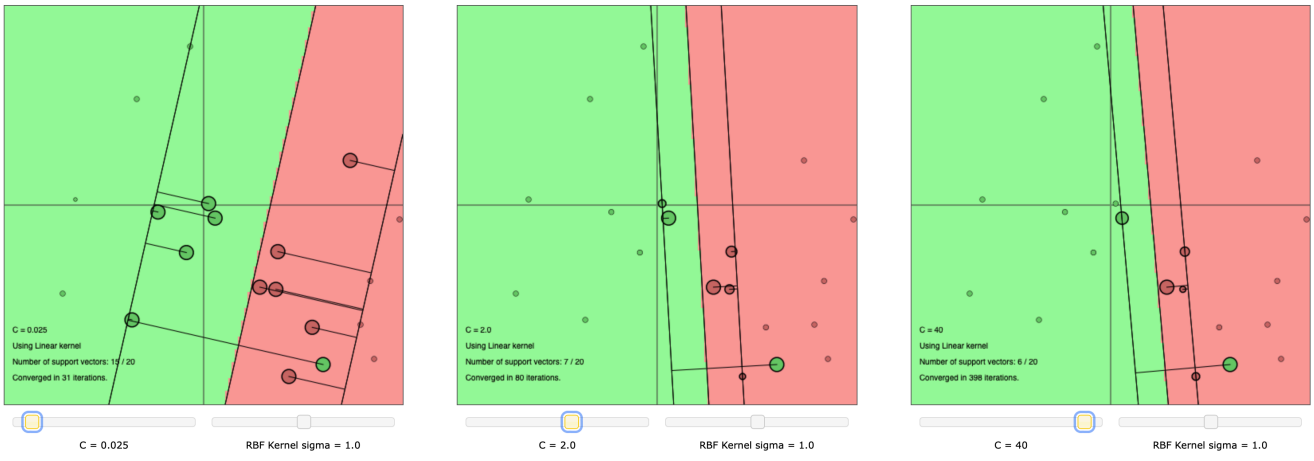


Figure 1. Classification of a two-class variable on the 2D plane

2. Support Vector Machine (SVM) Classifier

The exercise employs the demonstration tool available at `https://cs.stanford.edu/people/karpathy/svmjs/demo/`. This allows for comparison between different models for SVM kernel-based classification. A set of input points belonging to two different classes is introduced onto the 2D plane; first of all, a linear kernel, a basic method of classifying input instances, is assessed. The regularization parameter controls the fraction of points that are used as support vectors for the determination of the separating hyperplane; different values are tested. As it can be seen in Figure 2, imposing a value of 40 results in only 6 of the 20 available points being employed. Furthermore, it must be noted how increasing regularization puts more strain onto the algorithm: in fact, the highest



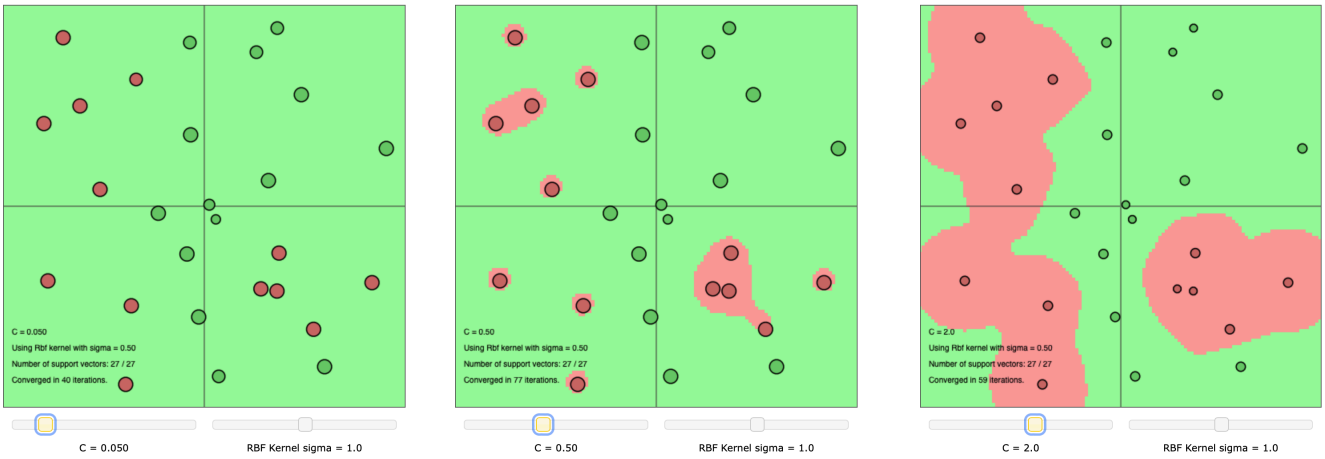
(a) Linear kernel, `c=0.0025`

(b) Linear kernel, `c=2`

(c) Linear kernel, `c=40`

Figure 2. SVM with linear kernel, influence of the `c` parameter

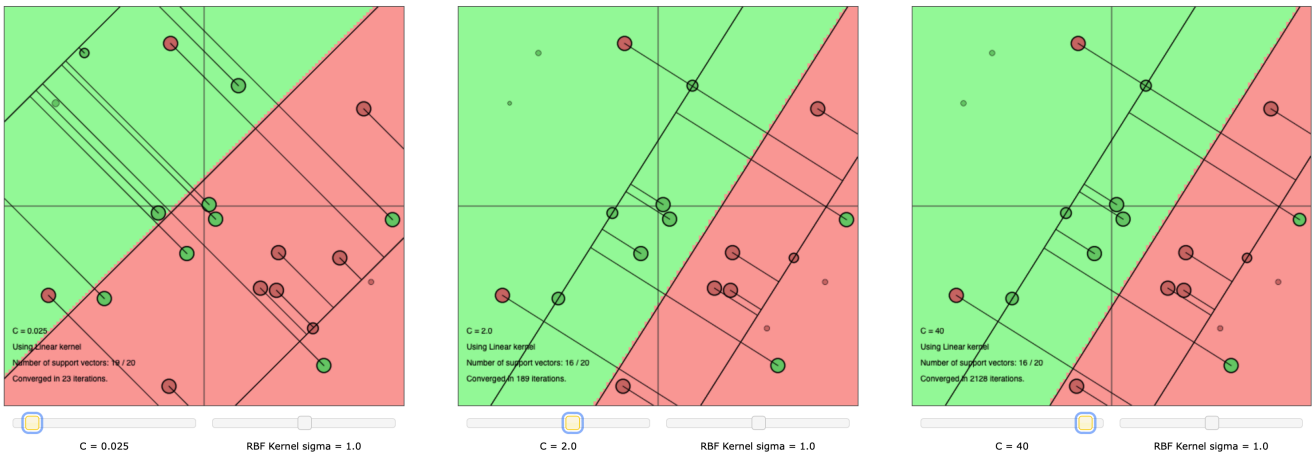
value tested (c) requires 398 iterations for full convergence, as opposed to the 31 iterations required in the first trial (a). Next, a Radial Basis Function (RBF) kernel is tested. As more points are added onto the right side of the hyperplane (green units being inserted onto the green region and red ones being added to the other one) one can observe that the decision boundary becomes more defined, without going through drastic deformations. It is worth noticing how the effect differs when points are inserted onto critical regions, thus becoming support vectors for the classifier; these correspond to units with non-zero Lagrangian multipliers in the dual representation of the target function. In this case, the maximum-margin decision boundary can undergo significant transformations in order to



(a) RBF kernel, `c=0.05`, `sigma=1` (b) RBF kernel, `c=0.5`, `sigma=1` (c) RBF kernel, `c=2`, `sigma=1`

Figure 3. SVM with RBF kernel, influence of the `c` parameter

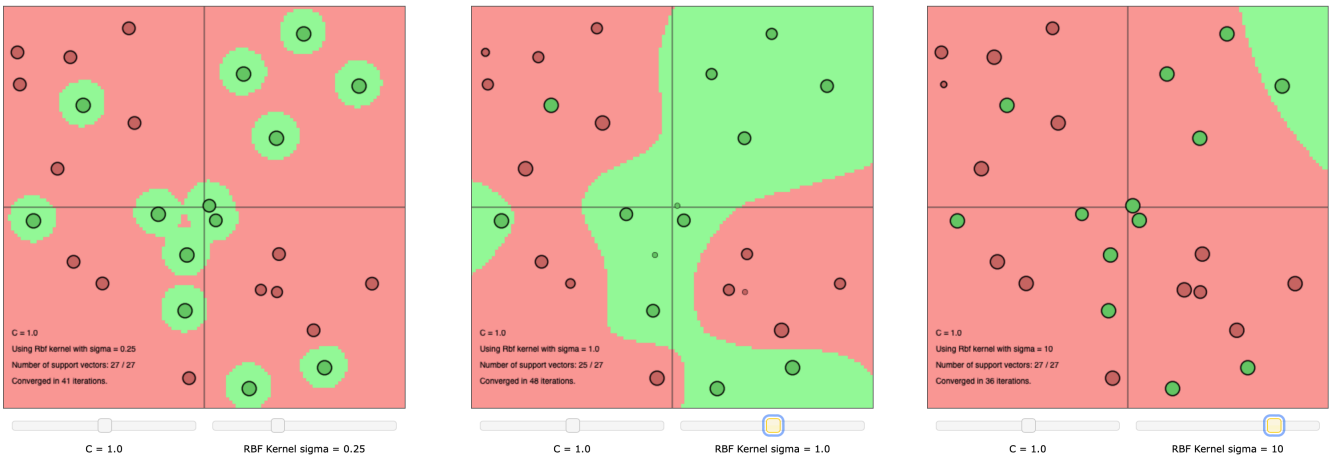
accommodate new training data. Finally, in case points are inserted onto the opposite side of the hyperplane the modifications observed on the boundary can be significant, as the newly added vectors can drastically alter the interpretation that the model operates on the set of data. One of the most evident cases is that of a vector that alters a previously continuously assigned region: the model then needs to re-evaluate the shape of the hyperplane on that area in order to take into account a newly added point. Indeed, these phenomena appear to be analogous to those observed when applying a linear kernel. Different values of the regularization and bandwidth parameters are then explored; as seen previously, growing regularization causes the model to steer away from the possibility of overfitting, while conversely sacrificing its flexibility by employing less points as support vectors; it can be observed that at higher values of the constant the model hardly responds to the



(a) Linear kernel, `c=0.025` (b) Linear kernel, `c=2` (c) Linear kernel, `c=40`

Figure 4. SVM with linear kernel, influence of the `c` parameter on non-separable classes

insertion of new datapoints onto the plane. The underlying rationale is that of controlling the complexity of the modeled function by imposing a bounded norm in some function space; in fact, complex functions tend to have higher norms. Further trials are run by selecting strongly non-separable input features and using a linear kernel. As it can be seen on Figure 4, not all points are



(a) RBF kernel, `c=1`, `sigma=0.25`

(b) RBF kernel, `c=1`, `sigma=1`

(c) RBF kernel, `c=1`, `sigma=10`

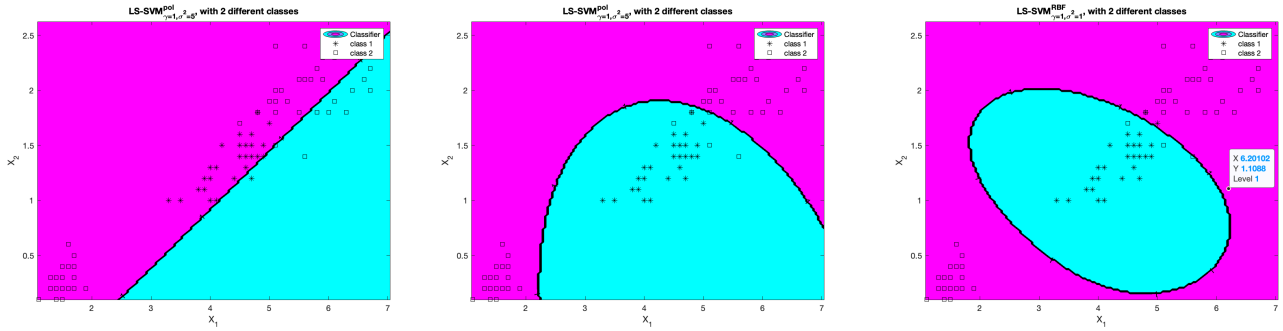
Figure 5. SVM with RBF kernel, influence of the `sig2` parameter on non-separable classes

employed as support vectors, but rather about half of them; this is a major difference with RBF-type kernels, which tend to consider more vectors for definition of the separation hyperplane. In general, switching to a RBF kernel shows decreased values for the misclassification error: the power of this type of functions resides in the adaptability to newly entered points, irrespective of the side. In case the regularization is kept small, the whole region is assigned one unique label (the predominant one); as the value is increased, decision regions start to form: the non-linearity of the kernel can be appreciated in the way the regions evolve (Figure 3). Moreover, the size of the points correspond to their weight in the definition of the hyperplane. Tweaking the bandwidth parameter while keeping regularization unchanged intervenes on a crucial component in the formula of the RBF kernel: more specifically, lower values result in single vectors having an extended influence onto the plane, whereas higher ones restrict it to areas closer to the vectors' original positions. This means that as the bandwidth increases the decision boundary becomes more and more dependent on close points, ultimately ignoring instances that are far away. In contrast, lower values result in a hyperplane border that also accounts for distant points. As shown in Figure 5, the first case corresponds to noticeably flexed, elastic decision boundaries, while the second one ends up creating more linear ones. As opposed to the linear case, RBF tends to use almost all of the points as support vectors: a previously misclassified point is employed as a support vector if it is close enough to the decision boundary, thus making the corresponding slack variable active in the optimization process.

3. Least-Squares Support Vector Machine (LSSVM) classifier

3.1 Testing hyperparameters

The least-square variant of the SVM algorithm is explored on the `iris` dataset, which consists of a set of two-dimensional points whose features describe the sepal length and width of a series of specimen of the Iris flower. The aim of the classification is to associate every point to either of two available classes, representing different variants of the same flower. Some first experimentations focus on a polynomial kernel, with increasing `degree` values. A purely linear kernel shows unsatisfactory results on the test set, with an error rate of 0.55. The same is not valid for a second-



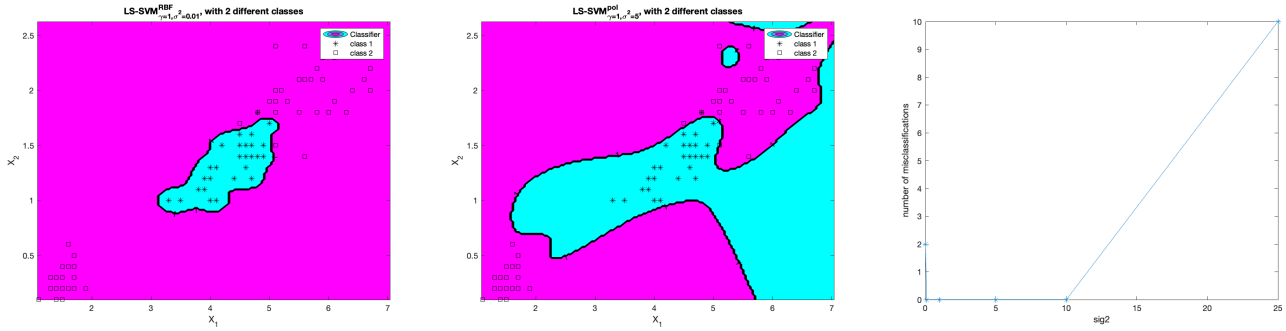
(a) LSSVM, polynomial kernel, `degree=1`, `error=0.55`

(b) LSSVM, polynomial kernel, `degree=2`, `error=0.05`

(c) LSSVM, polynomial kernel, `degree=3`, `error=0`

Figure 6. Classification of the iris dataset employing a polynomial kernels at varying `degree` values

degree kernel; in this case the decision boundary is modeled as a quadratic function on the bi-dimensional plane. Extending the complexity of the kernel by imposing `degree=3` reduces the training error to 0. At this stage the decision boundary is shaped into an ellipsoid: further increasing the value of `degree` seems to overcomplicate the modeling power of the algorithm, forcing it to overfit on the training data. This is exemplified by the case `degree=10`: the resulting decision boundary is extremely complex and would hardly perform correctly on any newly presented dataset. Further trials employ a RBF kernel with a fixed value of `gam=1` and ranging values for the squared kernel bandwidth `sig2`; the selected values are 0.01, 0.1, 1, 10 and 25. This parameter controls the reach of the radial basis functions over the 2D space: greater values yield increased smoothing. The error rate is 0.1 for the first simulation; it then drops to 0 for all the other set values of the parameter, except for the last one. In fact, a value of `sig2=25` results in an overall error of 0.5, which

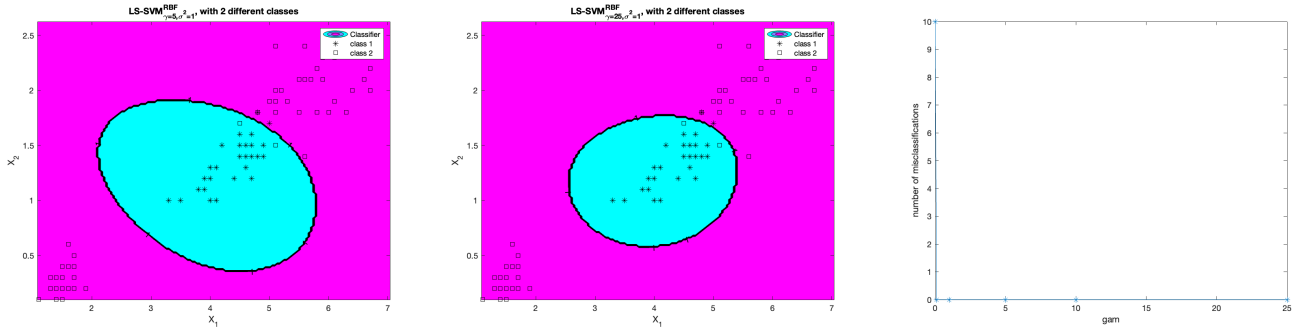


(a) LSSVM, RBF kernel, `gam=1, sig2=0.01`

(b) LSSVM, RBF kernel, `gam=1, sig2=5`

(c) Misclassification vs `sig2`, `gam=1`

Figure 7. Classification of the iris dataset employing RBF kernels and misclassification cost as a function of `sig2`, (`gam=1`)



(a) LSSVM, RBF kernel, `gam=5, sig2=1`

(b) LSSVM, RBF kernel, `gam=25, sig2=1`

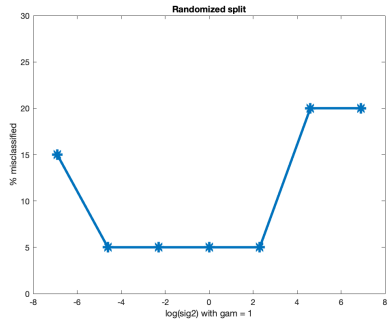
(c) Misclassification vs `gam`, `sig2=1`

Figure 8. Classification of the iris dataset employing RBF kernels and misclassification cost as a function of `gam`, (`sig2=1`)

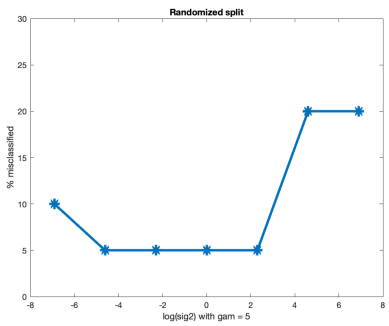
corresponds to purely random predictions on the test set. Fixing `sig2=1` and tweaking the `gam` parameter shows varying effects of regularization on the model: switching from `gam=0.1` to `gam=1` makes all the difference in creating a meaningful model that completely separates the given data points: too low of a value yields random differentiation between the instances on the test dataset. Interestingly enough, as `gam` increases the region for the first class tends to become smaller, meaning that uncertainty in the definition of the decision boundary is reduced. Still, this effect would greatly vary depending on the specific dataset investigated.

3.2 Tuning parameters using validation

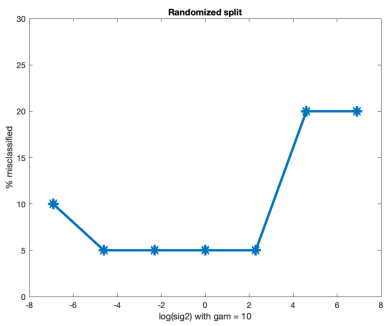
This section focuses on three different techniques for tuning the hyperparameters of a SVM algorithm. A common approach is to split the dataset into three different subgroups: a training set (in this case, 80 instances), for letting the algorithm learn on available data, a validation set (20 instances), for finding the most appropriate values for the hyperparameters, and finally a test set for evaluation of the performance; this in particular stays the same as in the previous exercises. The three sets need to be selected without repetition in order for the process to be sound; moreover, the validation test must not be employed for assessing the performance of the model, otherwise a strongly biased (underestimated) error rate would be computed. In fact, the model would perform on data that has already been processed. First of all, a randomized split of the `iris` dataset is applied; then, different combinations of `gam` and `sig2` values are evaluated, in order to find the ones leading to the best performances. Lower values of `gam` either shrink the range of best-performing `sig2` assignments, or conversely shift the whole graph towards higher percentages of misclassified instances. A common challenge with a similar approach is that the random split could cause the training and validation sets to be unrepresentative of the whole population; this can happen in case the random assignments identify sets containing points of high similarity. Such an issue can be tackled by implementing K-fold Cross-Validation (CV), which averages out the influence of a possible



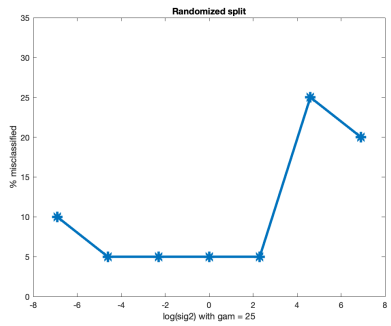
(a) Randomized split, `gam=1`



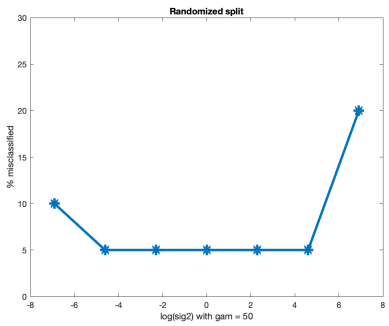
(b). Randomized split, `gam=5`



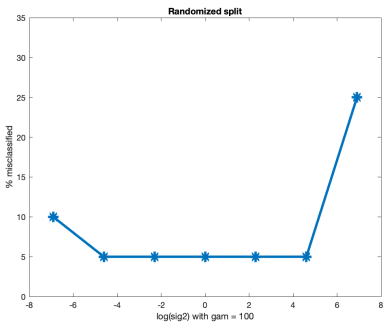
(c) Randomized split, `gam=10`



(d) Randomized split, `gam=25`



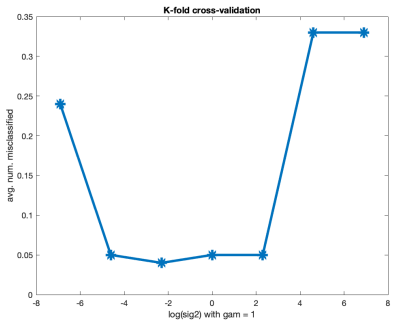
(e) Randomized split, `gam=50`



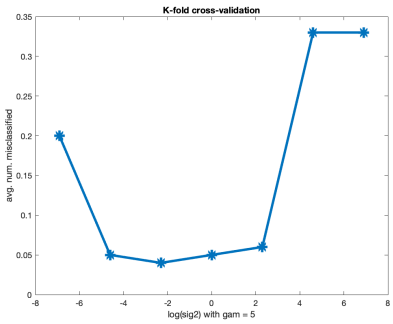
(f) Randomized split, `gam=100`

Figure 9. SVM classification (randomized split, fixed validation set) on the iris dataset by employing a RBF kernel with varying values of `sig2` for set values of `gam`. The number of misclassified examples is plotted against different combinations of the hyperparameter values

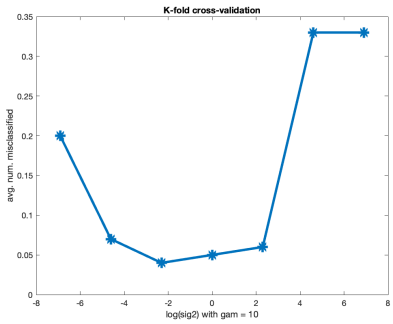
selection bias. Employing this technique on the dataset with a selected amount of folds equal to 10 shows sensibly improved classification performances. The training set is split into 10 disjoint sets; at each step, the model is trained on 9 folds and evaluated on the remaining subset, employed for validation. One can finally select the hyperparameter values that minimize the misclassification error over all 10 subsets. Indeed, the misclassification percentage reported in the plots refers to the mean amount of misclassified instances across all folds, with each fold used as a validation set. A last trial consists of applying the Leave-One-Out (LOO) methodology, a special case of K-fold cross-validation where the value of the amount of folds is set to be equal to the cardinality of the whole dataset. Validation is then run of the single instance that has not been employed for training. The misclassification results are comparable to those obtained with K-fold cross-validation; still, the computational complexity is higher than in the previous case and thus a similar approach should be avoided for bigger datasets. For smaller datasets like this one however, LOO might be a preferable methodology, as it maximizes the amount of instances that the algorithm learns on. It must be noted that the choice of setting K=10 is somewhat arbitrary: selecting this parameter should depend upon a couple considerations. In general, larger values for K result in reduced bias in the estimation of the misclassification error (as the training folds get closer to the whole dataset), but also in higher variance and higher running times. This can be pushed as far as the extreme case of LOOCV, where validation occurs on single patterns.



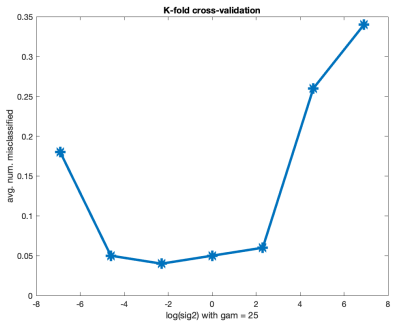
(a) K-fold CV, `gam=1`



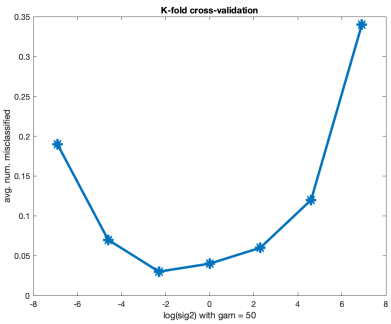
(b) K-fold CV, `gam=5`



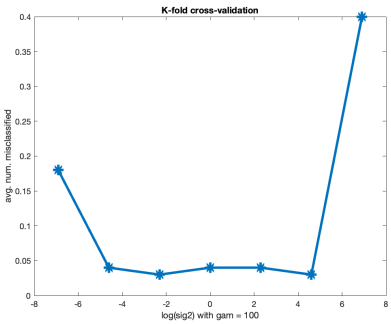
(c) K-fold CV, `gam=10`



(d) K-fold CV, `gam=25`



(e) K-fold CV, `gam=50`

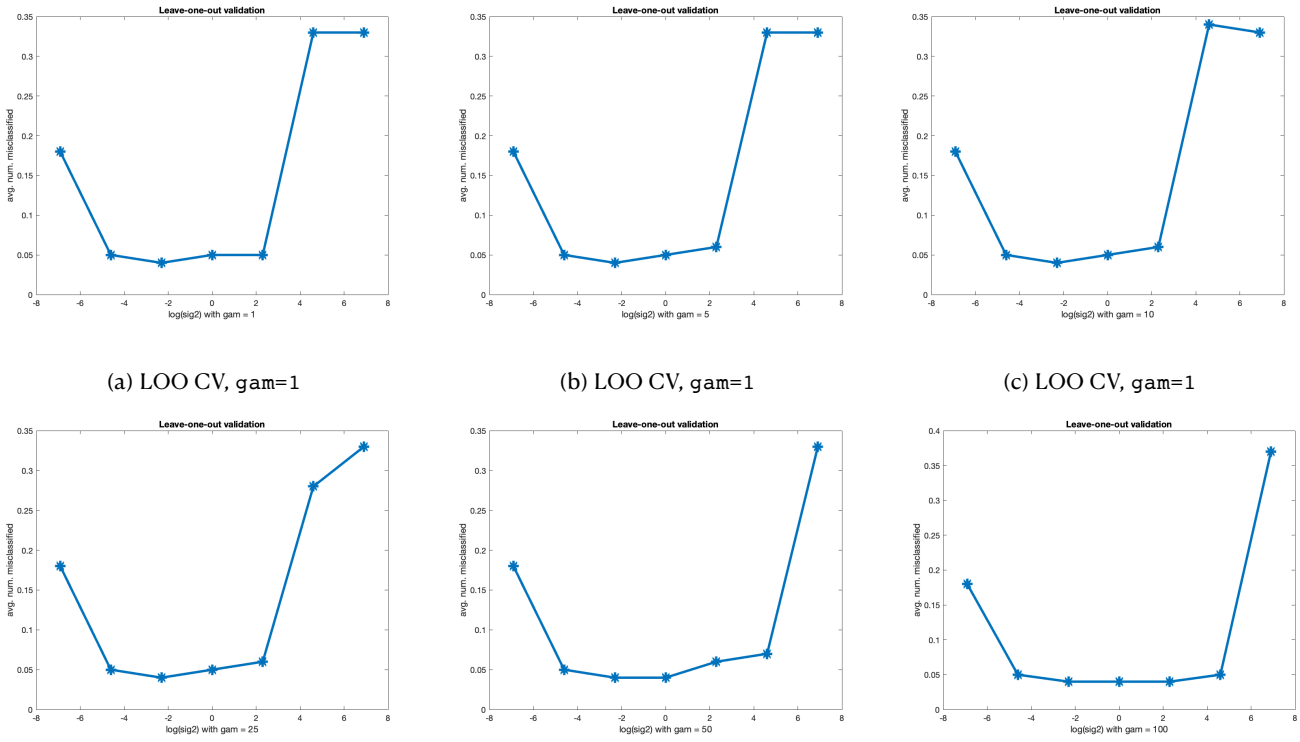


(f) K-fold CV, `gam=100`

Figure 10. SVM classification (10-fold cross-validation) on the iris dataset by employing a RBF kernel with varying values of `sig2` for set values of `gam`. The average number of misclassified examples is plotted against different combinations of the hyperparameter values

3.3 Automatic parameter tuning

Hyperparameters can be tuned in a fully automatic fashion within the LS-SVMLab toolbox. After several trials, running the `tunelssvm` function with the `simplex` algorithm in the `ds` (randomized directional search) mode gives the lowest misclassification results. More specifically, all combinations show strongly varying results for the assignment of the hyperparameters, in spite of the comparable error rates. This may be caused by the randomized initial conditions leading to different local minima in the search process.



(a) LOO CV, `gam=1`

(b) LOO CV, `gam=1`

(c) LOO CV, `gam=1`

(d) LOO CV, `gam=25`

(e) LOO CV, `gam=1`

(f) LOO CV, `gam=100`

Figure 11. SVM classification (leave-one-out cross-validation) on the iris dataset by employing a RBF kernel with varying values of `sig2` for set values of `gam`. The average number of misclassified examples is plotted against different combinations of the hyperparameter values

Method	Cost	γ	σ
`simplex + csa`	`0.03`	`0.053723`	`0.55436`
`simplex + ds`	`0.02`	`8192.6907`	`0.080561994`
`gridsearch + csa`	`0.04`	`0.30186`	`0.076798`
`gridsearch + ds`	`0.03`	`304.7916`	`0.06663311`

Table 1. Results obtained by application of the `tunelssvm` function using different methods

3.4 Using ROC curves

A further way of evaluating the performance of a classifier is to plot a Receiver Operating Characteristic (ROC) curve, where the model's performance is represented onto a 2D diagram with horizontal and vertical axes indicating the False Positive (FP) and True Positive (TP) rates, respectively. In practice the evaluation is done entirely on the test set, since the ultimate goal of the model is to perform well on unseen data. Several plots are created, assessing the performance of the classifier under different pairs of values for the hyperparameters. It is not surprising to see that tuned hyperparameters (`gam=5` and `sig2=0.01`) yield an optimal classifier (Figure 12), covering a maximal area of 1 under the obtained curve. This corresponds to a classifier that, depending on the set threshold, can possibly detect all true positive cases without ever misclassifying any other instances.

4. Bayesian framework

A Bayesian interpretation of the problem can result in assigning probability estimates to each single point: the plot shows the probability of belonging to one class or the other, where more solid shades of color equal to higher values. A first trial is done by employing tuned hyperparameter values, picking `gam=1` and `sig2=0.01`. The decision boundary is rather jagged, as it could be expected by employing such low values for the Gaussian bandwidth hyperparameter; it is also worth noticing how the Bayesian process identifies one of the edges of the boundary in correspondence with tightly-spaced positive and negative instances. This can be observed on the top-right part of the blue region in Figure 13, (a) for example. In general, increasing the value of `sig2` creates a smoother decision boundary. On the other side, intervening on the values for `gam` seems to create blurred regions, indicating that the confidence at which the algorithm assigns either class to a specific instance is reduced. This is intimately tied with the concept of regularization in SVM-type algorithms: higher values of this parameter correspond to increasing misclassification penalization. As a consequence, the Bayesian framework outputs milder probability values for areas where class assignment is not immediately identifiable.

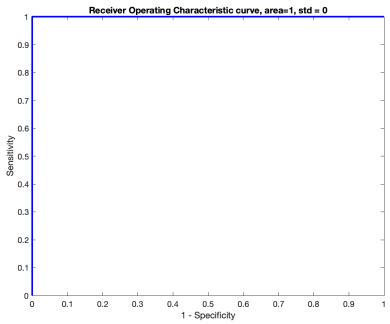
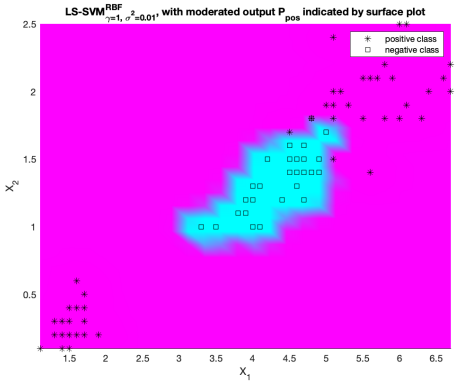
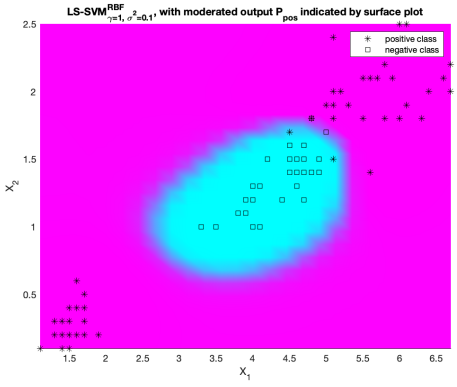


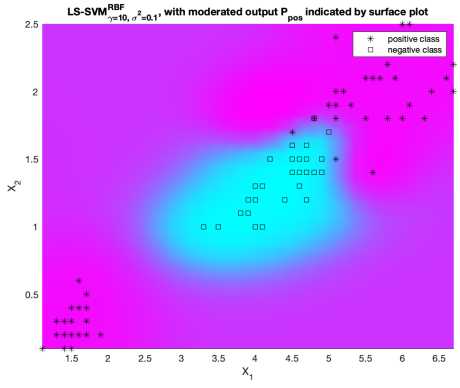
Figure 12. ROC curve for the `iris` dataset, using tuned hyperparameter values, `gam=5` and `sig2=0.01`



(a) Application of the Bayesian framework, `gam=1, sig2=0.01`



(b) Application of the Bayesian framework, `gam=1, sig2=0.1`



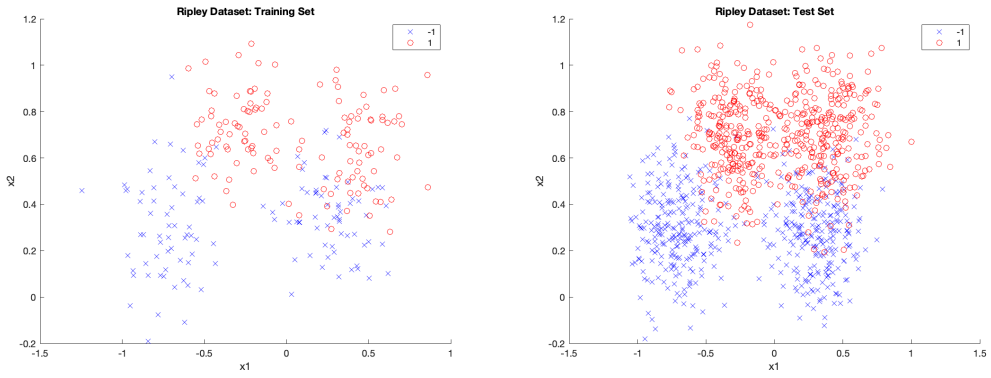
(c) Application of the Bayesian framework, `gam=10, sig2=0.1`

Figure 13. LSSVM classification employing a RBF kernel in the Bayesian framework at varying values of `gam` and `sig2`

4. Classification - Exercises

4.1 Ripley dataset

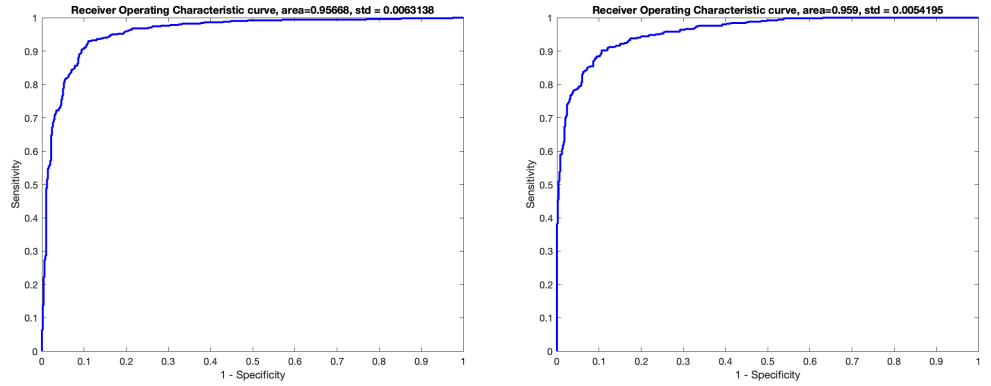
The dataset is quite simple, consisting of a training set (n=250) and a test set (n=1000) of bi-dimensional vectors; as a consequence, the points can be easily plotted and visualized (Figure 14). At a first glance it can already be noted that the points seem to be easily separable, with few overlapping regions. Indeed, it is reasonable to expect both linear and RBF kernels to perform well in this case. As shown in Figure 15, a RBF-type kernel seems to be a marginally better choice .



(a) Ripley dataset, training set

(b) Ripley dataset, training set

Figure 14. Ripley dataset scatterplots



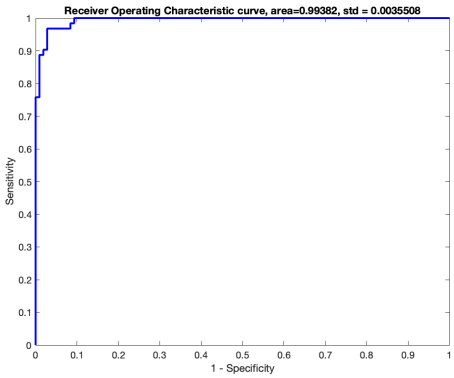
(a) ROC curve for linear SVM

(b) ROC curve for RBF-kernel SVM, tuned hyperparameters

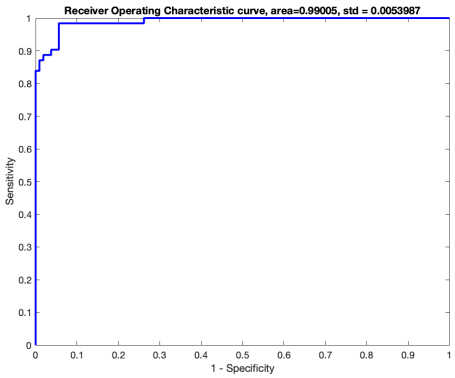
Figure 15. ROC curves for the Ripley dataset

4.2 Wisconsin Breast Cancer dataset

Both the training set (n=400) and the test set (n=169) are composed of 30-dimensional points; as such, visualization of the dataset is hardly performable on a 2D plane. After training the model, it can be observed that both linear and RBF kernels yield a high accuracy result, indicating that classes can be rather easily distinguished by identifying a hyperplane in the feature space. Automated cross-validation through the `tunelssvm` command is employed to fine-tune hyperparameters; the accuracy results obtained are considerably robust, in spite of the high variations in the assignment of the parameters at each new run of the algorithm.



(a) ROC curve for linear SVM

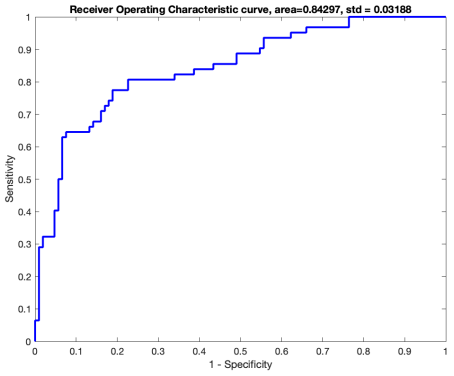


(b) ROC curve for RBF-kernel SVM, tuned hyperparameters

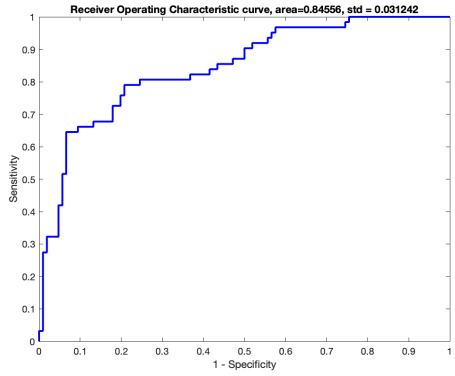
Figure 16. ROC curves for the Breast cancer dataset

4.3 Diabetes dataset

Once again, visualization of the dataset on a 2D plot turns out to be challenging, since the features are 8-dimensional. 300 points are given in the training, while 168 in the test set. As in the previous case, the classification task is binary; employing a RBF kernel minimally improves the performance of the algorithm. As seen with with the `breast.mat` dataset, strong oscillations in the assignment of the automated hyperparameters through `tunelssvm` do not perturb the overall results, hinting again at the presence of multiple optimal points in the hyperparameter assignment space.



(a) ROC curve for linear SVM



(b) ROC curve for RBF-kernel SVM, tuned hyperparameters

Figure 17. ROC curves for the Diabetes dataset

1. Two Gaussians

In the first exercise two clusters of points are created in a two-dimensional plane. These are defined as Gaussian-type random distributions over the feature space. The first group consists of a series of 50 bi-dimensional points distributed according to a Gaussian curve and shifted above by one unit. Conversely, the other one is constituted by 51 points following the same random assignment, but shifted downwards by one unit. This is an instance of a binary classification problem, where only two classes are available (shown in red and blue in Figure 1). A basic approach could consist in drawing a line over the plane; within such a confined framework, the separation can be considered to be optimal as long as the instances are correctly separated over the entirety of the dataset. However, since the two classes are overlapping, it is impossible to define a boundary that perfectly splits the space into the target classes without any misclassified instances. One could relax the problem by finding a line yielding perfect classification for at least one of the classes; anyway, this would not be an optimal approach. A further alternative would be to define a non-linear decision boundary, which can be done by employing the Mercer theorem; this would allow to map the two-dimensional vectors onto a higher-dimensional space on which a separating hyperplane could be defined. This separation plane would then be projected back onto the initial space as to trace the non-linear decision boundary.

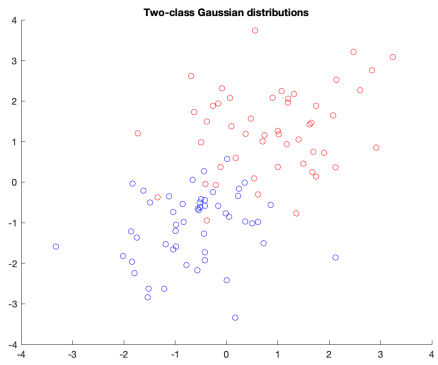
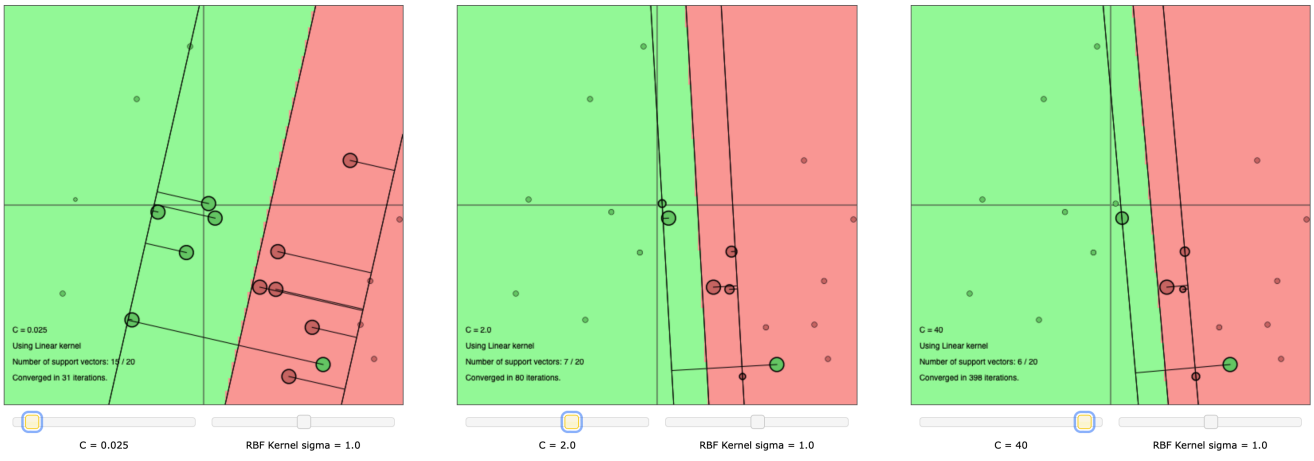


Figure 1. Classification of a two-class variable on the 2D plane

2. Support Vector Machine (SVM) Classifier

The exercise employs the demonstration tool available at `https://cs.stanford.edu/people/karpathy/svmjs/demo/`. This allows for comparison between different models for SVM kernel-based classification. A set of input points belonging to two different classes is introduced onto the 2D plane; first of all, a linear kernel, a basic method of classifying input instances, is assessed. The regularization parameter controls the fraction of points that are used as support vectors for the determination of the separating hyperplane; different values are tested. As it can be seen in Figure 2, imposing a value of 40 results in only 6 of the 20 available points being employed. Furthermore, it must be noted how increasing regularization puts more strain onto the algorithm: in fact, the highest



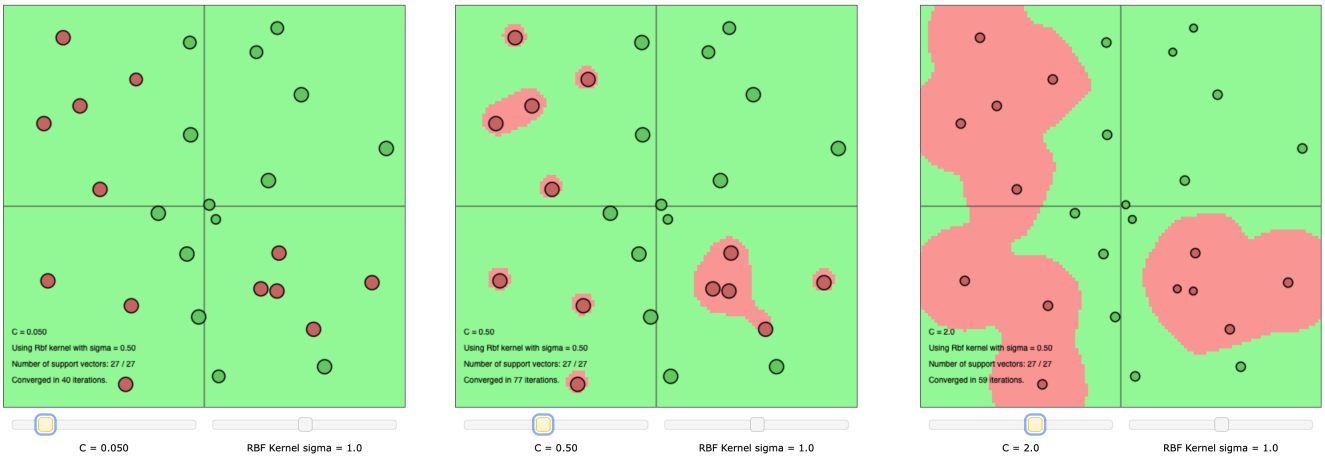
(a) Linear kernel, `c=0.0025`

(b) Linear kernel, `c=2`

(c) Linear kernel, `c=40`

Figure 2. SVM with linear kernel, influence of the `c` parameter

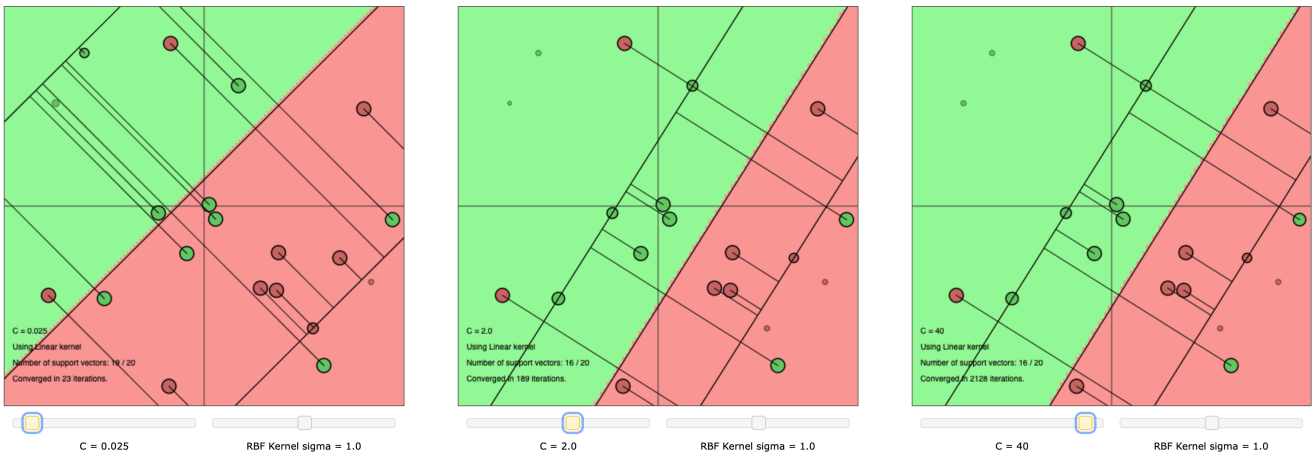
value tested (c) requires 398 iterations for full convergence, as opposed to the 31 iterations required in the first trial (a). Next, a Radial Basis Function (RBF) kernel is tested. As more points are added onto the right side of the hyperplane (green units being inserted onto the green region and red ones being added to the other one) one can observe that the decision boundary becomes more defined, without going through drastic deformations. It is worth noticing how the effect differs when points are inserted onto critical regions, thus becoming support vectors for the classifier; these correspond to units with non-zero Lagrangian multipliers in the dual representation of the target function. In this case, the maximum-margin decision boundary can undergo significant transformations in order to



(a) RBF kernel, `c=0.05`, `sigma=1` (b) RBF kernel, `c=0.5`, `sigma=1` (c) RBF kernel, `c=2`, `sigma=1`

Figure 3. SVM with RBF kernel, influence of the `c` parameter

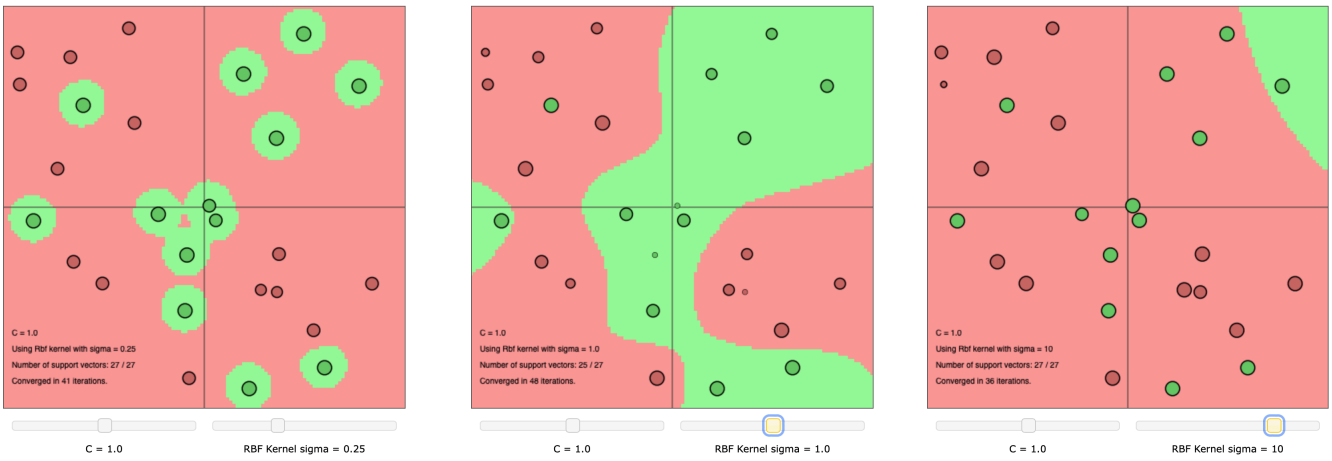
accommodate new training data. Finally, in case points are inserted onto the opposite side of the hyperplane the modifications observed on the boundary can be significant, as the newly added vectors can drastically alter the interpretation that the model operates on the set of data. One of the most evident cases is that of a vector that alters a previously continuously assigned region: the model then needs to re-evaluate the shape of the hyperplane on that area in order to take into account a newly added point. Indeed, these phenomena appear to be analogous to those observed when applying a linear kernel. Different values of the regularization and bandwidth parameters are then explored; as seen previously, growing regularization causes the model to steer away from the possibility of overfitting, while conversely sacrificing its flexibility by employing less points as support vectors; it can be observed that at higher values of the constant the model hardly responds to the



(a) Linear kernel, `c=0.025` (b) Linear kernel, `c=2` (c) Linear kernel, `c=40`

Figure 4. SVM with linear kernel, influence of the `c` parameter on non-separable classes

insertion of new datapoints onto the plane. The underlying rationale is that of controlling the complexity of the modeled function by imposing a bounded norm in some function space; in fact, complex functions tend to have higher norms. Further trials are run by selecting strongly non-separable input features and using a linear kernel. As it can be seen on Figure 4, not all points are



(a) RBF kernel, `c=1`, `sigma=0.25`

(b) RBF kernel, `c=1`, `sigma=1`

(c) RBF kernel, `c=1`, `sigma=10`

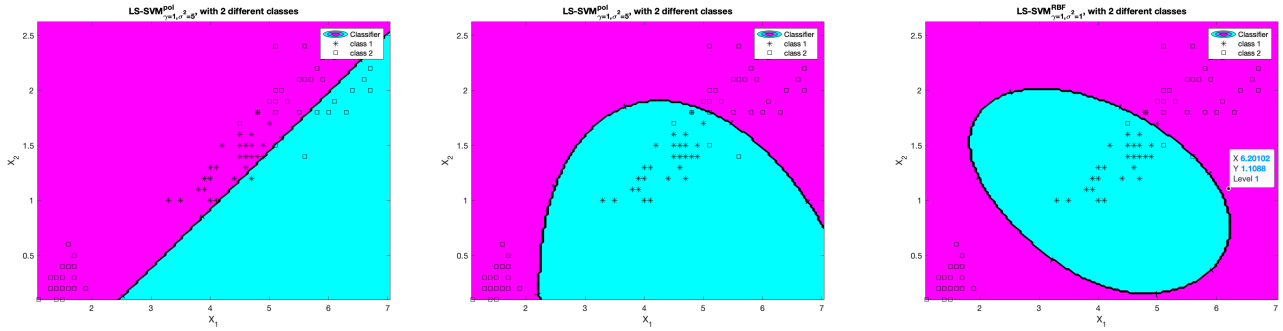
Figure 5. SVM with RBF kernel, influence of the `sig2` parameter on non-separable classes

employed as support vectors, but rather about half of them; this is a major difference with RBF-type kernels, which tend to consider more vectors for definition of the separation hyperplane. In general, switching to a RBF kernel shows decreased values for the misclassification error: the power of this type of functions resides in the adaptability to newly entered points, irrespective of the side. In case the regularization is kept small, the whole region is assigned one unique label (the predominant one); as the value is increased, decision regions start to form: the non-linearity of the kernel can be appreciated in the way the regions evolve (Figure 3). Moreover, the size of the points correspond to their weight in the definition of the hyperplane. Tweaking the bandwidth parameter while keeping regularization unchanged intervenes on a crucial component in the formula of the RBF kernel: more specifically, lower values result in single vectors having an extended influence onto the plane, whereas higher ones restrict it to areas closer to the vectors' original positions. This means that as the bandwidth increases the decision boundary becomes more and more dependent on close points, ultimately ignoring instances that are far away. In contrast, lower values result in a hyperplane border that also accounts for distant points. As shown in Figure 5, the first case corresponds to noticeably flexed, elastic decision boundaries, while the second one ends up creating more linear ones. As opposed to the linear case, RBF tends to use almost all of the points as support vectors: a previously misclassified point is employed as a support vector if it is close enough to the decision boundary, thus making the corresponding slack variable active in the optimization process.

3. Least-Squares Support Vector Machine (LSSVM) classifier

3.1 Testing hyperparameters

The least-square variant of the SVM algorithm is explored on the `iris` dataset, which consists of a set of two-dimensional points whose features describe the sepal length and width of a series of specimen of the Iris flower. The aim of the classification is to associate every point to either of two available classes, representing different variants of the same flower. Some first experimentations focus on a polynomial kernel, with increasing `degree` values. A purely linear kernel shows unsatisfactory results on the test set, with an error rate of 0.55. The same is not valid for a second-



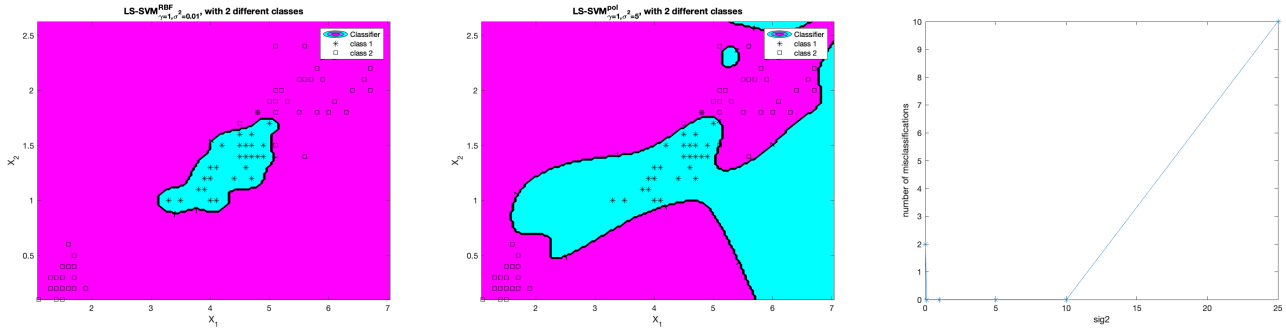
(a) LSSVM, polynomial kernel, `degree=1`, `error=0.55`

(b) LSSVM, polynomial kernel, `degree=2`, `error=0.05`

(c) LSSVM, polynomial kernel, `degree=3`, `error=0`

Figure 6. Classification of the iris dataset employing a polynomial kernels at varying `degree` values

degree kernel; in this case the decision boundary is modeled as a quadratic function on the bi-dimensional plane. Extending the complexity of the kernel by imposing `degree=3` reduces the training error to 0. At this stage the decision boundary is shaped into an ellipsoid: further increasing the value of `degree` seems to overcomplicate the modeling power of the algorithm, forcing it to overfit on the training data. This is exemplified by the case `degree=10`: the resulting decision boundary is extremely complex and would hardly perform correctly on any newly presented dataset. Further trials employ a RBF kernel with a fixed value of `gam=1` and ranging values for the squared kernel bandwidth `sig2`; the selected values are 0.01, 0.1, 1, 10 and 25. This parameter controls the reach of the radial basis functions over the 2D space: greater values yield increased smoothing. The error rate is 0.1 for the first simulation; it then drops to 0 for all the other set values of the parameter, except for the last one. In fact, a value of `sig2=25` results in an overall error of 0.5, which

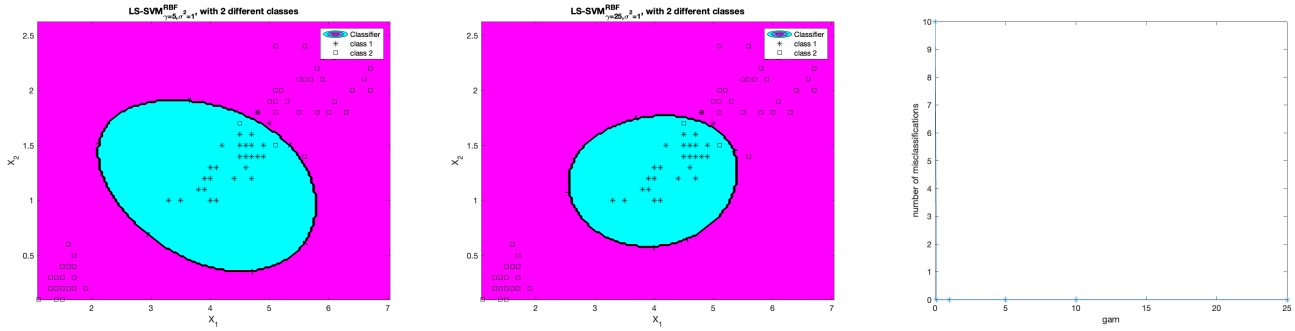


(a) LSSVM, RBF kernel, `gam=1, sig2=0.01`

(b) LSSVM, RBF kernel, `gam=1, sig2=5`

(c) Misclassification vs `sig2`, `gam=1`

Figure 7. Classification of the iris dataset employing RBF kernels and misclassification cost as a function of `sig2`, (`gam=1`)



(a) LSSVM, RBF kernel, `gam=5, sig2=1`

(b) LSSVM, RBF kernel, `gam=25, sig2=1`

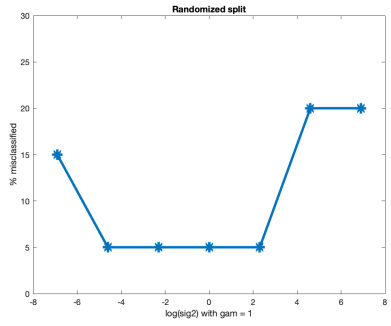
(c) Misclassification vs `gam`, `sig2=1`

Figure 8. Classification of the iris dataset employing RBF kernels and misclassification cost as a function of `gam`, (`sig2=1`)

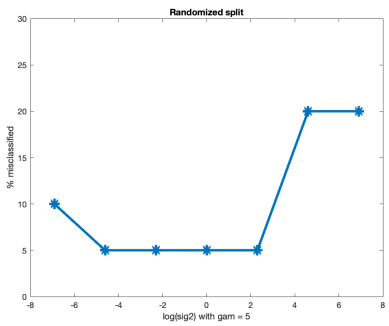
corresponds to purely random predictions on the test set. Fixing `sig2=1` and tweaking the `gam` parameter shows varying effects of regularization on the model: switching from `gam=0.1` to `gam=1` makes all the difference in creating a meaningful model that completely separates the given data points: too low of a value yields random differentiation between the instances on the test dataset. Interestingly enough, as `gam` increases the region for the first class tends to become smaller, meaning that uncertainty in the definition of the decision boundary is reduced. Still, this effect would greatly vary depending on the specific dataset investigated.

3.2 Tuning parameters using validation

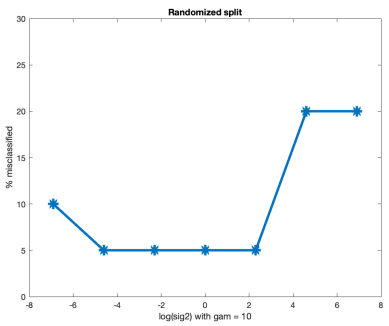
This section focuses on three different techniques for tuning the hyperparameters of a SVM algorithm. A common approach is to split the dataset into three different subgroups: a training set (in this case, 80 instances), for letting the algorithm learn on available data, a validation set (20 instances), for finding the most appropriate values for the hyperparameters, and finally a test set for evaluation of the performance; this in particular stays the same as in the previous exercises. The three sets need to be selected without repetition in order for the process to be sound; moreover, the validation test must not be employed for assessing the performance of the model, otherwise a strongly biased (underestimated) error rate would be computed. In fact, the model would perform on data that has already been processed. First of all, a randomized split of the `iris` dataset is applied; then, different combinations of `gam` and `sig2` values are evaluated, in order to find the ones leading to the best performances. Lower values of `gam` either shrink the range of best-performing `sig2` assignments, or conversely shift the whole graph towards higher percentages of misclassified instances. A common challenge with a similar approach is that the random split could cause the training and validation sets to be unrepresentative of the whole population; this can happen in case the random assignments identify sets containing points of high similarity. Such an issue can be tackled by implementing K-fold Cross-Validation (CV), which averages out the influence of a possible



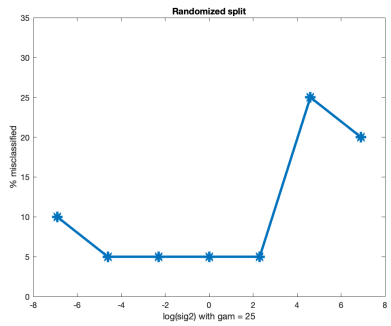
(a) Randomized split, `gam=1`



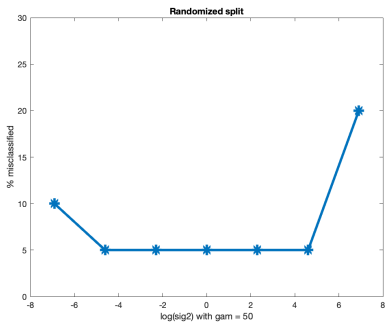
(b). Randomized split, `gam=5`



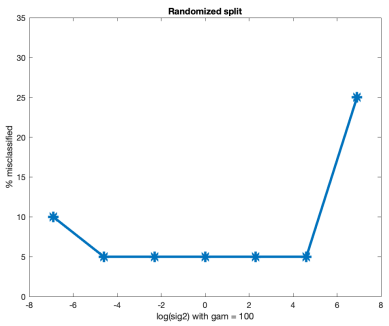
(c) Randomized split, `gam=10`



(d) Randomized split, `gam=25`



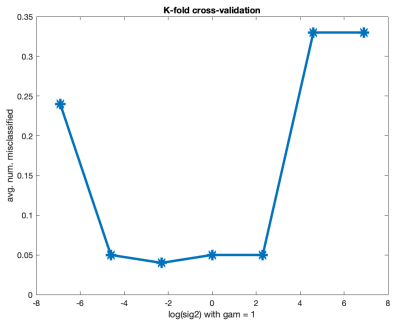
(e) Randomized split, `gam=50`



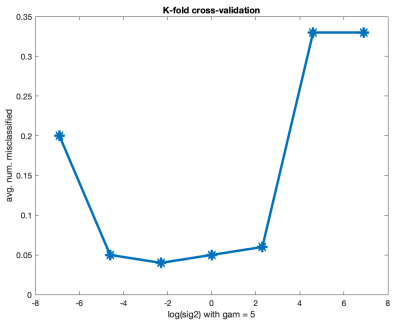
(f) Randomized split, `gam=100`

Figure 9. SVM classification (randomized split, fixed validation set) on the iris dataset by employing a RBF kernel with varying values of `sig2` for set values of `gam`. The number of misclassified examples is plotted against different combinations of the hyperparameter values

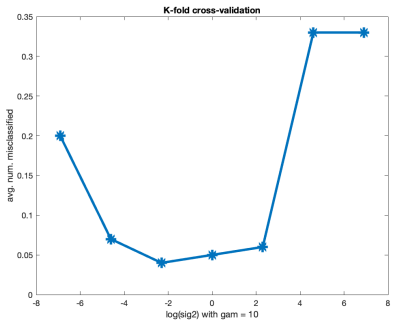
selection bias. Employing this technique on the dataset with a selected amount of folds equal to 10 shows sensibly improved classification performances. The training set is split into 10 disjoint sets; at each step, the model is trained on 9 folds and evaluated on the remaining subset, employed for validation. One can finally select the hyperparameter values that minimize the misclassification error over all 10 subsets. Indeed, the misclassification percentage reported in the plots refers to the mean amount of misclassified instances across all folds, with each fold used as a validation set. A last trial consists of applying the Leave-One-Out (LOO) methodology, a special case of K-fold cross-validation where the value of the amount of folds is set to be equal to the cardinality of the whole dataset. Validation is then run of the single instance that has not been employed for training. The misclassification results are comparable to those obtained with K-fold cross-validation; still, the computational complexity is higher than in the previous case and thus a similar approach should be avoided for bigger datasets. For smaller datasets like this one however, LOO might be a preferable methodology, as it maximizes the amount of instances that the algorithm learns on. It must be noted that the choice of setting K=10 is somewhat arbitrary: selecting this parameter should depend upon a couple considerations. In general, larger values for K result in reduced bias in the estimation of the misclassification error (as the training folds get closer to the whole dataset), but also in higher variance and higher running times. This can be pushed as far as the extreme case of LOOCV, where validation occurs on single patterns.



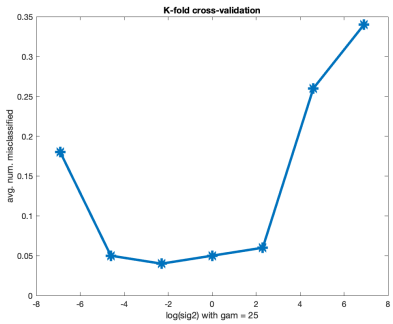
(a) K-fold CV, `gam=1`



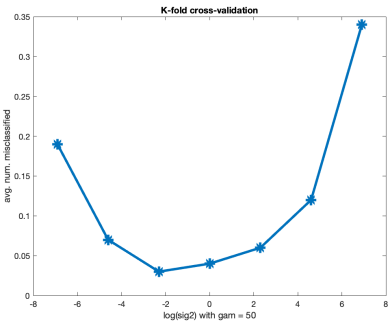
(b) K-fold CV, `gam=5`



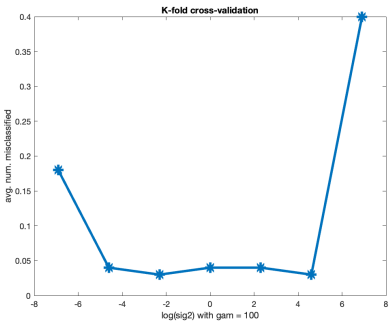
(c) K-fold CV, `gam=10`



(d) K-fold CV, `gam=25`



(e) K-fold CV, `gam=50`

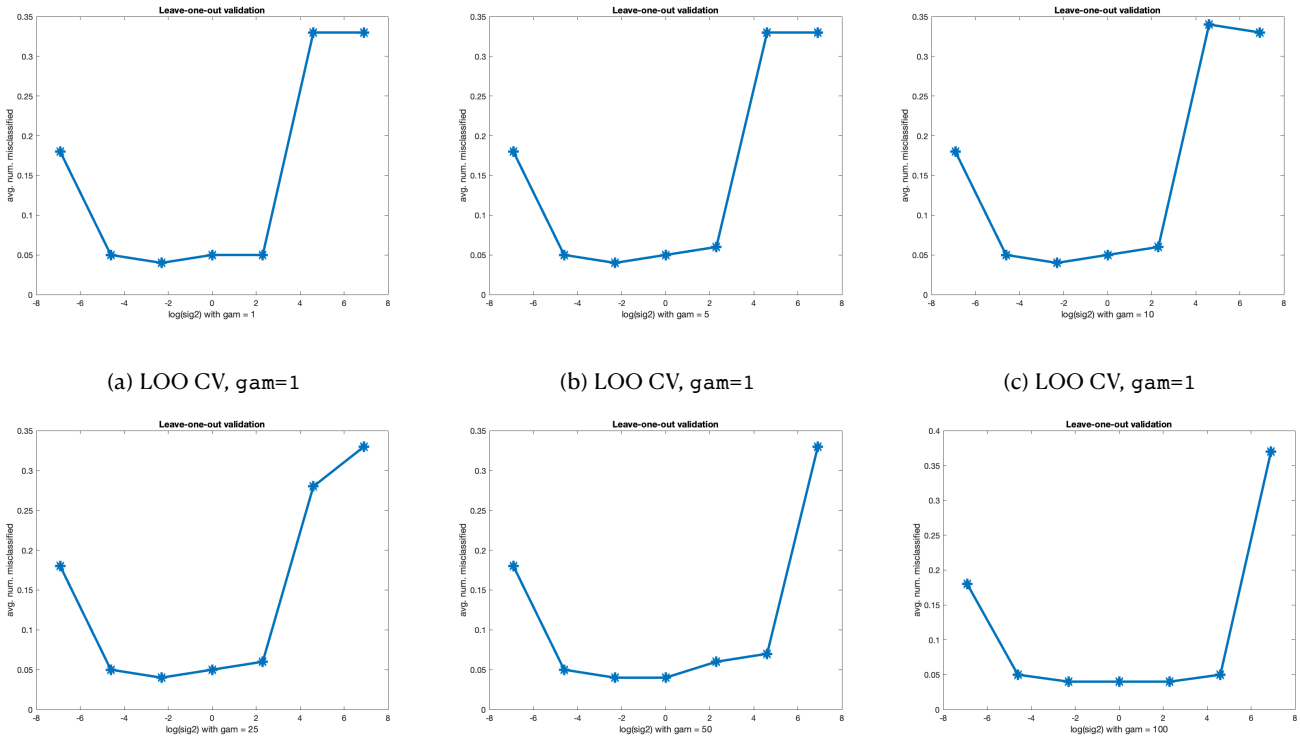


(f) K-fold CV, `gam=100`

Figure 10. SVM classification (10-fold cross-validation) on the iris dataset by employing a RBF kernel with varying values of `sig2` for set values of `gam`. The average number of misclassified examples is plotted against different combinations of the hyperparameter values

3.3 Automatic parameter tuning

Hyperparameters can be tuned in a fully automatic fashion within the LS-SVMLab toolbox. After several trials, running the `tunelssvm` function with the `simplex` algorithm in the `ds` (randomized directional search) mode gives the lowest misclassification results. More specifically, all combinations show strongly varying results for the assignment of the hyperparameters, in spite of the comparable error rates. This may be caused by the randomized initial conditions leading to different local minima in the search process.



(a) LOO CV, `gam=1`

(b) LOO CV, `gam=1`

(c) LOO CV, `gam=1`

(d) LOO CV, `gam=25`

(e) LOO CV, `gam=1`

(f) LOO CV, `gam=100`

Figure 11. SVM classification (leave-one-out cross-validation) on the iris dataset by employing a RBF kernel with varying values of `sig2` for set values of `gam`. The average number of misclassified examples is plotted against different combinations of the hyperparameter values

Method	Cost	γ	σ
`simplex + csa`	`0.03`	`0.053723`	`0.55436`
`simplex + ds`	`0.02`	`8192.6907`	`0.080561994`
`gridsearch + csa`	`0.04`	`0.30186`	`0.076798`
`gridsearch + ds`	`0.03`	`304.7916`	`0.06663311`

Table 1. Results obtained by application of the `tunelssvm` function using different methods

3.4 Using ROC curves

A further way of evaluating the performance of a classifier is to plot a Receiver Operating Characteristic (ROC) curve, where the model's performance is represented onto a 2D diagram with horizontal and vertical axes indicating the False Positive (FP) and True Positive (TP) rates, respectively. In practice the evaluation is done entirely on the test set, since the ultimate goal of the model is to perform well on unseen data. Several plots are created, assessing the performance of the classifier under different pairs of values for the hyperparameters. It is not surprising to see that tuned hyperparameters (`gam=5` and `sig2=0.01`) yield an optimal classifier (Figure 12), covering a maximal area of 1 under the obtained curve. This corresponds to a classifier that, depending on the set threshold, can possibly detect all true positive cases without ever misclassifying any other instances.

4. Bayesian framework

A Bayesian interpretation of the problem can result in assigning probability estimates to each single point: the plot shows the probability of belonging to one class or the other, where more solid shades of color equal to higher values. A first trial is done by employing tuned hyperparameter values, picking `gam=1` and `sig2=0.01`. The decision boundary is rather jagged, as it could be expected by employing such low values for the Gaussian bandwidth hyperparameter; it is also worth noticing how the Bayesian process identifies one of the edges of the boundary in correspondence with tightly-spaced positive and negative instances. This can be observed on the top-right part of the blue region in Figure 13, (a) for example. In general, increasing the value of `sig2` creates a smoother decision boundary. On the other side, intervening on the values for `gam` seems to create blurred regions, indicating that the confidence at which the algorithm assigns either class to a specific instance is reduced. This is intimately tied with the concept of regularization in SVM-type algorithms: higher values of this parameter correspond to increasing misclassification penalization. As a consequence, the Bayesian framework outputs milder probability values for areas where class assignment is not immediately identifiable.

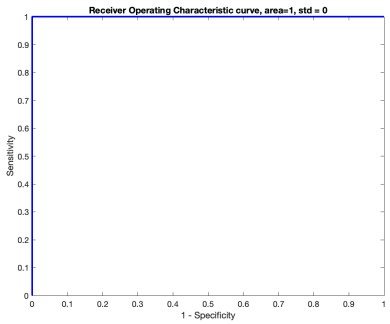
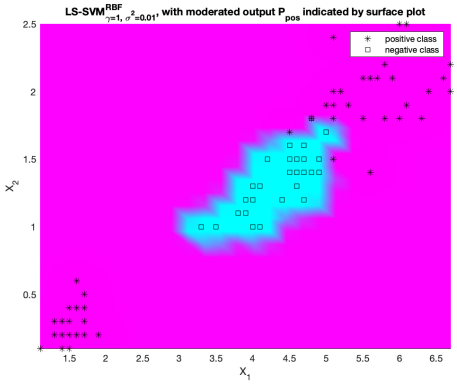
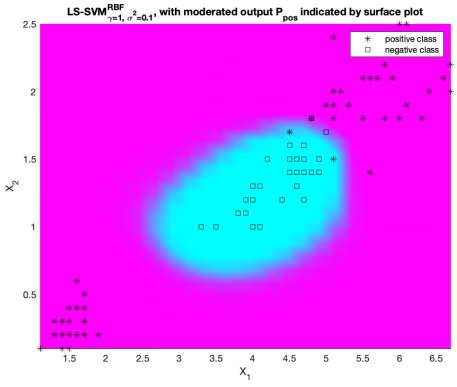


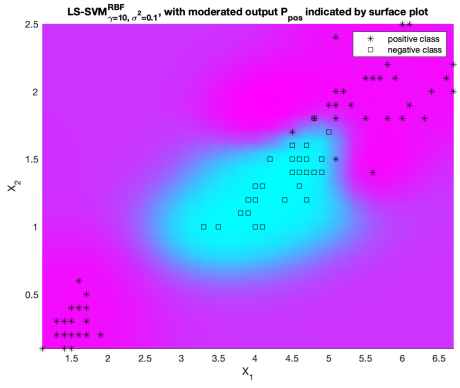
Figure 12. ROC curve for the `iris` dataset, using tuned hyperparameter values, `gam=5` and `sig2=0.01`



(a) Application of the Bayesian framework, `gam=1, sig2=0.01`



(b) Application of the Bayesian framework, `gam=1, sig2=0.1`



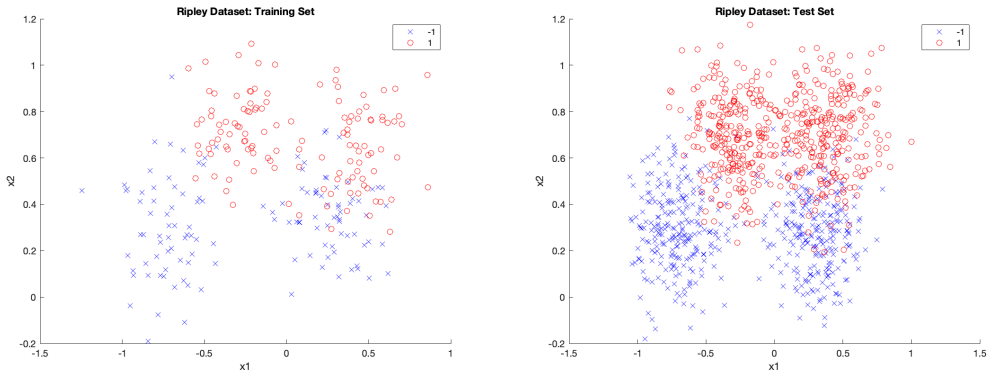
(c) Application of the Bayesian framework, `gam=10, sig2=0.1`

Figure 13. LSSVM classification employing a RBF kernel in the Bayesian framework at varying values of `gam` and `sig2`

4. Classification - Exercises

4.1 Ripley dataset

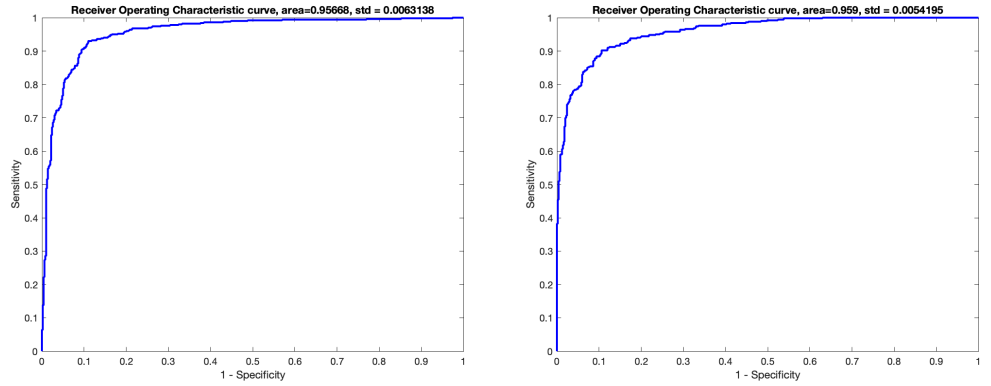
The dataset is quite simple, consisting of a training set (n=250) and a test set (n=1000) of bi-dimensional vectors; as a consequence, the points can be easily plotted and visualized (Figure 14). At a first glance it can already be noted that the points seem to be easily separable, with few overlapping regions. Indeed, it is reasonable to expect both linear and RBF kernels to perform well in this case. As shown in Figure 15, a RBF-type kernel seems to be a marginally better choice .



(a) Ripley dataset, training set

(b) Ripley dataset, training set

Figure 14. Ripley dataset scatterplots



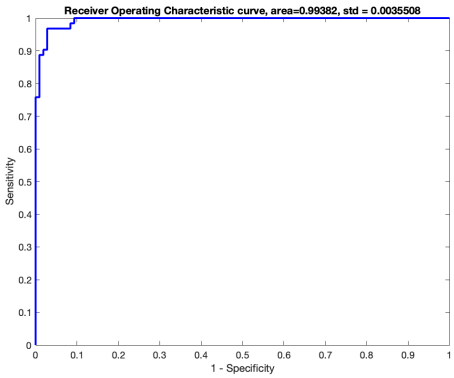
(a) ROC curve for linear SVM

(b) ROC curve for RBF-kernel SVM, tuned hyperparameters

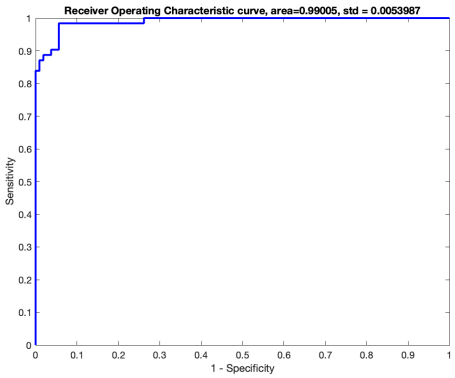
Figure 15. ROC curves for the Ripley dataset

4.2 Wisconsin Breast Cancer dataset

Both the training set (n=400) and the test set (n=169) are composed of 30-dimensional points; as such, visualization of the dataset is hardly performable on a 2D plane. After training the model, it can be observed that both linear and RBF kernels yield a high accuracy result, indicating that classes can be rather easily distinguished by identifying a hyperplane in the feature space. Automated cross-validation through the `tunelssvm` command is employed to fine-tune hyperparameters; the accuracy results obtained are considerably robust, in spite of the high variations in the assignment of the parameters at each new run of the algorithm.



(a) ROC curve for linear SVM

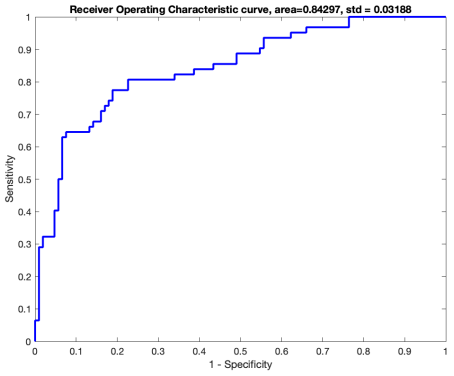


(b) ROC curve for RBF-kernel SVM, tuned hyperparameters

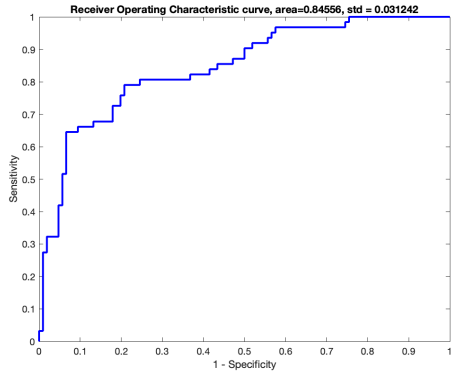
Figure 16. ROC curves for the Breast cancer dataset

4.3 Diabetes dataset

Once again, visualization of the dataset on a 2D plot turns out to be challenging, since the features are 8-dimensional. 300 points are given in the training, while 168 in the test set. As in the previous case, the classification task is binary; employing a RBF kernel minimally improves the performance of the algorithm. As seen with with the `breast.mat` dataset, strong oscillations in the assignment of the automated hyperparameters through `tunelssvm` do not perturb the overall results, hinting again at the presence of multiple optimal points in the hyperparameter assignment space.



(a) ROC curve for linear SVM



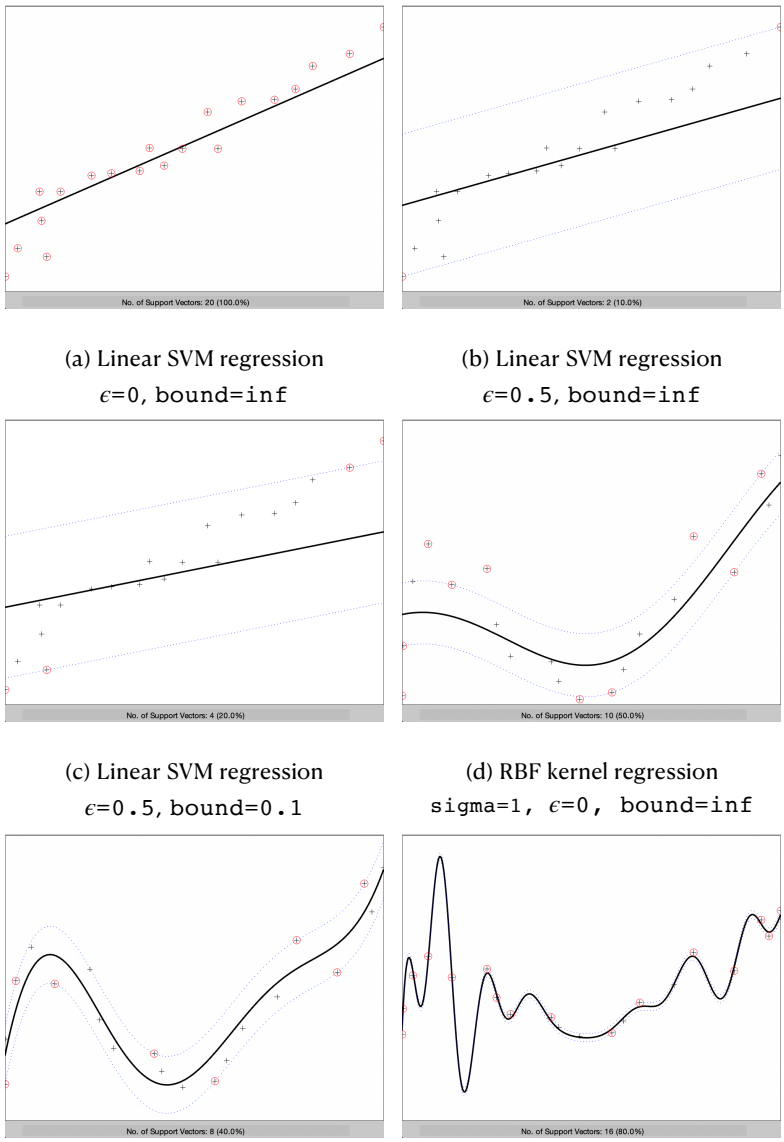
(b) ROC curve for RBF-kernel SVM, tuned hyperparameters

Figure 17. ROC curves for the Diabetes dataset

5. Function estimation

5.1 SVMs for regression

In the following exercises, SVMs are explored in terms of their function approximation abilities, rather than with the target of distinguishing instances between pre-established target labels. The `uiregress` demonstration tool is called upon on MATLAB. Both the C and ϵ values influence the response of the fitted model: the first parameter determines the trade-off between model complexity (in its simplest form, it is reduced to a line in the 2D plane) and the degree at which deviations larger than ϵ are tolerated in the optimization formulation. The value of ϵ instead controls the width of the ϵ -insensitive zone, crucial for the training stage. The more this area is expanded, the more points are potentially discarded as support vectors; this conversely results in flatter and simpler fitting curves. The decision boundaries and, in this case, fitted curves originated by SVMs typically depend on a subset of the training data: the support vectors. It is this property that is referred to as *sparsity* of the obtained SVM solutions. In terms of the dual representation, most of the vectors tend to have null Lagrangian multipliers, thus having no influence in the definition of the fitted model. Such a phenomenon allows for the algorithm to be more easily implemented: in the case of few non-zero elements, memory and time complexities are lowered by a factor inversely proportional to their amount. In practice, a sparse solution requires only a few parameters out of all the input datapoints fed to the model. Testing a linear-kernel SVM algorithm on a rather simple dataset of 20 points helps grasping the effect of tweaking the two hyperparameters mentioned above. An extremely conservative value for ϵ allows for considering all points as support vectors; bringing down the C bound progressively reduces the complexity of the model, which in the linear case is only determined by its slope and intercept, to a horizontal line. At the same time, setting ϵ equal to 0.5 causes the line's equation to be fit by considering a lower amount of support vectors, namely two. It is at this point that the sparsity property of the model can start to be appreciated. More complex datasets are more conveniently modeled by non-linear kernels, which inherently have a higher VC-dimension; a RBF kernel is chosen, with varying values of `sig2`. As seen in other



(a) Linear SVM regression ϵ=0, bound=inf

(b) Linear SVM regression ϵ=0.5, bound=inf

(c) Linear SVM regression ϵ=0.5, bound=0.1

(d) RBF kernel regression sigma=1, ϵ=0, bound=inf

(e) RBF kernel regression sig2=1, ϵ=0.2, bound=inf

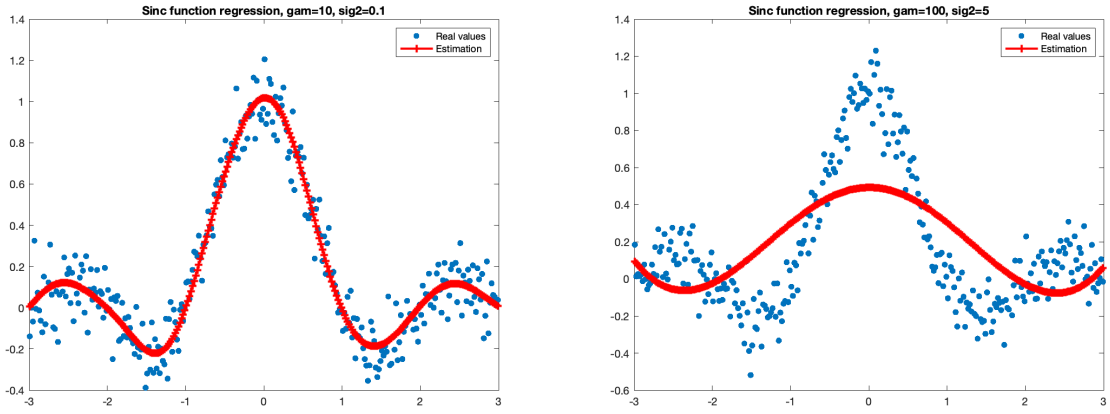
(f) RBF kernel regression sig2=0.1, ϵ=0.5, bound=inf

Figure 18. SVM regression; Figures (a), (b), (c) are all generated on the same dataset and show the influence of varying values for the hyperparameters on linear kernels. The remaining figures are also referred to one unique two-dimensional dataset and explore the same concept on RBF-type kernels

exercises, this parameter controls the influence of the single Gaussian-type curve; as such, it is natural to expect lower values to produce a more tightly-fitting hyperplane intersection with the original 2D feature space. Moreover, as observed in the linear case, tweaking SVM-related parameters alters the flexibility of the model and allows for selection of the amount of data points that are selected as support vectors. Intuitively, it is possible to both consider Least-Square Regression (LSR) and SVMs models as methods to minimize a given cost function: in the first case, the aim is to decrease the sum of the squared differences between real and predicted values; since this is a convex function, a minimum is granted to exist and can be easily found. Regularization methods like Ridge and Lasso enforce limits on the derived weights, at the cost of reducing the flexibility of the obtained model. In the SVM case, instead, the model is trained as a methodology to maximize the margin defined by a set number of support vectors. This is done up to a certain number of exceptions, as granted by the introduction of slack variables in the case of classification and by the ϵ parameter in the case of regression. As with LSR, regularization can be introduced in the function to be minimized. A further crucial possibility introduced by SVMs is that of employing non-linear kernels to identify separation boundaries in the given feature space. By applying the kernel trick it is possible to define more complex curves to fit the training data.

5.2 Regression of the sinc function

This exercises explores the LS-SVM Toolbox by operating on an artificial dataset derived from the sinc function. The generated dataset is conveniently split into a training and a test subsets. Subsequently, different values for `gam` and `sig2` are evaluated; the first parameter controls the trade-off between the smoothness of the fitted curve (described by the regularization term) and the minimization of the approximation errors, while the second one sets the reach of the RBF kernel. A first trial is done by setting `gam=10` and `sig2=0.1`; as it can be seen from Figure 19 (a), the obtained approximation is quite precise, with a Mean Squared Error (MSE) on the test set of 0.0097. A second run employs `gam=100` and `sig2=5`; in this case the results are poorer and do not yield a better approximation of the assigned distribution; this is also clear by inspecting the obtained test set MSE, attesting at 0.0344. Indeed, it is generally hard to find the most suitable assignment of hyperparameters for a SVM algorithm: more specifically, finding the optimal values depends on the aim of the research, apart from the type of tradeoff that can be tolerated in that specific situation. As it is customary, cross-validation could be employed here to select the best-fitting hyperparameter values.



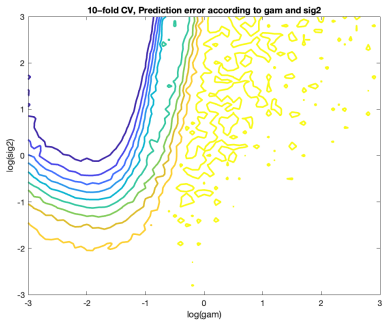
(a) LSSVM on sinc function `gam=10, sig2=0.1`

(b) LSSVM on sinc function `gam=100, sig2=5`

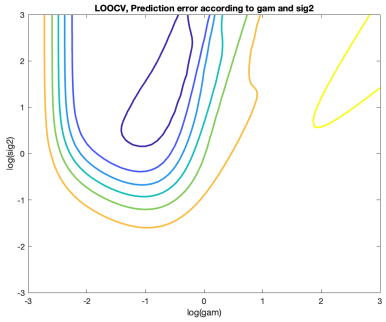
Figure 19. LSSVM regression of the sinc function for varying values of the RBF kernel hyperparameters; blue dots represent the test dataset, while the red lines show the obtained predicted values

5.3 Hyperparameter tuning

Selection of the most appropriate values for `gam` an `sig2` is operated through the typical Cross-Validation (CV) approach. The available training set is first split into ten folds of equal size; at each run of the algorithm, training is performed on all but one of the folds. Then, performance evaluation is done on the remaining split. At the same time, Leave-One-Out (LOO) CV is also tested, where the amount of folds to be derived is set to be equal to the cardinality of the training set. In order to assess different combinations of the chosen hyperparameters, a grid is created, containing values that span across different orders of magnitude. In particular, a set of possible values is created at 0.1 increments between -3 and 3 and then used as exponents in base 10 as to define different set points for the hyperparameters. In both cases, the combination yielding the lowest average MSE on the validation fold is chosen as the best performing value assignment. The SVM model is then trained accordingly on the whole training set, and its performance evaluated on the test set. As it appears from the plots on Figure 20, there seem to be a multitude of value assignments for the hyperparameters that could potentially lead to optimal performance. Although a specific assignment is selected in order to evaluate performance of the model on the test set, there could exist other values that could lead to comparable performance. As a further proof of this, the optimal values obtained for `gam` strongly differ between 10-fold CV and LOOCV; on the other side, the assignment for `sig2` seems to only be altered by a minimal margin.

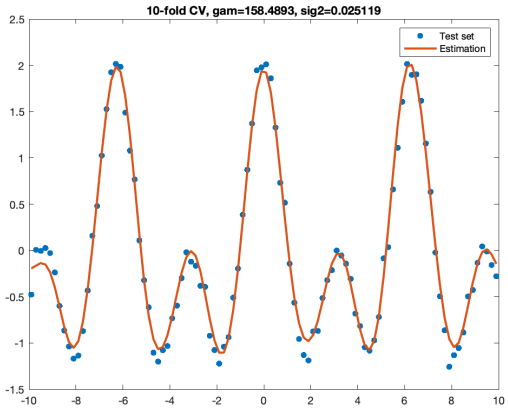


(a) Contour plots for 10-fold CV

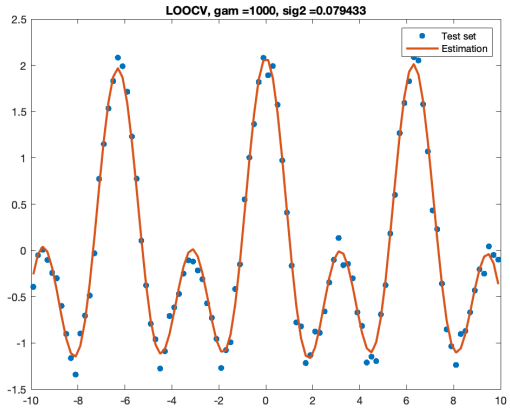


(b) Contour plots for LOOCV

Figure 20. Contour plots showing the average Prediction Error (PE) for varying values of `gam` and `sig2`, plotted on a $\log_{10}$ scale.



(a) 10-fold CV
`gam`=158.4893, `sig2`=0.025



(b) LOOCV
`gam`=1000, `sig2`=0.079

Figure 21. LSSVM regression results for 10-fold CV and LOOCV, using the hyperparameter values obtained from the grid-based tuning process

5.4 Automatic hyperparameter tuning

Automatic tuning is then applied through the `tunelssvm` tool, both implementing the `gridsearch` and `simplex` methodologies. Results for the two inspected parameters are reported on Table 2: a

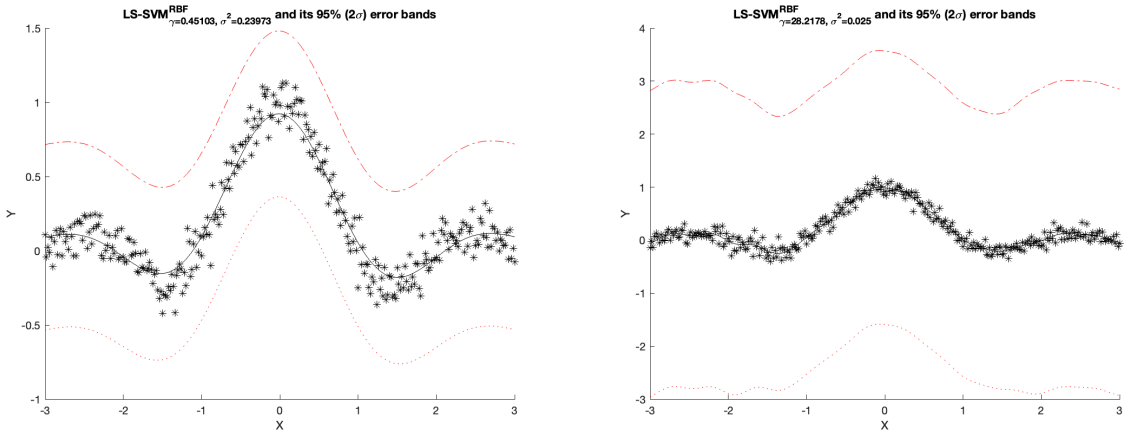
Method	Cost	γ	σ
`simplex + csa`	`0.01`	`24.9951`	`1.1267`
`simplex + ds`	`0.0104`	`222.3201`	`1.4152`
`gridsearch + csa`	`0.0093`	`2643.1`	`1.6578`
`gridsearch + ds`	`0.0087`	`5.3045`	`0.4979`

Table 2. Automatic hyperparameter tuning through `tunelssvm`

great variability in value assignment is observed, which however does not seem to impact the obtained MSE. Graphical results are not reported for all the methods as they all appeared to fit the target curve in comparable ways. As in the case of 10-fold CV and LOOCV, the variability of the `sig2` value is rather contained, contrarily to what is observed with `gam`.

5.5 Application of the Bayesian framework

This section focuses on the use of LSSVM for function estimation in the context of the Bayesian framework. A first run is done by imposing `sig2=0.4` and `gam=10`; the posterior parameter values obtained are 0.23973 and 0.45103, respectively. Figure 22 (a) shows a plot of the estimated function including error bars. The obtained MSE is in this case 0.4397, which is quite satisfactory. Still, another trial is performed by using the optimal value estimates obtained with 10-fold CV as explained in the previous section. As it turns out, the MSE is minimized on the test set, achieving a value of 0.015. The fact that performance be not exactly equal to that obtained by simply plugging the tuned hyperparameter values into the LSSVM problem is due to the different nature of the Bayesian approach, which aims at modeling the generator function that is inherent to the given dataset, in spite of solving a Karush-Kuhn-Tucker linear system. More specifically, the Bayesian methodology can be considered as a three-layered inference process, where information on the data and assumptions on a given prior distribution are employed to compute maximum posterior probabilities at the level of parameters, hyperparameters and model selection. At each step, the likelihood equals the evidence at the previous inference level: such a property grants the method a recursive approach for



(a) Prior values `gam=0.4`, `sig2=10`, posterior values `gam=0.45103`, `sig2=0.23973`

(b) Prior values `gam=158.4893`, `sig2=0.025`, posterior values `gam=28.2178`, `sig2=0.025`

Figure 22. LSSVM regression for the noisy sinc function employing the Bayesian framework approach. Test datapoints are represented by stars, while the estimation is shown as a solid black line. Values for `gam` and `sig2` are inherently computed by the application of the Bayesian resolution methodology

determining the model's moderated outputs, as well as hyperparameters (`gam`) and parameters (`sig2`). A general scheme for Bayesian inference on LSSVM classifiers consists of: normalization and standardization, selection of a model, computation of the effective number of parameters, computation of the optimal hyperparameters (related to maximization of the posterior probability at level 2), model comparison using the expression at level 3 and finally recursive refining of the kernel tuning parameters.

5.6 Automatic Relevance Determination (ARD)

ARD can be employed to detect the relative importance of different input features and ultimately operating a scaling down of the complexity of the problem. This is done by employing the RBF kernel and by re-expressing the kernel function as an exponential with a diagonal matrix among its arguments. The magnitude of the elements on the diagonal is representative of the relevance of the analyzed couple of vectors: the process is looped and elements are pruned until a satisfactory simplification of the problem is achieved. The proposed code selectively identifies only one feature for the creation of a response. The script is run by first tuning parameters within the Bayesian framework and then feeding them to the `bay_lssvmARD` function to output a relevance score, as visible in Figure 23. The obtained results vary across the different runs, probably due to the probabilistic hyper-parameter search process. In this setting, the different inputs refer to the results of the three inference level of the Bayesian framework; as it turns out, the usual unknown parameters for the primal SVM formulation (w and b) are ranked as the most influential ones. An alternative to ARD could consist in employing CV by iteratively training on different features and then assessing the obtained MSE. The selection of features yielding the lowest generalization error could then be considered as the most meaningful one. Depending on the application case and the dimensionality of the input space, it could either be possible to examine different combinations of features or simply take them one at a time; of course, such an approach would be considerably expensive in terms of computation efforts.

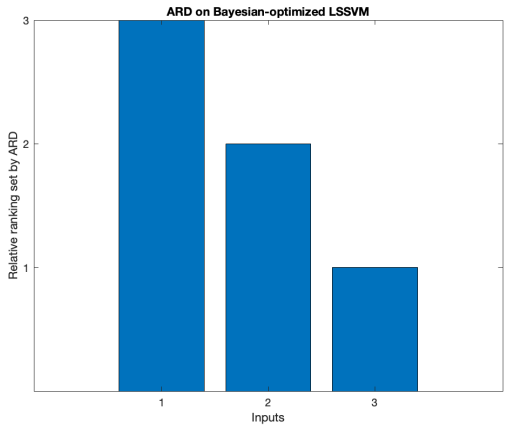
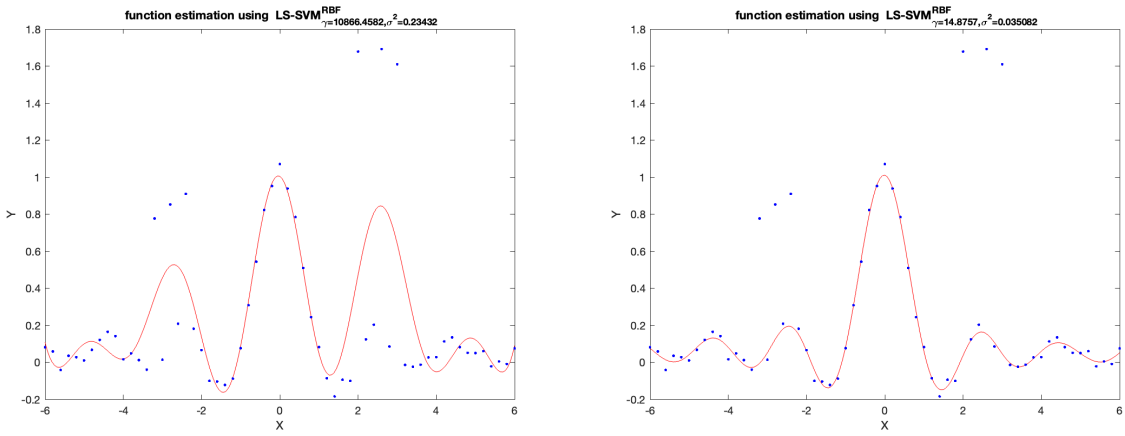


Figure 23. ARD on LSSVM after fine-tuning through the Bayesian framework; the inputs indicate the corresponding inference level outputs

5.7 Robust regression

The effects of outlying input points on a LSSVM algorithm are studied. Two different scripts are implemented: the first one does not strictly focus on the presence of outliers, while the latter aims at building a more robust fitting curve; more specifically, it is an example of employing robust statistics as a tool to infer a weighted SVM model for improved response to noisy or scattered data. Starting parameters are chosen as `gam=100` and `sig2=0.1`; as it appears, the presence of particularly outstanding points does not exert a great pull on the curve. In order to tentatively improve results, `tunelssvm` is employed for automatic parameter selection: the occurrence of vastly varying values for the parameters at each different run confirms the results observed in previous automated parameter-tuning tests. Finally, the best-performing hyperparameters are selected and regression results plotted , as visible in Figure 24. As a next step in the exercise, the Huber loss function is selected for robust regression: the obtained Mean Absolute Error (MAE) is approximately 0.195, which already indicates a valid result. Further tests involve the adoption of different loss functions, always in concurrence with 10-fold cross-validation. The Hampel function yields a MAE of 0.1434; Myriad gives 0.1354, while using the Logistic function results in an error of approximately 0.1547. As the results are



(a) Robust fit response to outliers

(b) Non-robust fit response to outliers

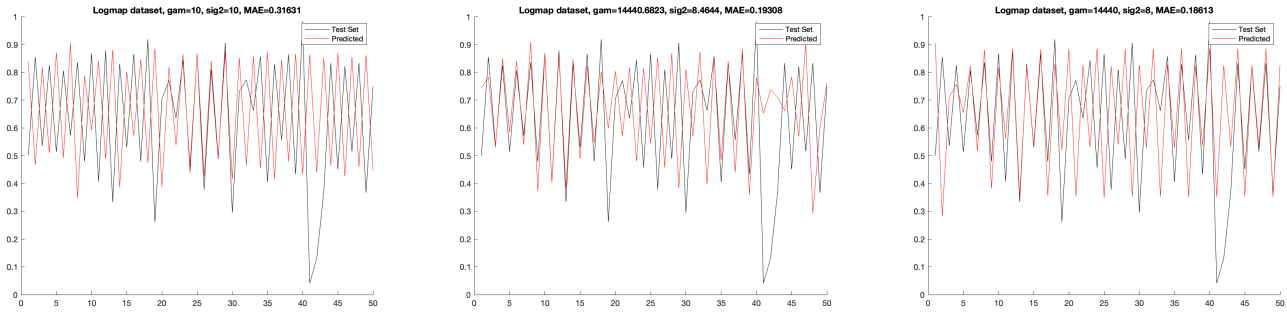
Figure 24. Robust and non-robust function estimation on the sinc function, exploring the influence of outlying points

rather comparable, the plots for these three variants are not presented. Finally, it must be noted that in this section MAE is used for automated cross-validation in spite of Mean Squared Error (MSE): this choice allows for the outliers to be given less weight in the evaluation of the model's performance, since the discrepancies between expected and actually obtained values are not squared.

6. Time-series prediction - Exercises

6.1 Logmap dataset

Time-series prediction is performed on the `logmap` dataset: in particular, the effects of tuning the `gam`, `sig2` and `order` parameters are investigated. It is immediately observed that the default values do not yield very satisfactory results; moreover, estimation appears to be more accurate in the central section of the graph. The obtained MAE is 0.31631. Improvement on the performance can be achieved in multiple ways. Apart from performing cross-validation, it also possible to select appropriate values for the parameters by automatic tuning. A first test consists of applying the `tunelssvm` automated function to the `gam` and `sig2` hyperparameters. Training of the model Is repeated for ten times, and the best-performing model selected. Judging from the obtained plots, it



(a) Prediction with default values, `gam=10`, `sig2=10`, `order=10`, `MAE=0.31631`

(b) Prediction after `tunelssvm` implementation, `gam=1440.6823`, `sig2=8.4644`, `order=10`, `MAE=0.19308`

(c) Prediction after 10-fold CV for determination of order, `gam=1440`, `sig2=8`, `order=5`, `MAE=0.18613`

Figure 25. Time-series prediction on the logmap dataset, employing varying couples of hyperparameters obtained from different tuning processes

is immediately possible to see a better phase-alignment between the test curve and the prediction. Intuitively, it seems that the higher value of the `gam` parameter could contribute to a better-fitting regularization of the model for the analyzed case scenario. Although the MAE is improved to 0.19308, the jagged profiles in the right-end side of the graph still prove to be quite challenging for the time-series estimation. Proceeding further with the exercise, tuning of the `order` parameter has to be taken into account. One idea is to apply 10-fold CV on this parameter only, by simultaneously fixing the values of `gam` and `sig2` at their obtained best-performing levels. The tested values for `order` are selected in the range 1 to 20, with a step of 1. At each iteration, cross-validation is applied on the training set as a way to average-out the influence of a selection bias on the evaluation of the parameter. As it turns out, the best results are obtained for `order=5`, ultimately yielding a MAE of 0.18613. Alternatively, one could also autonomously define a three-dimensional grid on which different set values of the parameters are tested in conjunction, in order to find the best-fitting vector of assignments for the problem. However, this would result in exponentially increasing complexity for the search grid. As an example, selecting a discretization of 1000 values over each parameter's domain would result in a billion test to be run. Hypothetically, these could be performed either by deriving a validation set from the training datapoints, but also by implementing K-fold CV. Of course, the latter case would further worsen the computational power required for the resolution of the problem, by adding a K factor to the number of tests to be foreseen.

6.2 Santa Fe dataset

The aim of the exercise is to train a SVM to use past values from a given dataset in order to predict future evolutions. Performance is, as always, evaluated on test data. This specific approach is called non-linear autoregressive model, as the model is regressed on previous predicted values, thereby yielding a recursive structure; a key aspect is to determine the amount of steps that have to be looked back at into the past in order to capture significant influence on future data. The interest of such a process lies in the possibility of effectively predict unknown, future values. The explored data consists of a training set of 1000 entries and a test set of 200 datapoints, contained in the `santafe.mat` file; these are plotted on Figure 26. The exercise starts with fitting a NAR model with `order=50`

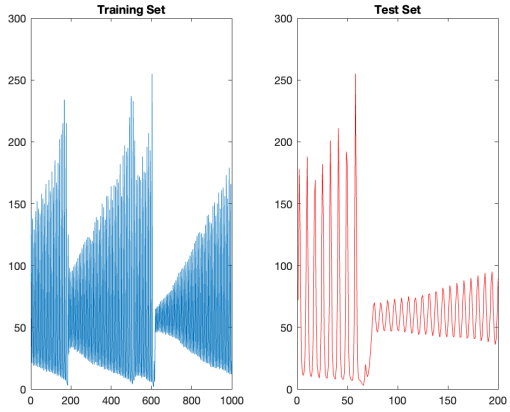
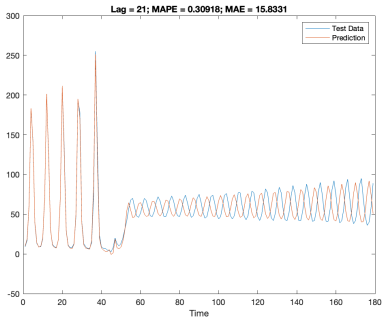
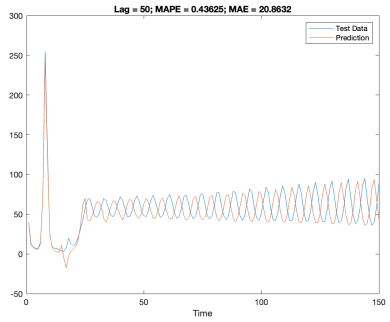


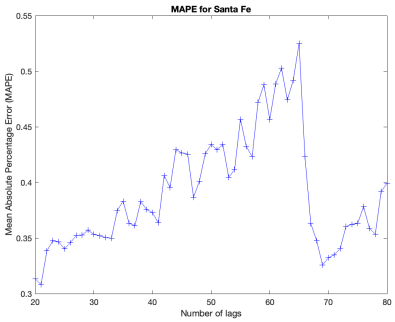
Figure 26. Training and test sets for the Santa Fe prediction problem.



(a) Time-series NAR model `p=21`, MAPE=0.30918



(b). Time-series NAR model `p=50`, MAPE=0.43625



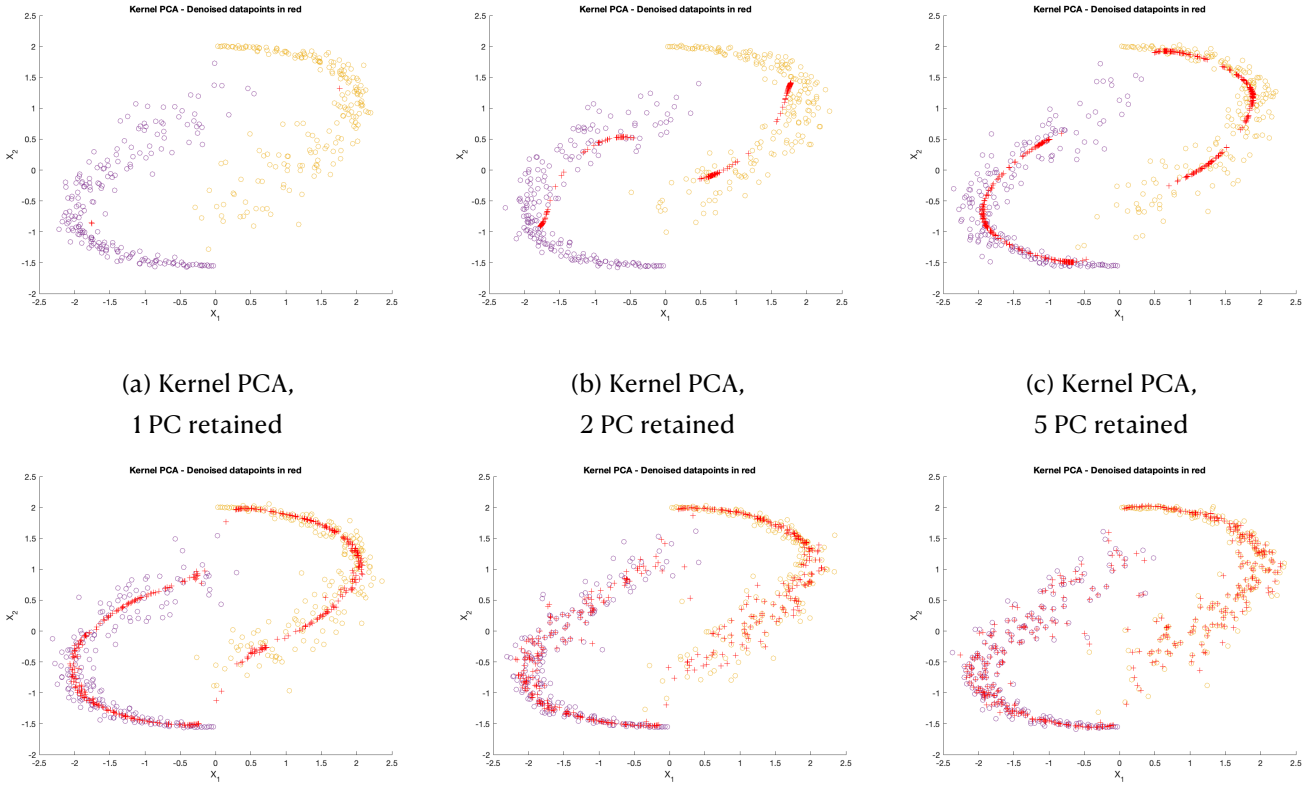
(c) MAPE versus number of lags

Figure 27. (a) and (b) report results for time-series prediction at varying values of `p`. (c) shows the evolution of the MAPE for different values of the same parameter

as the number of lags for the AR component. 10-fold cross-validation with the usual MAE error metric is employed for the `tunelssvm` command, in order to automate the selection of optimal hyperparameters. Afterwards, other settings are investigated, as to clarify whether performance could be improved; with this intent in mind, different values for `order` are tested, specifically in an interval between 30 and 70. For each single trial, Mean Absolute Percentage Error (MAPE) values are plotted so as to identify the best-fitting extent of retrospection in the training dataset. As it appears, the minimum error is obtained for `order=21`, achieving a MAPE of 0.30918: although setting the value at 50 seems a reasonable choice, it would definitely not achieve the best performance score with this metric.

7. Kernel Principal Component Analysis (PCA)

Kernel PCA is an attractive technique that can be employed on complex datasets for a multitude of applications: dimensionality reduction, above all, but also denoising, feature extraction and density estimation. In this section, denoising is explored. An artificial dataset is employed, consisting of 400 points split into two clusters of equal cardinality. As it is clear form a first plot of the set, the points are disposed according to a spiral-like pattern: a standard linear PCA would surely fail in detecting the inherent characteristics of the data, since it cannot project it in to a higher dimensional space prior to executing component analysis; this specific aspect will be later dealt with in greater detail. Data point dispersion is set to 0.3, while the `sig2` value for the RBF kernel is set to 0.5. Reconstruction of the original dataset is tested at increasing numbers of principal components (PCs), namely 1, 2, 5, 10, 15 and 20 (Figure 28). With linear PCA, the size of the correlation matrix is set by the dimensionality of the input space, making it more and more challenging to operate on datasets with lots of features. On the contrary, an important aspect in applying kernel PCA is that the size of the correlation matrix grows together with the amount of selected input datapoints, irrespective of their dimensionality; this of course is a double-edged sword: it allows to operate on high-dimensional input data, but is not a viable solution in case the size of the dataset is considerable. The results hint at improving results as the number of PC increases, as expected. However, after a certain threshold is surpassed, improvements are negligible. Indeed, at a value of 15 it even looks like the model is fitting the applied noise, somehow overfitting the data dispersion over the 2D plane; also, the separation between the two clusters appears to become blurred. Interestingly, only applying one component results in failed detection of the shape of the dataset: in fact, maximal variance is here better



(a) Kernel PCA, 1 PC retained

(b) Kernel PCA, 2 PC retained

(c) Kernel PCA, 5 PC retained

(d) Kernel PCA, 10 PC retained

(e) Kernel PCA, 15 PC retained

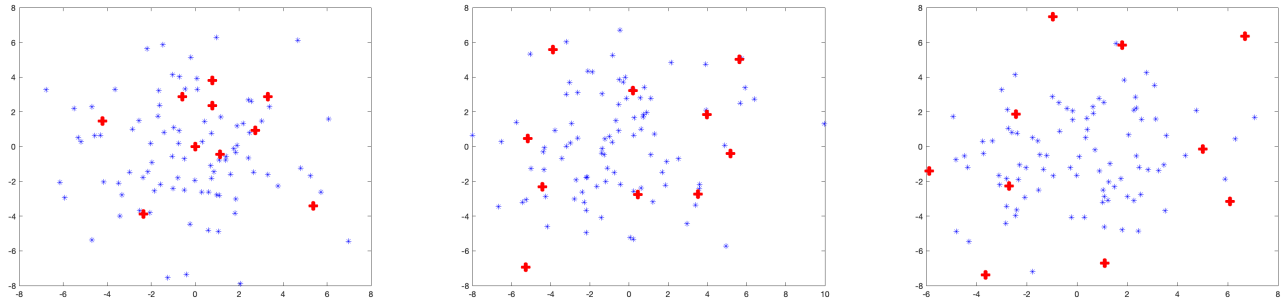
(f) Kernel PCA, 20 PC retained

Figure 28. Employing Kernel PCA to denoise a 2-cluster dataset in a 2-dimensional space. The method is able to learn the non-linear nature of the single data groups. Different amounts of Principal Components (PC) are retained at each single run. Original features are shown in purple and yellow, whereas learned PC are plotted in red

explained by projection onto a non-linear space. As for determination of the amount of PCs to be employed for reconstruction, the same concept applies as in linear PCA: the quality of the obtained output is in fact dependent on the sum of the employed eigenvalues. One can then identify a suitable amount of PCs by observing at which stage the introduction of a new component does not significantly affect the overall sum. As for the determination of the most suitable hyperparameters and kernel parameters, the usual options are viable: K-fold CV, LOOCV or Bayesian regularization. Due to the relatively small size of the considered dataset, K-fold CV seems indeed to be a good solution.

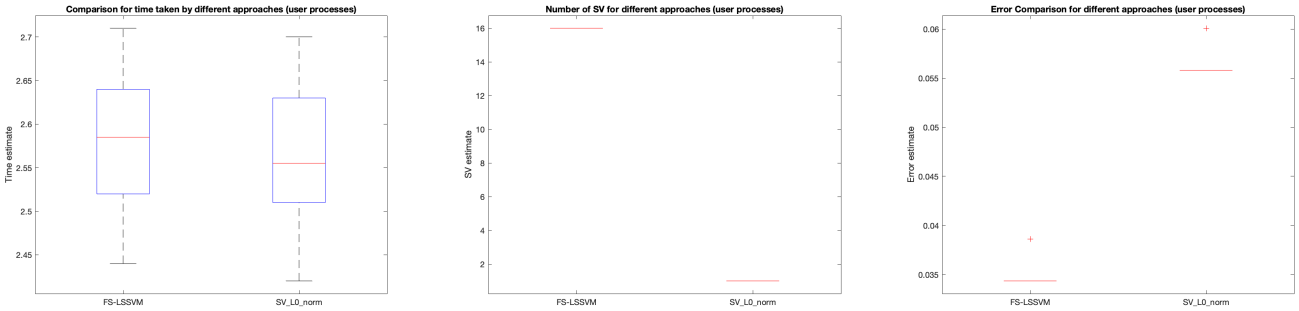
8. Fixed-size LSSVM

In this section, Fixed-size LSSVM is focused on. This methodology allows for prior specification of the number of desired support vectors; then, an iterative process selects the subset that best learns the decision boundary; this can be done, for example, by optimizing the Rènyi entropy criterion of the subset itself. First of all, a RBF kernel is selected and different values for `sig2` are tested (0.01, 0.1, 1). As it is clear from the obtained plots, once the values for the parameter are increased, the algorithm tends to select points that are further away from each other, while contextually converging towards a more even distribution over the feature space. In general, the double nature of the SVM problem can be advantageous in providing different options for the resolution of the inherent Quadratic Programming (QP) problem. On one side, computation in the original feature space is attractive when the input points are low-dimensional, no matter how many: this can be easily understood by considering that each single vector has to undergo dot product multiplication for achieving classification. On the other side, transformation to a higher-dimensional space becomes preferable in case the original input space already has numerous dimensions. Indeed, the Mercer's theorem allows for transferring the problem to the dual representation. In this case the size of the obtained kernel matrix is only dependent on the amount of considered data-points; another convenient aspect of this



(a) `sig2=0.01` (b) `sig2=0.1` (c) `sig2=1`

Figure 29. Selection of ten support vectors for fixed-size LSSVM with RBF kernel at varying values of `sig2`



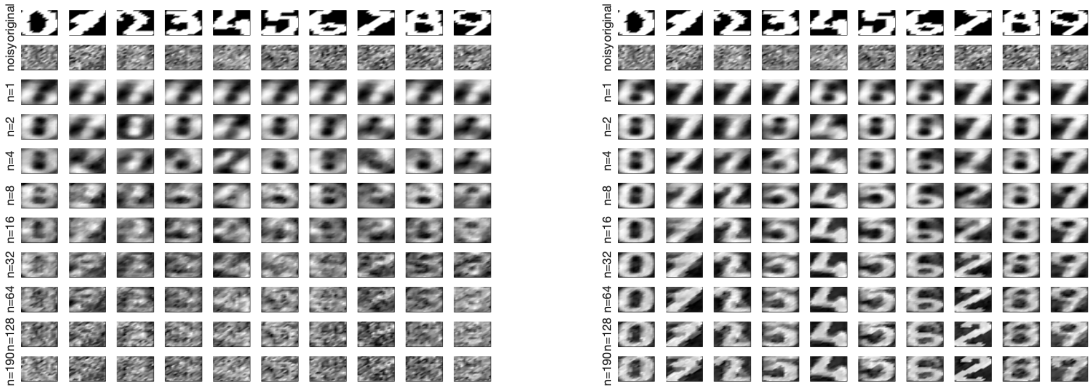
(a) Computational times (b) Number of selected SV (c) Error estimates

Figure 30. Comparison between fixed-size LSSVM and ℓ_0-approximation on classification of the shuttle dataset (first 700 instances)

alternative is that the employed feature map does not need to be explicit: the dot product in the new high-dimensional space is calculated by evaluating couples of points through the selected kernel function. As a rule of thumb, many, low-dimensional vectors are best fed to the primal representation, while fewer, high-dimensional points are best treated by conversion of the QP problem into its dual form. The second part of the exercise compares the performance of fixed-size LSSVM algorithms against ℓ_0-approximation. The script is run multiple times and the best obtained results are displayed. On average, it is evident that in spite of the comparable median errors, the fixed-size variant obtains more variable and error-prone results. As for the number of selected support vectors, ℓ_0 -approximation achieves a smaller subset than the standard fixed-size, yielding a median selection of just a fraction of the vectors considered by fixed-size LSSVM (roughly in between 5-10%); however, variability in the obtained results is definitely higher than observed with the pure fixed-size approach. This is because at each randomization the initialization is different; upon conclusion of the iterative sparsification process it is then possible to end up in different local minima at each run. Finally, computational time estimates seem to be comparable across the two methodologies, thereby indicating that neither of them should be preferable over the other in this regard. It must be noted that the same script is run on Section 10; in that case the shuttle dataset is inspected on the totality of the entry datapoints, in spite of just the first 700. Overall, introducing sparsity into a LSSVM model can have several benefits: it can make techniques more efficient in terms of memory and computational requirements; it can allow for application to large-scale datasets, since out-of-sample extensions would then require less SV and in turn less time for outputting a solution. Lastly, since sparsity can be introduced both at the primal and dual level, the problem of defining a convenient cardinality for the sub-sampled set of SV is by-passed: indeed, the obtained QP solution tends to be highly sparse.

9. Kernel PCA for handwritten digit denoising

In this section, kernel PCA is applied on the reconstruction of a dataset representing handwritten digits from a collection of Dutch utility maps. The influence of the `sig2` parameter is investigated. As a general rule, this is set to be equal to the mean of the variances of each dimension, times the dimension of the training data. Several values for the parameter are tested over a logarithmic scale; this is obtained by multiplying the rule-of-thumb value by a `sigmafactor` coefficient. In general, it seems that extreme values are not beneficial in attaining a well-performing model: increasing `sigmafactor` seems to lead to bad results, while setting it to values as low as 0.0001 shows that much better denoising is obtained by linear PCA rather by its kernel counterpart. Increasing the



(a) Noisy digit reconstruction with linear PCA, noisefactor=1, sigmafactor=1

(b) Noisy digit reconstruction with kernel PCA, noisefactor=1, sigmafactor=1

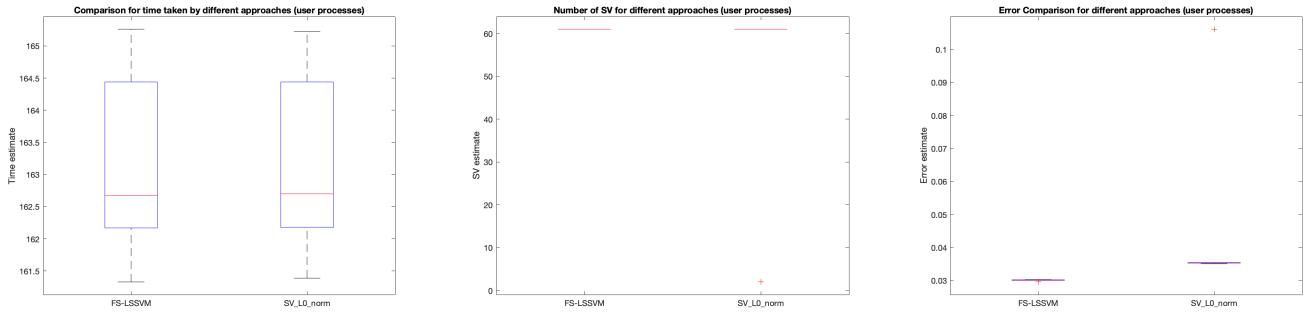
Figure 31. Handwritten digit denoising through linear and kernel PCA

sigmafactor to 0.001 proves how the kernel algorithm entirely fails to reconstruct any of the digits: pure noise is yielded across all possible values for the amount of principal components. At `sigmafactor=0.01` kernel PCA often misinterprets the number 7 by reconstructing it into a 2; Setting a value of 10 for `sigmafactor` seems to give the best results, as both methods effectively succeed in reconstructing the digits across the ten different classes; for linear PCA, this becomes visible with 16 components, while for kernel PCA the successful reconstruction capacity is appreciated as soon as 32 of them are employed. Exploration of the impact of added noise is obtained by setting `noisefactor=1` and then displaying the obtained results; `sigmafactor` is also set to 1. As it is visible, the performance of the linear PCA approach is entirely lost, while the kernel-based algorithm still succeeds in identifying some of the noisy digits. This does not come as a surprise, since the adoption of a higher-dimensional feature map for learning principal components intuitively adds more flexibility and overall robustness to the model.

10. Fixed-size LSSVM - Exercises

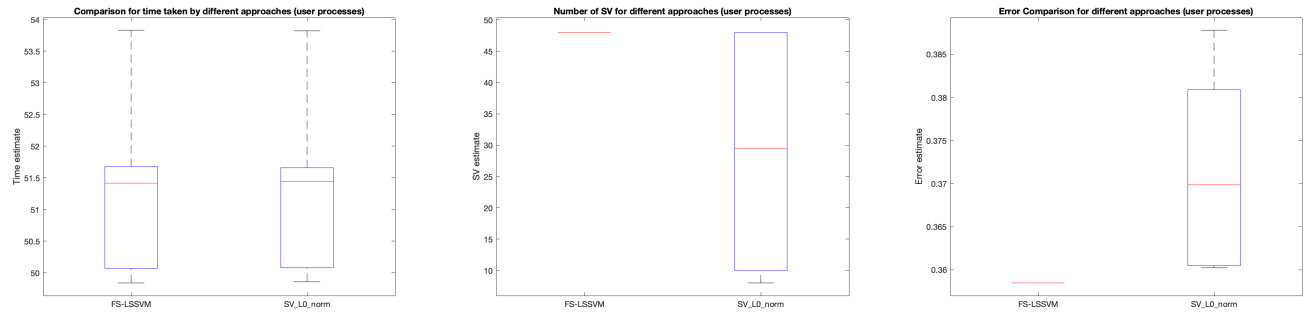
10.1 Shuttle dataset

Fixed-size LSSVM is employed here for classification on the shuttle dataset, grouping together 58000 10-dimensional entries. The data consists of physical measurements operated on a shuttle, with a total of 7 target classes. Inspecting the set already makes it clear that 80% of the points belong to class 1, thereby attesting the default accuracy to 0.8. A solid result then consists in obtaining an accuracy above 0.99. The `k` factor for the script corresponds to a constant employed for determination of the amount of points to be selected; the value is left untouched at `k=3`. Since one of the main drawbacks of LSSVM is the impossibility of achieving a sparse solution, normalization is



(a) Computational times (b) Number of selected SV (c) Error estimates

Figure 32. Comparison between fixed-size LSSVM and ℓ_0-approximation on classification of the shuttle dataset



(a) Computational times (b) Number of selected SV (c) Error estimates

Figure 33. Comparison between fixed-size LSSVM and ℓ_0-approximation on regression of the California dataset

applied with the aim of minimizing the number of non-zero support vector entries. In general, this grants simpler solution representations while causing non-significant losses in accuracy. After running `fslssvm_script.m` for ten consecutive times, the best results are reported: it is observed that the error rate for fixed-size LSSVM is comparable to that obtained with ℓ_0-approximation, although showing a higher median value. As for the number of obtained support vectors, ℓ_0 -approximation seems to consistently yield the same value as its fixed-size counterpart for the number of points selected in the feature space. Finally, the computational times for the two methodologies also appear to be comparable.

10.2 California dataset

On the last section, the same technique is applied on a different dataset, with the aim of performing regression. More specifically, the employed set consists of 20640 observations on different households in a 10-dimensional space; the dependent variable is the natural logarithm of the median house value. Once again, results are plotted in order to compare the performance of fixed-size LSSVM against ℓ_0-approximation. The script is run ten times and the best obtained results are shown. As expected, error estimates are higher and visibly more variable when ℓ_0-approximation is applied. On the other side, the median value for the selected number of support vectors is almost halved, differently from what has been observed in the previous section. Once again, computational time ranges are comparable across the two methodologies.