Classification

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

1 A Simple Gaussian Example

Obtain a line to classify the data by using what you know about the distributions of the data. In which sense is it optimal?

As we know the parameters of the underlying distributions from which the data were generated, we can use these parameters to classify new data generated from these distributions. Using the means of of the distributions for each class, a linear classifier can be made by bisecting the line connecting these two means. The generated data and classifier are given in Figure 1.

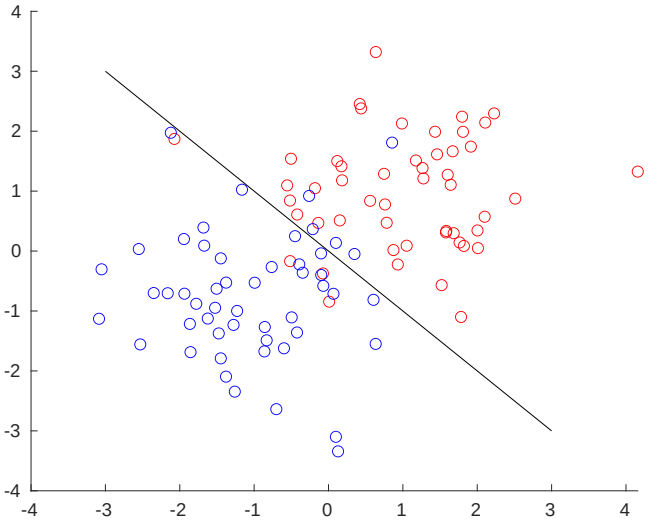


Figure 1: New data and linear classifier generated from underlying distributions.

This classifier is optimal in the sense of the Bayes decision rule. For a given class i and data x, Bayes rule gives that

$$P(C_i|x) \propto P(x|C_i)P(C_i) \tag{1}$$

In the case where $P(x|C_i)$ is normal, we have that the log of the conditional probability for a class C_i given by

$$\log \mathrm{P}\left(C_i \mid x\right) \propto -\frac{1}{2}\left(x-\mu_i\right)^T \Sigma_i^{-1}\left(x-\mu_i\right) - \frac{1}{2}\log\left|\Sigma_i\right| + \log \mathrm{P}\left(C_i\right) \tag{2}$$

for $i = 1, 2$. As $\Sigma_i = diag(1, 1)$ this becomes

$$\log \mathrm{P}\left(C_i \mid x\right) \propto -\frac{1}{2}\left\|x-\mu_i\right\|_2^2 + \log \mathrm{P}\left(C_i\right) \tag{3}$$

If the prior probabilities for each class are equal, then for class i we can use the decision rule

$$D_i = -\frac{1}{2} \|x - \mu_i\|_2^2 \tag{4}$$

Therefore, for a given point x, if $D_1 > D_2$, then x is in the first class, and is in the second class otherwise. This is then equivalent to the decision rule using the bisection of the line between the two class means.

2 Support Vector Machine Classifier

What is a support vector?
Mathematically speaking, support vectors are the data points $\mathbf{x}_i$ for which the Lagrange multipliers α_i are greater than zero. More specifically, consider the case of hard-margin SVMs where we want to find the solution to the minimisation problem. Therefore, depending on the different starting positions, different results are obtained by couple simulated annealing.

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 \tag{5}$$

In order to compute the bias b, the KKT conditions specify that the constraint for sample i must be tight. Hence, b depends only on those points for which $\alpha_i > 0$. Therefore, the solution to the minimisation problem depends only on all points $\mathbf{x}_i$ for which $\alpha_i > 0$.

In the case of soft-margin SVMS (or C-SVMs), the solution to the minimisation problem is given by

$$\min_{\boldsymbol{w},b} \frac{1}{2}\|\boldsymbol{w}\|^2 + C\sum_i \xi_i \tag{6}$$

The KKT conditions specify that in order to compute the bias b, the constraints for $\alpha_i > 0$ and ξ_i must be tight. The solution therefore depends on the points for which $\alpha_i > 0$.

When does a particular datapoint become a support vector? A datapoint $\mathbf{x}_i$ becomes a support vector when the Lagrange multipliers α_i are greater than zero.

When does the importance of the support vector change? Illustrate visually.
The importance of a support vector can change because the parameters of the model are changed (see Figures 6), or because a new datapoint is added. Certain datapoints will not change the importance of support vectors. In the linear case, new datapoints that are outside classification margins and predict the correct class have little effect on importance. In the RBF case, new datapoints at the center of clusters have little effect on importance (see Figure 2 Datapoints that are at the inside the margin (in the linear case) or at the edge of a cluster (in the RBF case) will reduce the importance of similar points further from the boundary (see Figure 3).

What is the role of parameters C and sigma?
The C parameter determines the 'cost' of misclassifying each training example. For large values of C, the SVM optimisation will give a hyperplane with a smaller margin if that hyperplane has better classification. Conversely, a small value of C will cause the optimiser to look for a hyperplane with a larger margin, even if that hyperplane misclassifies more points. In this regard, C is essentially a regularisation parameter which controls the trade-off between achieving a low misclassification error on the training data maximising the width of the margin.

Structural risk minimisation (SRM) is partly implemented in SVMs by the tuning of the C parameter. The C parameter enforces an upper bound on the norm of the weights. If you look at the dual formulation for the optimisation problem for the SVM, there is a box constraint that prevents the dual parameters from exceeding C:

$$\max_{\alpha_k} \mathcal{Q}(\alpha) = -\frac{1}{2}\sum_{k,l=1}^{N} y_k y_l K\left(x_k, x_l\right)\alpha_k\alpha_l + \sum_{k=1}^{N}\alpha_k \tag{7}$$

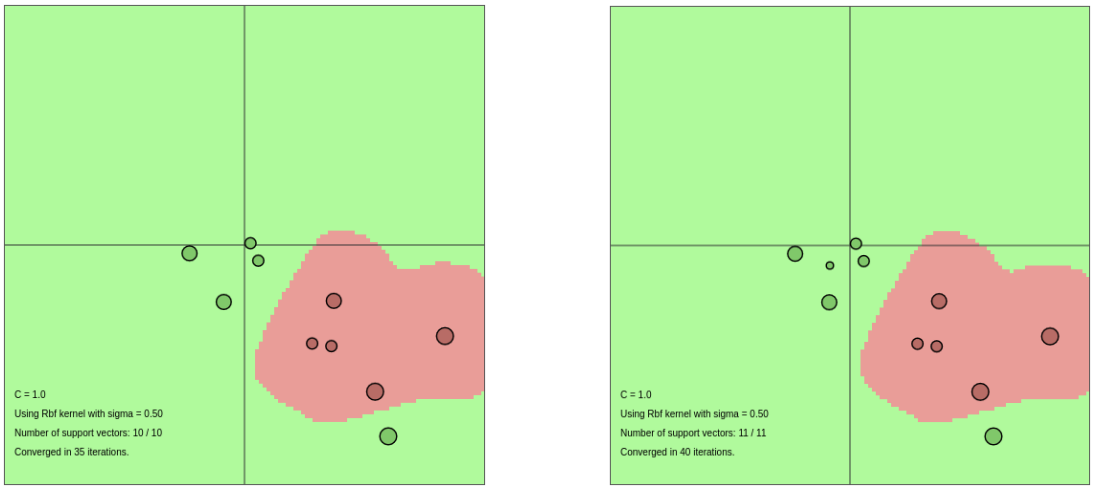


Figure 2: Adding a datapoint at the center of a cluster has little effect on importance of support vectors.

such that

$$\begin{cases} \sum_{k=1}^{N} \alpha_k y_k = 0 \\ 0 \leq \alpha_k \leq c, k = 1, \ldots, N \end{cases} \tag{8}$$

Imposing an upper bound on parameters in the dual formulation also imposes an upper bound on the norm of the primal weights. This means that C indexes a nested set of hypothesis classes, each determined by the upper bound on the norm of the weights. Therefore, as C increases, the complexity of the hypothesis class also increases. In this way, the C parameter implements structural risk minimisation by limiting the complexity of the hypothesis class.

The sigma parameter determines the region of influence of each datapoint. If the value of sigma is low, then every datapoint will have a small region of influence. Conversely, high values of sigma mean that datapoints will have a large region of influence. This is clear from the equation for the Gaussian RBF kernel:

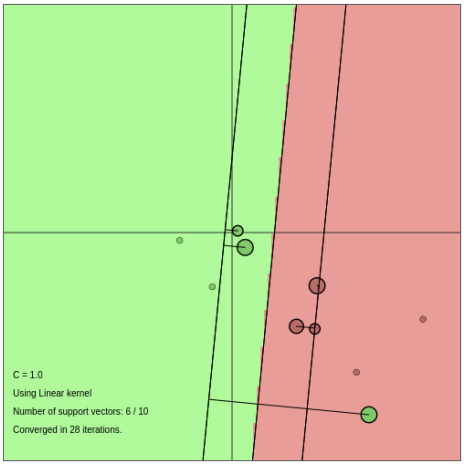
$$K(x_i, x_j) = \exp\left(-\|x_i - x_j\|_2^2 / \sigma^2\right) \tag{9}$$

With a low value of sigma, the SVM decision boundary will be dependent on only those points that are closest to the decision boundary, effectively ignoring points that are farther away. A high sigma value will give a decision boundary that considers points that are further from it. As a result of this, low values of sigma tend to give highly flexible decision boundaries, and high values of sigma result in more linear decision boundaries.

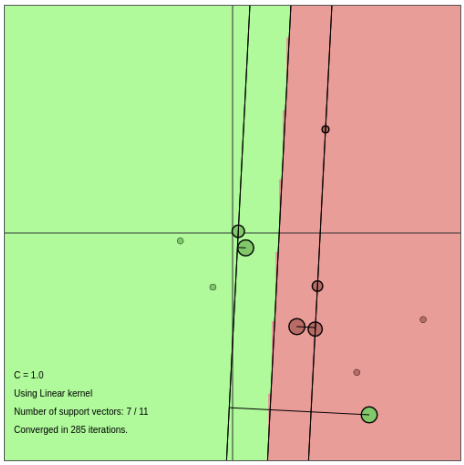
What happens to the classification boundary if you change these parameters. Illustrate visually.
In the case of the linear kernel, as C is increased, the margin must become smaller and in order to accommodate for this, the angle of the linear decision boundary changes so that the support vectors are closer to the boundary. This is very clear in Figures 4 and 5.

For the RBF kernel, as C is increased, the decision boundary becomes less flexible and more smooth (see Figure 6). As Sigma is increased, the range of effect of each data point is increased. This is clear in Figure 7, where for low values of sigma each point has a classification boundary, and for high values this boundary tends toward linear.

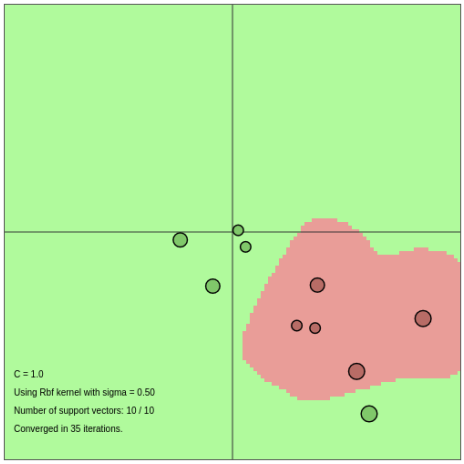
What happens to the classification boundary when sigma is taken very large? Why?
For the Gaussian RBF kernel, a large sigma means a large standard deviation of the Gaussian, and so the influence of each data point is larger. When sigma is very large, the model is too constrained and cannot capture the complexity or "shape" of the data. The region of influence of any selected support vector would include the whole training set. The resulting model will behave similarly to a linear model with a set of hyperplanes that separate the centres of high density of any pair of two classes. Keerthi et al wrote a paper titled *"The Asymptotic behaviour of Support Vector Machines with Gaussian Kernel"*, in which they discuss



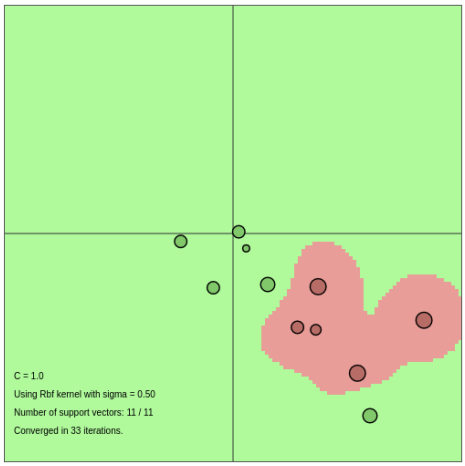
(a) Linear kernel with standard dataset.



(b) Linear kernel with new datapoint within margin.



(c) RBF kernel with standard dataset.



(d) RBF kernel with new datapoint at cluster boundary.

Figure 3: Illustrating how adding datapoints can change the importance of support vectors. *(Top-right)* Adding a new datapoint with margins for linear kernel changes the importance of other support vectors. *(Bottom-right)* Adding a new datapoint at the boundary of a cluster in RBF kernel changes the importance of support vectors

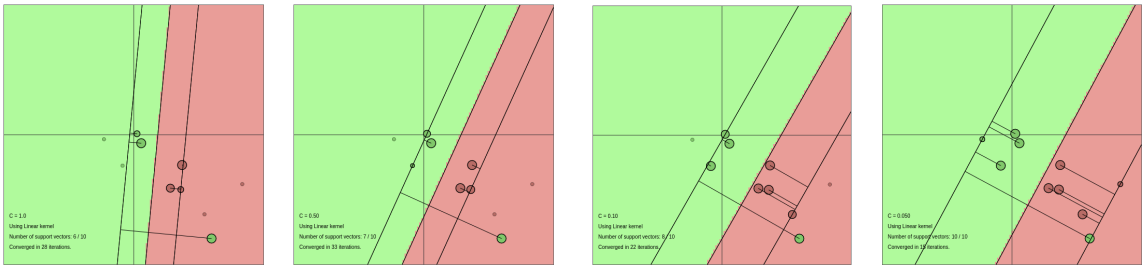


Figure 4: Linear kernel with C at values of 1, 0.5, 0.1, and 0.01.

this phenomenon [1]. They analysed the behaviours of the SVM classifier when C and/or Sigma take very small or very large values, to understand the hyperparameter space that will lead to efficient heuristic ways of searching for points in that space with small generalisation errors. They demonstrate mathematically that the RBF kernel is equivalent to a linear SVM when $\sigma \longrightarrow \infty$.

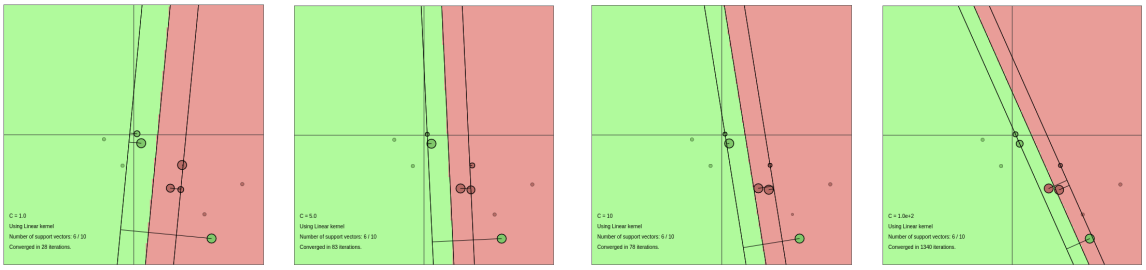


Figure 5: Linear kernel with C at values of 1, 5, 10, and 100

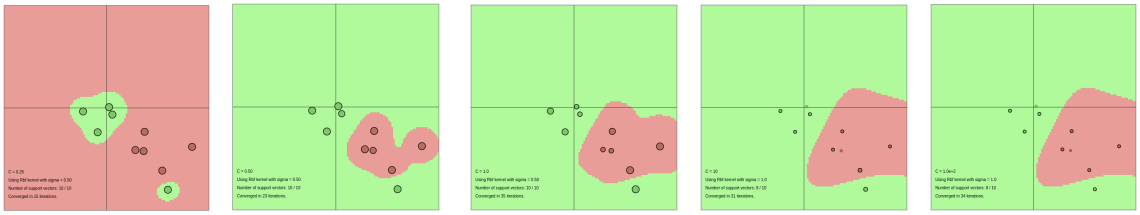


Figure 6: RBF kernel with C at values of 0.25, 0.5, 1, 10, and 100. Increasing the value of C clearly decreases the importance of each data point, and the number of support vectors.

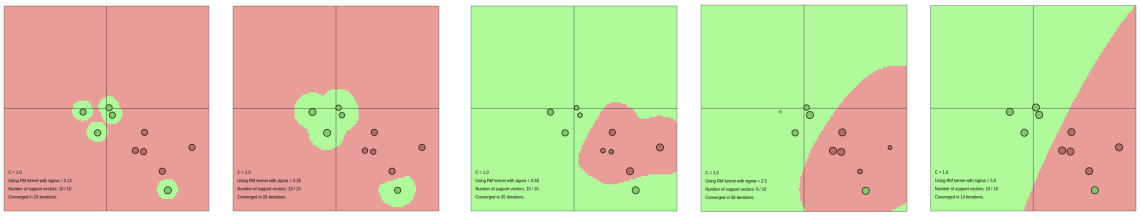


Figure 7: RBF kernel with Sigma at values of 0.13, 0.25, 1, 2.5, and 5. Increasing the value of Sigma clearly decreases the range of effect of each datapoint, ultimately resulting in a linear classification boundary at high values of sigma.

3 Least Squares SVM Classifier

Try out a polynomial kernel with different degrees. Assess the performance on the test set. What happens when you change the degree of the polynomial kernel?
A polynomial kernel was used with degrees 1 to 10. For degrees 1 and 2, the accuracy on the test set was 45% and 95%, respectively. For degrees greater than 2, the accuracy was 100%. Changing the degree of the polynomial kernel increases the flexibility of the decision boundary.

Using a RBF kernel with squared kernel bandwidth, try out a range of σ^2 and γ parameters. Assess performance on the test set and compare results with the provided script.

Figure 8 shows the accuracy of the LS-SVM over a range of possible sigma and gamma values. The results are the same as those given in the provided script.

3.1 Tuning Parameters using Validation

Compute the performance for a range of gamma and sigma values using the random split method, 10-fold CV and LOOCV. Visualise and interpret the results.

The misclassification error for the split method, 10-fold CV, and LOOCV was calculated using a grid search of sigma and gamma values between 10^{-3} and 10^{3}. Figure 9 shows the misclassification error for each grid search.

The random split naturally gives more varied accuracy than k-fold CV or LOOCV, as bias will be introduced

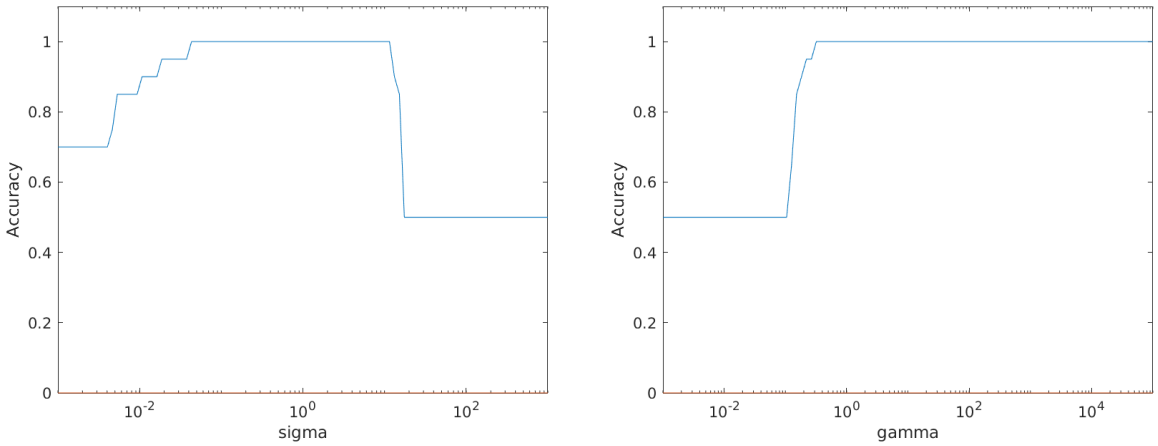


Figure 8: *(left)* Model accuracy on the test set for a range of sigma values, with gamma set to 1. *(right)* Model accuracy for a range of gamma values, with sigma set to 5.

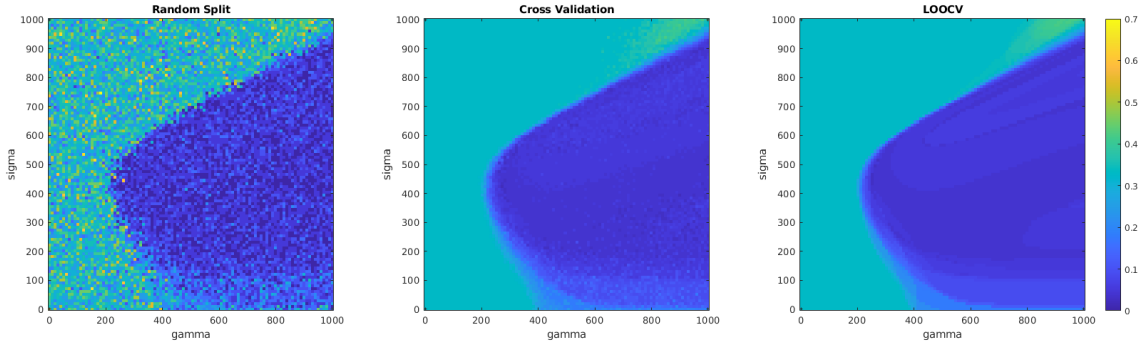


Figure 9: Colour plots showing misclassification error against gamma and sigma for different parameter tuning methods. The colour scale is shared for each plot.

based on this random selection of training and validation sets. Both k-fold CV and LOOCV show similar results, with k-fold CV showing more variability in accuracy, as evidenced by the 'spottiness' present. In LOOCV, the gradations of misclassifcation error values are more clearly delineated. This is to be expected. However LOOCV comes with increased computational cost, though that is not an issue here. There is a clear pattern in parameter space, showing decreased misclassification error for a range of sigma and gamma values.

3.2 Automatic Parameter Tuning

Try out the different 'algorithm'. What differences do you observe? Why do the obtained hyperparameters differ a lot in different runs? What about the cost? Computational speed? Explain the results.
The gamma and sigma parameters returned by both grid search and simplex vary massively between each run. The obtained parameters differ a lot in different runs because hyperparameter tuning is a non-convex optimisation problem, with many optima. The starting points given by the coupled simulated annealing are different each run, and therefore the optimum is also different due to the non-convexity of the cost-function. On average, the grid search gives slightly better performance.

Standard grid search is comparatively inefficient as it spends a lot of time investigating hyper-parameter settings that are nowhere near optimal. The efficiency of grid search can be improved by iteratively evaluating grids of increasing resolution over the parameter space (as implemented in this package). However, this is only applicable when the parameter space has low dimensionality and grid search quickly becomes computationally infeasible in higher dimensions. With NM simplex, adding an extra dimension only adds one extra point to the simplex and so it scales near-linearly with the number of dimensions. However, this optimisation problem has only two dimensions, and so this curse of dimensionality does not play a role here.

Being honest, I'm struggling to compare the computational speed of NM simplex and grid-search in low dimensions, and cannot find a definitive answer in the literature. I read Bergstra and Bengio's paper *"Random Search for Hyperparameter Optimisation"*, in which they argue that random search methods are more efficient in both high- and low-dimensional parameter spaces as not all hyperparameters are equally important to tune [2]. They state that *"grid search experiments allocate too many trials to the exploration of dimensions that do not matter and suffer from poor coverage in dimensions that are important"*, and conclude that random search methods such as NM simplex are more efficient in most cases.

As both grid search and Simplex gave similar prediction accuracies on our dataset, we can compare their computational efficiency by determining the number of cost function estimations used by each algorithm to reach this accuracy. The `tunelssvm` output for simplex showed it had 26 function estimations. The output for grid search showed that it had two iterations (each iteration has corresponds to a 'zooming' of the grid). The square root default number of function evaluations at each zoom level is given by the `grain` parameter, which has a value of 7 in the `tunelssvm` output. This means there are 49 function evaluations at each iteration. As there are two iterations for grid search on this dataset, we can conclude that the simplex algorithm took fewer cost function estimations to reach the same accuracy.

3.3 Using ROC Curves

In practice, we compute the ROC curve on the test set, rather than on the training set. Why?
We compute the ROC on the test set because it was not used to train the model in any way, and is therefore the set of data that can help us to estimate generalised performance. Computing the ROC curve on the training set would give an unrealistic estimation of how the model generalises to new data.

Generate the ROC curve for the iris.mat dataset (use tuned gam and sig2 values). Interpret the result.
Figure 10 below gives the ROC curves evaluated on the test set, using parameters estimated with both simplex (left) and grid search (right). For these specific parameters, the ROC curve for the simplex algorithm indicates perfect classification on the test data, with an AUC of 1. For the grid search, the AUC is 0.97 and has slightly worse performance.

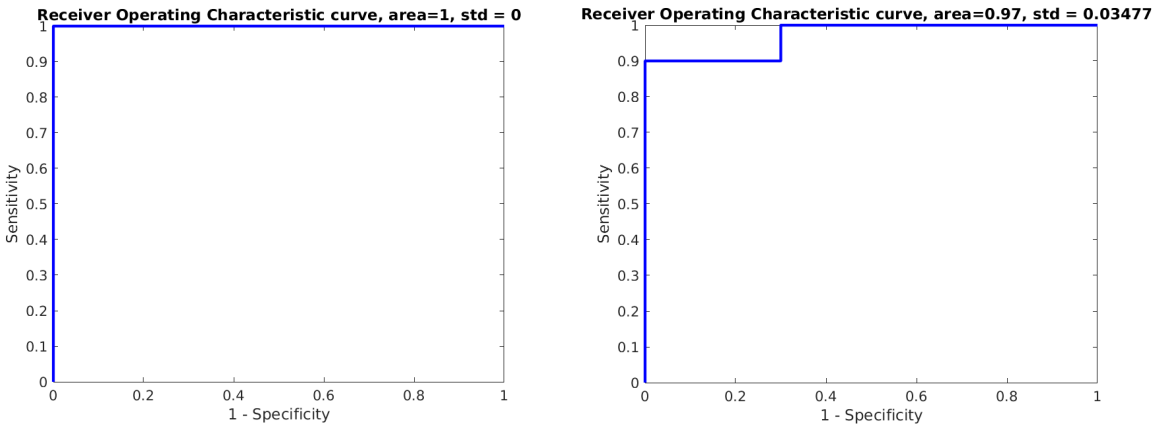


Figure 10: *(left)* ROC curves evaluated on the test set using tuned parameters from NM-simplex *(left)* and grid search *(right)*.

3.4 Bayesian Framework

The colour map indicates the probability that the instances belong a certain class. Figures 11 and 12 show the results of `bay_modoutClass` with different values of sigma and gamma, respectively. Figure 11 indicates that as sigma increases, the region of influence of each point increases, as per Equation 9. While 12 indicates that increasing gamma increases the smoothness of the surface and decreases certainty in classification away from the points.

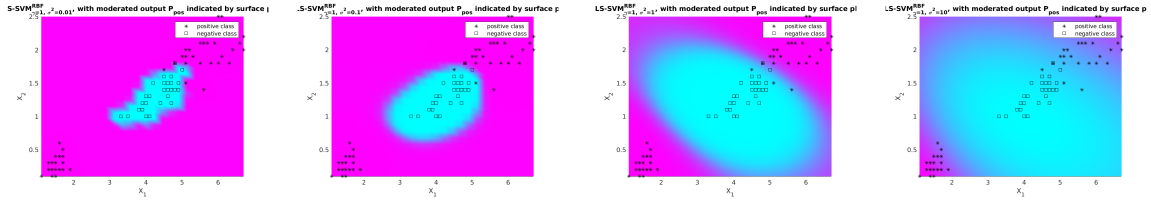


Figure 11: Colour maps indicating probability of class membership, with sigma at .001, .01, 1, and 10 (from left to right). Gamma is fixed at a value of 1.

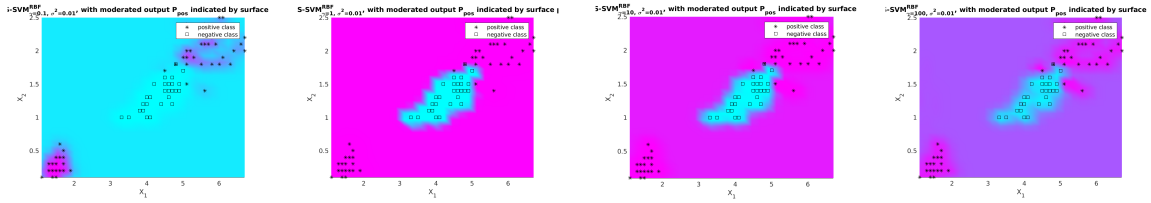


Figure 12: Colour maps indicating probability of class membership, with gamma at values of .01, 1, 10, and 100 (from left to right). Sigma is fixed at a value of 0.01.

4 Homework Problems

4.1 Ripley Data

This data can be easily visualised as it is two dimensional (see Figure 13). from this plot is is clear that x_2 is the more importance variable for classification, although x_1 is clearly still relevant. The classes are not linearly or near-linearly separable, so polynomial or RBF kernels should perform best. It is interesting that this dataset has 250 training instances and 1000 testing instances. To me, this seems like an inefficient use of data. We should instead be using at least some of this testing data to better train the model.

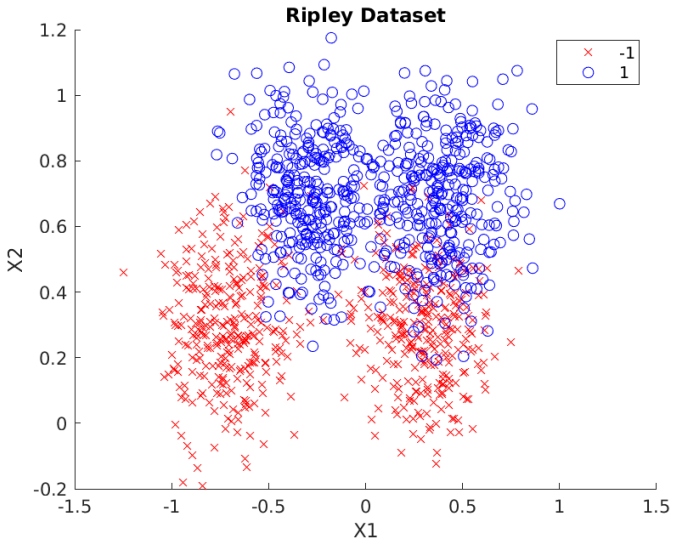


Figure 13: Scatter plot with class labels for the Ripley dataset.

The hyperparameters and kernel parameters for the linear, polynomial, and RBF models were tuned using `tunelssvm`. The resulting classifiers are visualised in Figure 14 and the ROC curves were computed for each trained model (see Figure 15). The area under the curve for each model was 0.958, 0.96, and 0.955, for linear, polynomial, and RBF, respectively. It was surprising to me that the linear model performed as well as the polynomial kernel, and better than the RBF kernel, given my initial thoughts about this data.
Are you satisfied with the performance of your model? Would you advise another methodology?

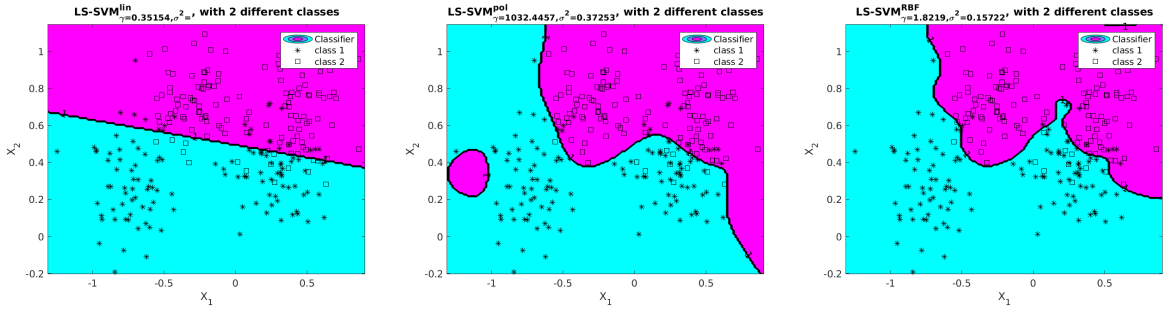


Figure 14: Classifiers for the linear, polynomial, and RBF kernels, from left to right.

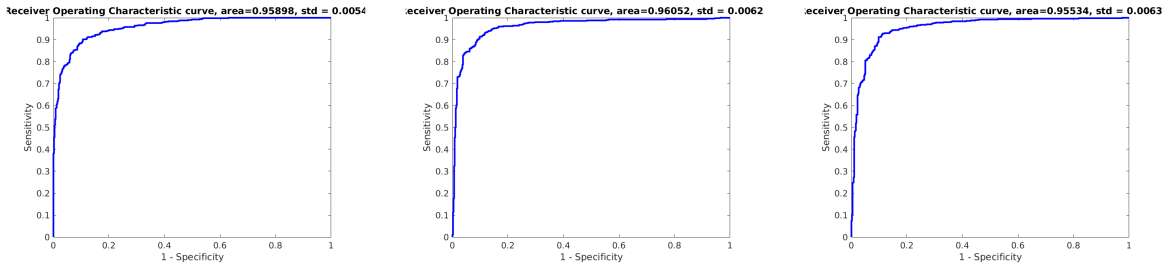
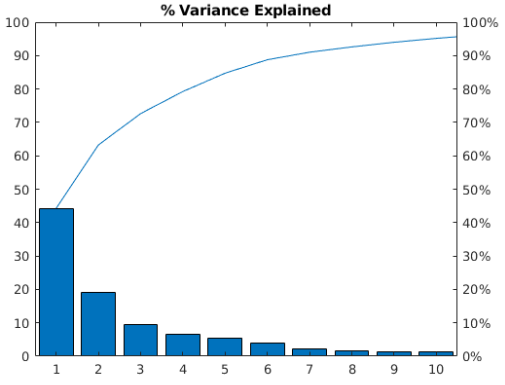


Figure 15: ROC curves for linear, polynomial, and RBF kernels, using 10-fold CV and tuned by NM-simplex.

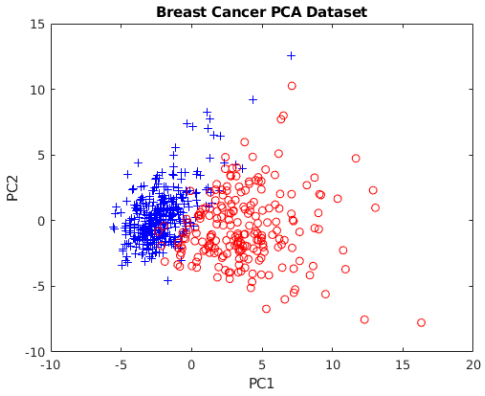
I am somewhat satisfied with the performance of the model. On the `esat.kuleuven.be` LSSVM tutorial, they achieve an AUC of 0.966, which is better than any of my models given above. I cannot think of another methodology to use, other than to change the amount of training and test data, as stated in the beginning.

5 Breast Cancer Data

The breast cancer data has 30 variables, and therefore the full dataset cannot be easily visualised. to understand the data, I first performed a PCA on the normalised data and viewed the distribution of total variance using a Pareto plot (see Figure 16a). Over 60% of the variability in the dataset is contained in the first two PCs. Plotting the transformed data in along PC1 and PC2, it is clear that the data is well-clustered and nearly linearly separable. Based on this, I think a linear model should be sufficient.



(a) Pareto plot of PCs.



(b) Scatter plot using the first two PCs.

SVMs with linear, polynomial, and RBF kernels were trained on the data using 10-fold cross validation and parameter tuning by NM-simplex. ROC curves were calculated on the test set for each model, giving AUCs of 0.994, 0.685, and 0.996, for linear, polynomial, and RBF, respectively (see Figure 17). Based on this, I would choose the linear model. It performs almost as well as the RBF kernel while being less complex and more

interpretable. When building models for medical data, interpretability and explainability is very important. We need to explain why the model gives a certain prediction. I would therefore choose the linear model in this context. The cost is also lower for the linear model, with fewer false negatives. I am satisfied with this model and would not suggest another methodology, though logistic regression could be interesting.

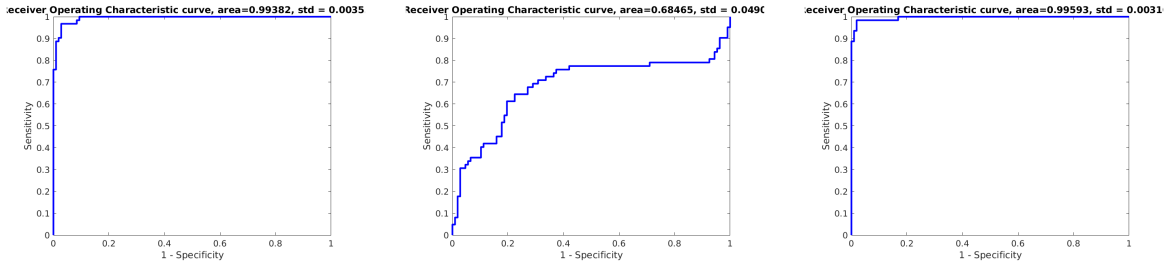
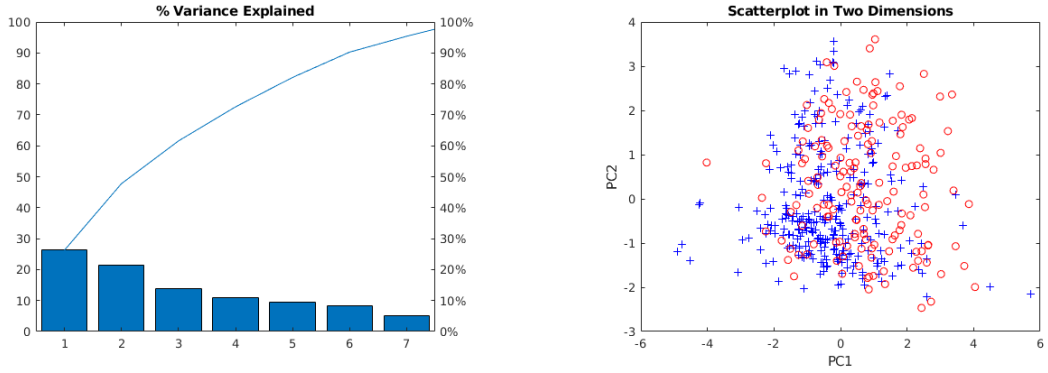


Figure 17: ROC curves for linear, polynomial, and RBF kernels (from left to right), using 10-fold cross validation parameters tuned by NM-simplex.

6 Diabetes Data

The diabetes dataset has 8 variables. Figure 18a shows that the variability present in the dataset is not concentrated on a small number of principle components, and the 95% threshold is reached only after the 7th PC. The adjacent scatter plot for the top two PCs does not show any separation between the two classes, though PC1 is clearly more informative. My intuition is that a linear model will perform poorly on this dataset, and that the RBF kernel method is most appropriate.



(a) Pareto plot of PCs.

(b) Scatter plot using the first two PCs.

Figure 19 shows the ROC curves for the three models, with AUCs of 0.84, 0.8, and 0.82 for linear, polynomial, and RBF respectively. I am surprised that the linear model outperforms the RBF kernel, given how the data appears to be structured into two overlapping clouds, with no clear linear separability between the classes.

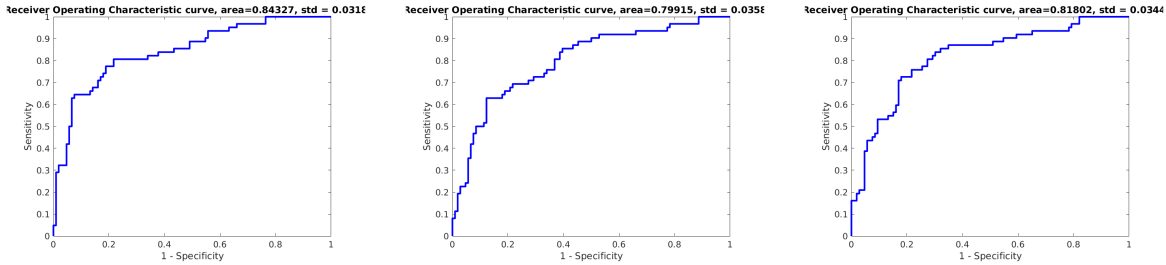


Figure 19: ROC curves for linear, polynomial, and RBF kernels (from left to right), using 10-fold cross validation parameters tuned by NM-simplex.

References

[1] S Sathiya Keerthi and Chih-Jen Lin. Asymptotic behaviors of support vector machines with gaussian kernel. *Neural computation*, 15(7):1667–1689, 2003.

[2] James Bergstra and Yoshua Bengio. Random search for hyper-parameter optimization. *Journal of machine learning research*, 13(2), 2012.

Function Estimation and Time Series Prediction

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

1 Function Estimation

Construct a dataset where a linear kernel is better than any other kernel (around 20 data points). What is the influence of e and of Bound. Where does the sparsity property come in?
The ϵ-insensitive loss function used to introduce sparsity of the support vectors, proposed by Vapnik, is given by:

$$|y - f(x)|_\epsilon = \begin{cases} 0 & , \quad \text{if } |y - f(x)| \leq \epsilon \\ |y - f(x)| - \epsilon & , \quad \text{otherwise} \end{cases} \tag{1}$$

Minimising a convex ϵ-insensitive loss function given above is equivalent to defining an ϵ-tube that contains the maximum possible number of training instances. The difference between the predicted values and the true values are minimised, and a penalty is applied to points that fall outside of the tube. The value of ϵ determines the width of the tube. A large ϵ value gives a higher tolerance for errors, a wider ϵ-tube, and fewer support vectors. The ϵ parameter controls sparsity by determining the number of support vectors. Conversely, a small ϵ value gives a lower tolerance for errors, a narrower ϵ-tube, and decreased sparsity, as training instances are outside the bounds of the ϵ-tube. Table 2 shows the results of a linear regression on the constructed dataset, varying both the bound parameter and ϵ. It is clear that increasing ϵ increases the width of the ϵ-tube. Table 1 demonstrates that sparsity increases with ϵ.

The *bound* parameter seems to have the same role as the gamma parameter in RBF kernels, where it is seen as the inverse of the radius of influence of instances selected by the model as support vectors. Similarly, the bound parameter here controls the range of influence of the instances. A lower bound parameter increases the range of influence for each instance, resulting in a flatter function with decreased flexibility (as seen in Table 2). Increasing the bound parameter decreases this range of influence, therefore increasing the flexibility of the function, and allowing for functions which are less flat. Naturally, reducing the flexibility of the function decreases the sparsity, as a greater number of instances fall outside the ϵ-tube, in this way, the bound parameter also controls the sparsity.

	$\epsilon = 0.1$				$\epsilon = 0.25$				$\epsilon = 0.5$			
bound	0.01	0.1	1	10	0.01	0.1	1	10	0.01	0.1	1	10
#SV	18	12	7	6	14	8	2	2	8	4	2	2

Table 1: Number of support vectors for each combination of ϵ and bound parameters.

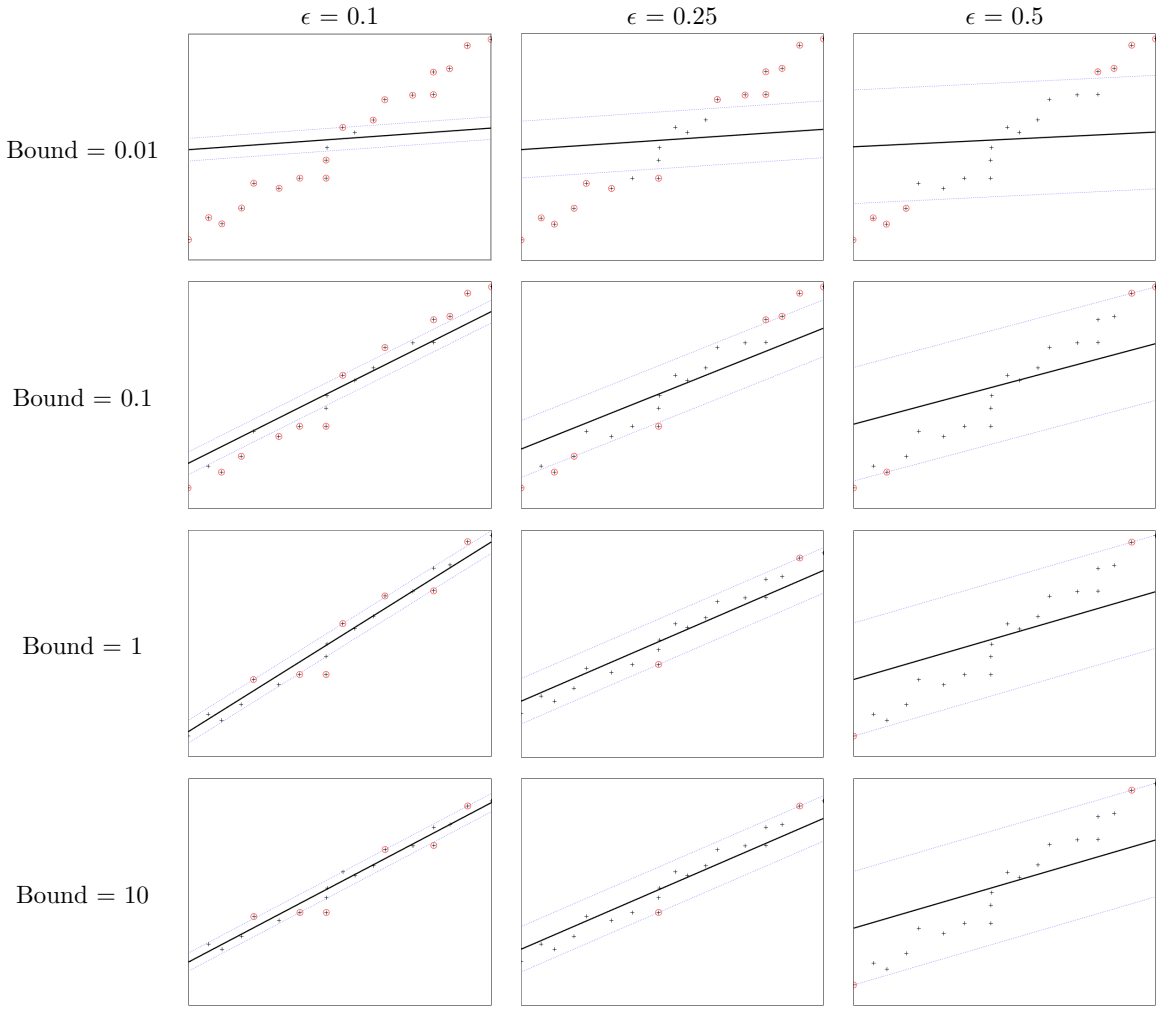


Table 2

Construct a more challenging dataset (around 20 data points). Which kernel is best suited for your dataset? Motivate why.

Figure 1 shows the results of fitting linear, polynomial, linear-bspline, and RBF kernels to the data. For each kernel I varied the parameters to try to get the best estimate from each kernel. Based on the results, I would suggest that either Gaussian RBF or polynomial are best suited to this dataset. Both the linear and linear-bspline kernels underfit the data. The Gaussian and polynomial kernels are better able to capture the non-linearity present in the data. The Gaussian RBF kernel has better sparsity properties than the polynomial kernel, with significantly fewer support vectors, and for that reason it would be my top choice. I would rank the models as follows: Gaussian RBF > Polynomial > Linear-BSpline > Linear.

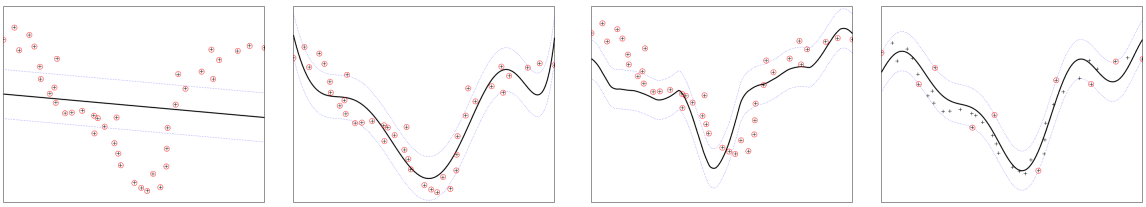


Figure 1: *(left to right)* Linear, polynomial, linear-bspline, and Gaussian RBF function estimation results on the more challenging dataset.

In what respect is SVM regression different from a classical least squares fit?
The primary difference between the two methods is clear in the cost function which they aim to minimise. Classical least squares regression uses the L2 squared loss function, which finds a line with minimum distances from the observations:

$$cost = \sum_{i=1}^{n} (y_i - f(x_i))^2 \tag{2}$$

Support vector regression with a linear kernel uses an ϵ-insensitive L1 loss function (see Equation 1), in which only the observation outside of the ϵ-tube contribute to the loss function. This gives support vector regression the ability to regularise and avoid overfitting, which is an advantage over classical least squares fit.

2 A simple example: the sinc function

2.1 Regression of the sinc function

Function estimation was performed on noised data from the sinc function. The estimation was performed using a RBF kernel and for a number of different sigma and gamma parameters. The results of the estimation are given in Table 3.

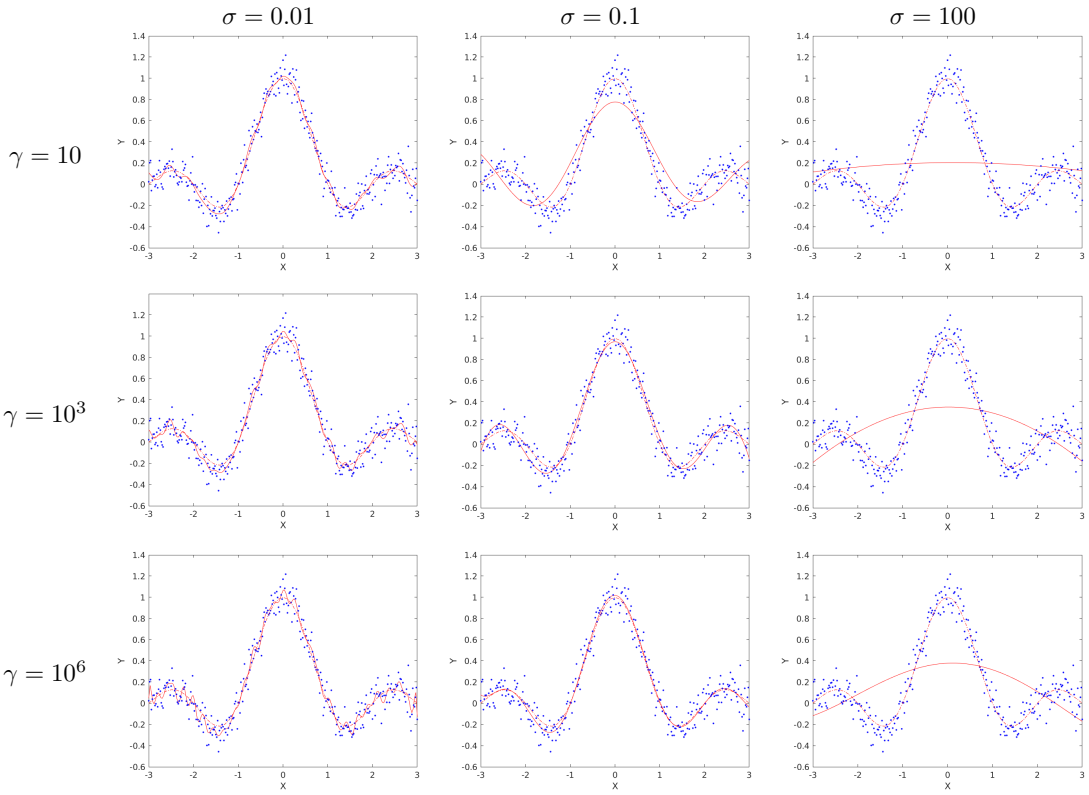


Table 3

From the resulting estimations, we can see the effects of gamma and sigma. Gamma is the regularisation parameter, determining the trade-off between the training error minimisation and the smoothness of the estimated function. This can be clearly seen in the leftmost column of the image grid. Increasing gamma prioritises training error minimisation, sacrificing function smoothness and possibly leading to some overfitting. Sigma is the kernel function parameter, and determines the radius of influence of each instance. Increasing sigma leads to a smoother function, with the function tending toward linear as $\sigma \longrightarrow \infty$. Both these parameters control the 'smoothness' of the estimated function, but at two different levels. The optimal combination of these parameters

finds the correct trade-off between training error minimisation and function smoothness (at both levels). The MSE for the estimated functions was calculated on the test set. The results are given in Table 4.

MSE	$\sigma = 0.01$	$\sigma = 0.1$	$\sigma = 100$
$\gamma = 10$	3.01	10.11	39.7
$\gamma = 10^3$	3.13	3.46	33.71
$\gamma = 10^6$	3.33	2.89	31.96

Table 4

Do you think there is one optimal pair of hyperparameters?
No. I think there is likely a number of optimal pairs which give very similar performances on the test set. To verify this, I performed a comprehensive grid search of the hyperparameter space, varying both gamma and sigma for the RBF kernel. The result is given in Figure 2, and shows that there are many combinations of hyperparameters which give very low MSE. I tried to search the parameter space for values of gamma greater than 10^{-12}, in the hopes of finding an increase in MSE once more, but the `trainlssvm` function started screaming at me so I stopped. The figure doesn't disprove the existence of a global minimum, although my opinion is that there is unlikely a single global optimum, at least to two significant digits.

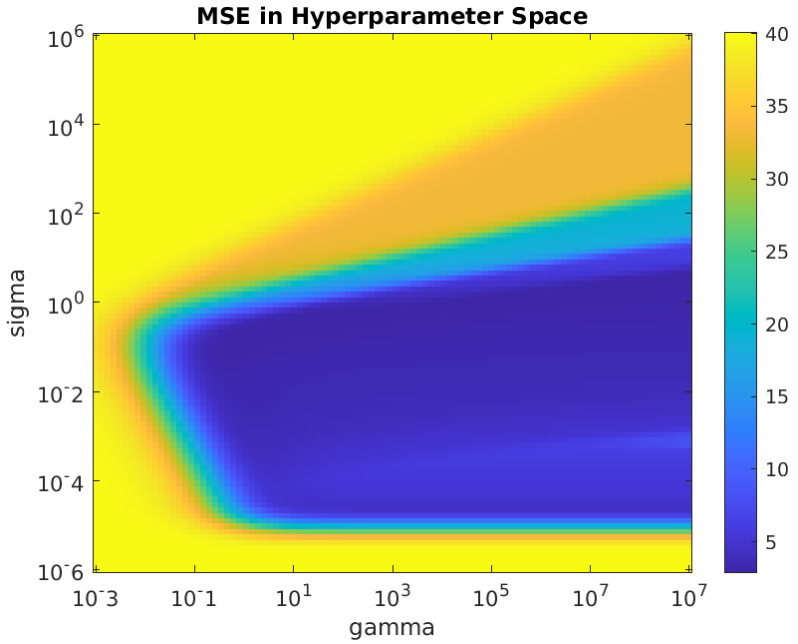


Figure 2: Colour plot showing MSE for a grid search of hyperparameter space.

The parameters were then tuned using the `tunelssvm` procedure, with both the NM-simplex and grid search algorithms. Both of the optimisation algorithms gave an MSE of 0.01 consistently over a number of runs. The gamma and sigma parameters varied between each run, validating that there is no one global optimum, but many local optima.

2.2 Application of the Bayesian framework

Discuss in a schematic way how parameter tuning works using the Bayesian framework. Illustrate this scheme by interpreting the function calls denoted above.
A schema for the Bayesian framework is given in Figure 3. Parameter tuning within the Bayesian framework is done by estimating the posterior probabilities on three different inference levels, as illustrated in the schema. The likelihood at a given level is equivalent to the evidence at the previous level.

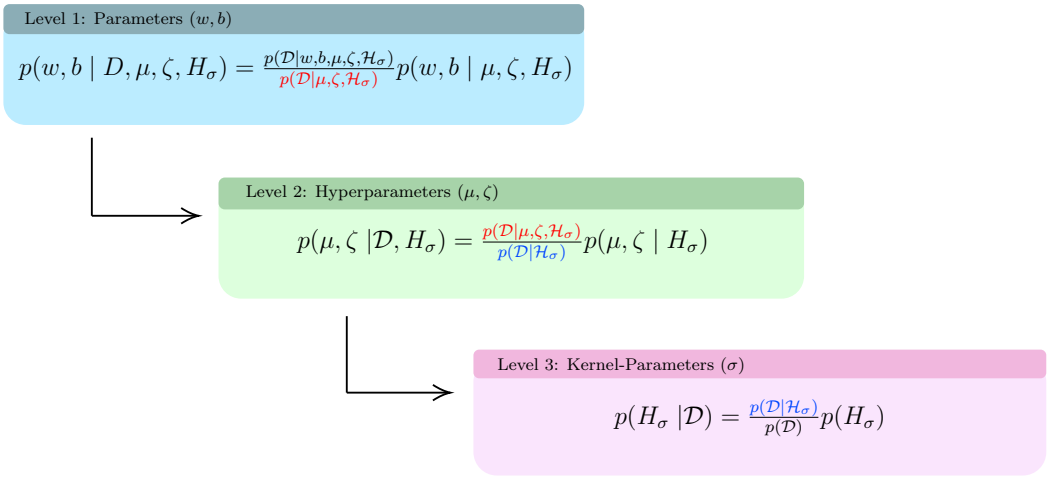


Figure 3: Schema of the multi-level Bayesian framework.

- **Level 1:** Inference of model parameters w and b occurs at the first level. This is done by finding w_{MP} and b_{MP} which maximise the logarithm of the posterior at level 1 (see schema).
- **Level 2:** Inference of hyperparameters μ and ζ which then form the hyperparameter $\gamma = \frac{\zeta}{\mu}$. The likelihood at this level $p(\mathcal{D}|\mu, \zeta, \mathcal{H}_\alpha)$ is equal to the evidence from level 1. Optimal μ and ζ are calculated by maximising the log posterior. Solving this optimisation problem requires computing the effective number of parameters d_{eff} from the eigenvalue decomposition of the Gram matrix G.
- **Level 3:** Inference of kernel parameters (α in the case of RBF kernel) and model comparison. The likelihood at this level $p(\mathcal{D}|\mathcal{H}_\alpha)$ is equal to the evidence from level 2.

The posterior probabilities of model parameters at the different inference levels are estimated using the `bay_lssvm` function. The cost at each level is computed by taking the negative logarithm of the posterior.

```
crit_L1 = bay_lssvm({Xtrain, Ytrain, 'f', gam, sig2}, 1);
crit_L2 = bay_lssvm({Xtrain, Ytrain, 'f', gam, sig2}, 2);
crit_L2 = bay_lssvm({Xtrain, Ytrain, 'f', gam, sig2}, 3);
```

Listing 1

The posterior probabilities of the model parameters and hyperparameters at different levels of inference are then optimised using `bay_optimize`.

```
[~, alpha, b] = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 1);
[~, gam]      = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 2);
[~, sig2]     = bay_optimize({Xtrain, Ytrain, 'f', gam, sig2}, 3);
```

Listing 2

3 Automatic Relevance Determination

Input selection can be done by Automatic Relevance Determination (ARD). This allows one to determine the most relevant inputs of an LS-LSVM within the Bayesian evidence framework. A different weighting parameter is assigned to each dimension in the kernel and this is then optimised using the third level of inference. Backward selection of inputs is performed by iteratively removing the input with the largest optimal sigma.

Automatic relevance determination is performed using the `bay_lssvmARD` function. This determines the relevant subset of variables, returns a ranked list of variable importance, and the costs associated with third level of inference in every selection step, which is an approximation of generalisation performance. This process can be visualised for a simple toy example with few dimensions (see Figure 4). Starting with a three input variables x_1, x_2, x_3, ARD selected x_1 as the most relevant subset. The ranked list indicated the ordering of importance: $x_1 > x_2 > x_3$, with the costs at each selection step given by -19, -63, and -75.

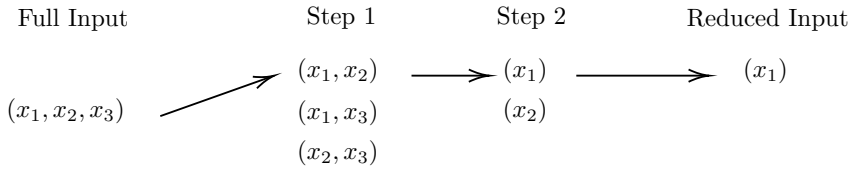


Figure 4: Illustration of ARD on the toy example.

How can you do input selection in a similar way using the `crossvalidate` *function instead of the Bayesian framework?*
Instead of determining the most relevant input variables by minimising the cost function in the Bayesian framework. One can perform forward or backward selection and evaluate the performance of the model at each step using cross validation. This will again give a ranking of variable importance, but it should be noted that these methods have their drawbacks and are not exhaustive (do not check every possible subset).

4 Robust Regression

Compare the non-robust version with the robust version. Do you spot any differences?
Figure 5 shows the function estimation of the non-robust and robust LS-SVM regression models on a dataset with outliers. The non-robust version is affected by the outliers while the robust version is not.

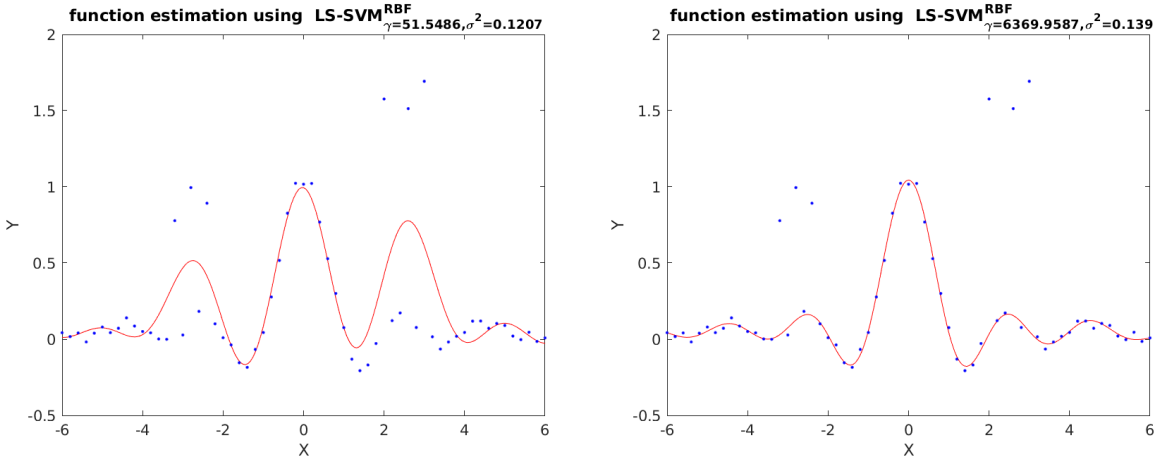


Figure 5: *(left)* Non-robust LS-SVM regression. The effect of the outliers is clearly seen skewing the function. *(right)* Robust regression, mitigating the effect of the outliers.

Why in this case is the mean absolute error preferred over the classical mean squared error?
Mean absolute error is less sensitive to outliers than mean squared error, which exaggerates the impact of outliers. As there are outliers present, MSE will give a poor performance for the robust model, which is an inaccurate reflection of its performance. MAE is more appropriate for comparing models in this case.

Try alternatives to the weighting function ***wFun****. Report on differences.*
The choice of weighting function in the cost function is very important for robust kernel regression. Brabanter et al conducted a comparison of different weighting schemes in the context of robust kernel regression [1]. They compared four different types of weight functions (Huber, Hampel, Logistic, and Myriad) and their use in iterative re-weighted LS-SVM. They discovered a trade-off between speed of convergence and the degree of robustness between weighting schemes. They concluded that the Myriad weight function is highly robust against (extreme) outliers but has a slow speed of convergence. A good compromise between speed of convergence and robustness can be achieved by using Logistic weights. The mean absolute error of each weighting function on the modified sinc dataset is given in Table 5.

	Huber	Myriad	Hampel	Logistic
MAE	0.154	0.141	0.146	0.143

Table 5: Mean absolute error for the different weighting functions.

5 Homework Problems

5.1 Logmap dataset

As indicated numerous times before, the parameters gam and sig2 can be optimised using cross-validation. In the same way, one can optimise order as a parameter. Define a strategy to tune these 3 parameters.
The sigma and gamma parameters were optimised using the `tunelssvm` function with 10-fold cross validation, simplex and an MAE cost function. The order was tuned simply by looping through order values from 1-100 and calculating the MAE on the prediction set using the tuned parameters. The results for the untuned and tuned parameters are given in Figure 6, along with a plot of MAE against order in Figure 7.

Do time series prediction using the optimised parameter settings. Visualise your results. Discuss.
The results show that the untuned model with an order of 10 does not have good prediction on the data, and only predicts well for a small portion. The tuned model has somewhat better performance, and accurately predicts in areas of low volatility. However, it fails to correctly predict the volatile movements. This is likely because these volatile movements are the result of noise, and the tuned model has learned the underlying non-noisy function. It is also interesting to note that prediction accuracy does not increase monotonically with order (see Figure 7). There is a 'sweet-spot' for order, with low prediction accuracies. Very low and very high orders give more volatile and worse prediction accuracies.

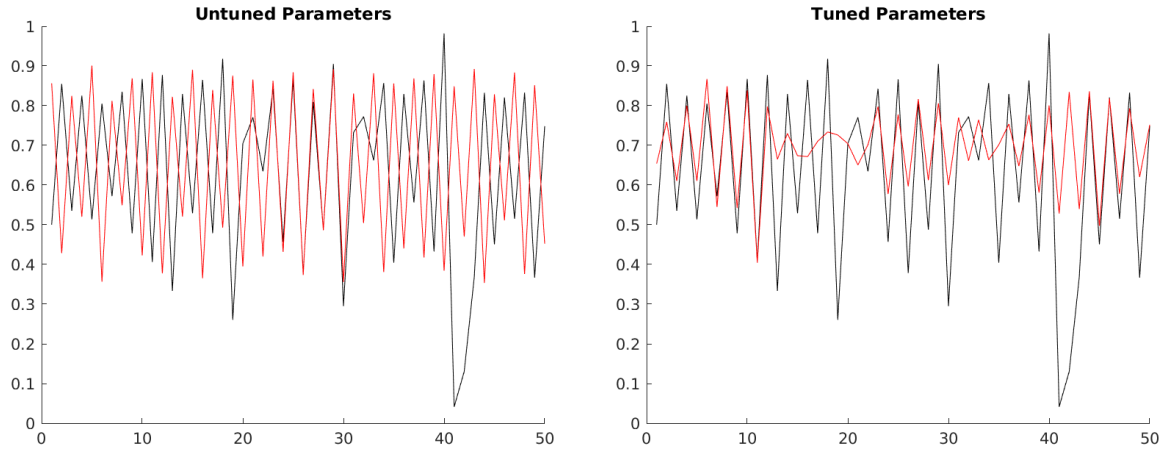


Figure 6: *(left)* Untuned logmap prediction results. Actual data is in black. Predicted data is in red. *(right)* Prediction results on logmap data using tuned model parameters and order = 23.

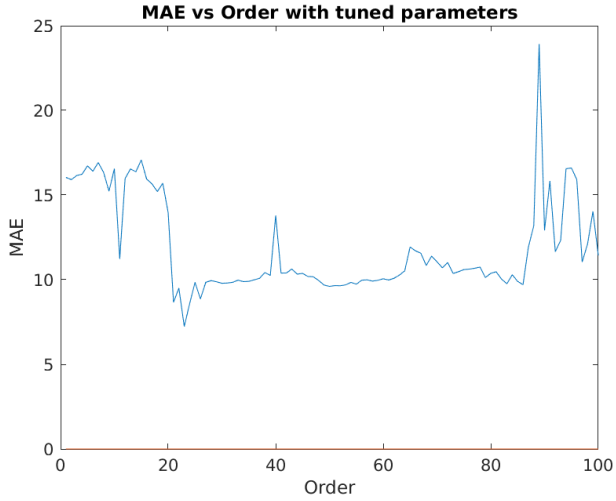


Figure 7: Mean absolute error plotted against order, using tuned hyperparameters for each order.

5.2 Santa Fe dataset

In this section I apply time series prediction on the Santa Fe laser dataset. The concatenated training and test data is given in Figure 8.

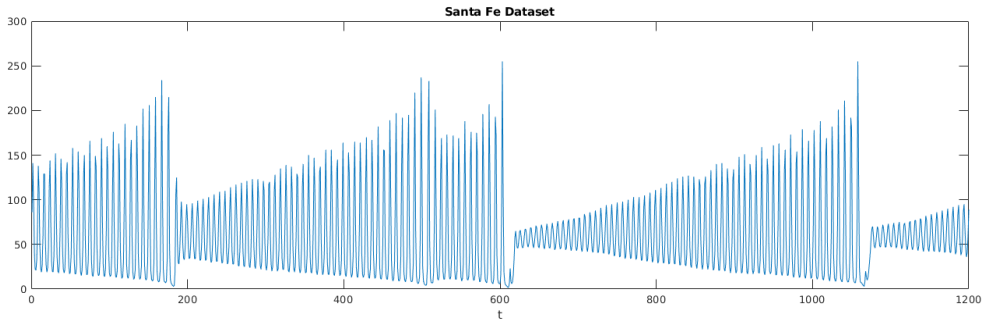


Figure 8: Visualising the Santa Fe dataset.

Does order = 50 for the utilised auto-regressive model seem like a good choice?
Looking at the plot given above, I would intuit that an order of 50 would be a good choice. It is sufficiently large to span the spike-crash-recover behaviour which is present. The model should therefore be able to learn this pattern and determine at what point it is going to occur.

Would it be sensible to use the performance of this recurrent prediction on the validation set to optimise hyperparameters and the model order?
There is no distinct validation set present. If one were to do a training-validation-test split of the data, I would think it sensible to use the validation set to optimise model hyperparameters and model order.

Tune the parameters (order, gam and sig2) and do time series prediction. Visualise your results. Discuss.
The parameters were tuned using the same strategy employed for the logmap dataset, tuning the hyperparameters for a given range of orders (0-100), using cross-validation, simplex and a mean absolute error cost function. The mean absolute error and of prediction was then calculated on the test set, and the order with the smallest MAE was determined to be the best. At orders less than 20 the models performed very poorly, with very high MAEs. It is interesting again that prediction accuracy doesn't increase with order. Being honest, I cannot figure out why exactly this is the case. The prediction is clearly not perfect, but it captures the qualitative behaviour of the data (spike-drop-recover) nonetheless.

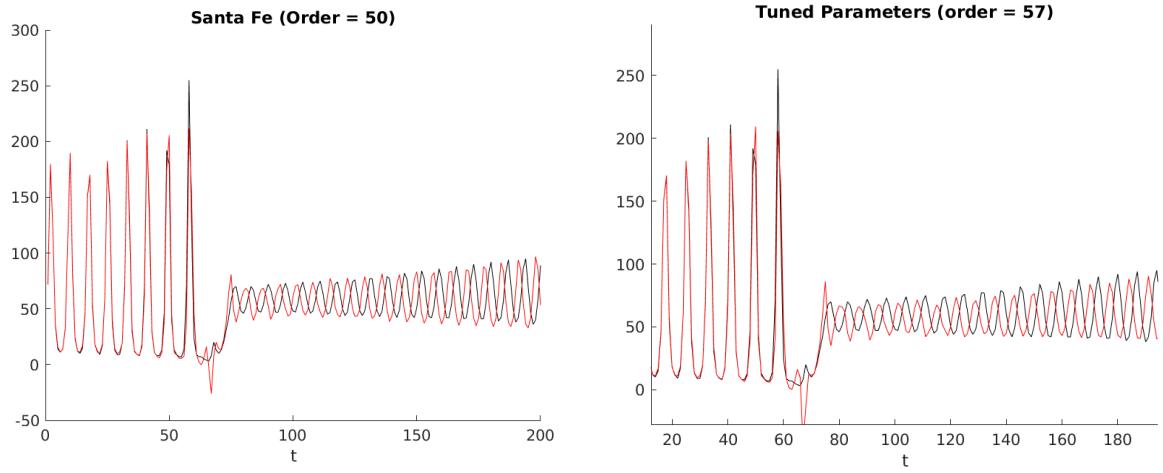


Figure 9: *(left)* Tuned Santa Fe prediction results with order = 50. Actual data is in black. Predicted data is in red. *(right)* Prediction results on logmap data using tuned model parameters and order (determined by MAE). The order of the best performing model was 57 (see Figure 10).

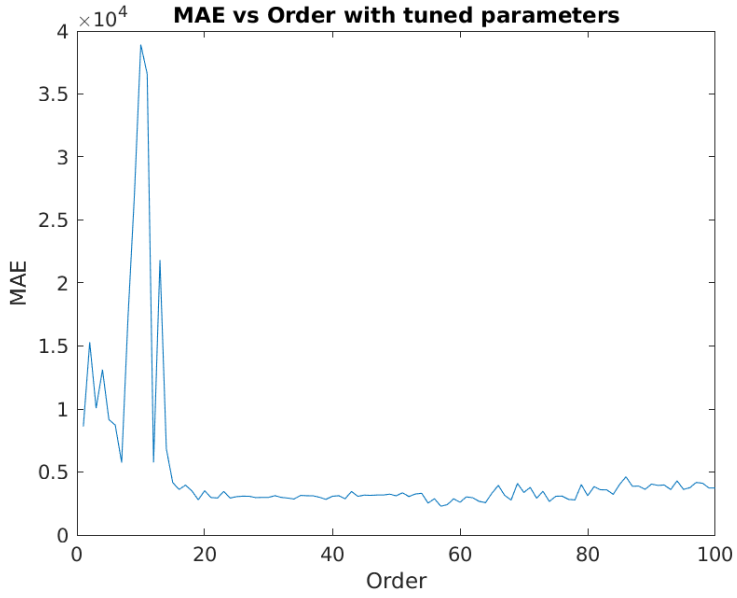


Figure 10: MAE of tuned Santa Fe prediction results plotted against order. Note the drop in MAE at around order = 20.

References

[1] Kris De Brabanter, Kristiaan Pelckmans, Jos De Brabanter, Michiel Debruyne, Johan AK Suykens, Mia Hubert, and Bart De Moor. Robustness of kernel based regression: a comparison of iterative weighting schemes. In *International conference on artificial neural networks*, pages 100–110. Springer, 2009.

Unsupervised Learning and Large Scale Problems

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

1 Kernel Principal Component Analysis

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

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

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

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

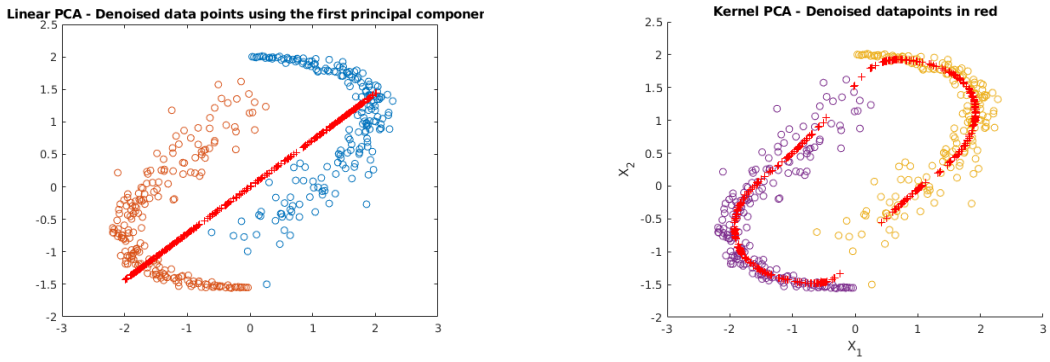


Figure 1: *(left)* Standard on the yin-yang dataset. *(right)* Kernel PCA on the yin yang dataset with 6 principal components and $\sigma = 0.4$.

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

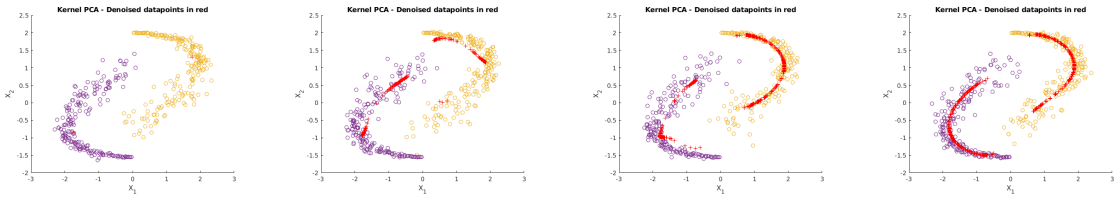


Figure 2: kPCA with 1, 2, 3, and 4 principal components (from left to right).

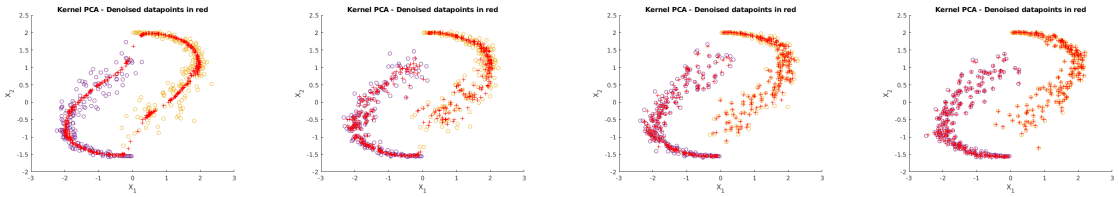


Figure 3: kPCA with 10, 15, 20, and 30 principal components (from left to right).

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

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

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

As a reference, I used Fukumizu and Alam's paper *"Hyperparameter Selection in Kernel Principal Component Analysis"*, in which they outline the difficulties of measuring reconstruction error for kPCA and discuss the

method given above for choosing hyperparameters, optimal kernel and the number of kernel principal components using cross-validation on the reconstruction errors of pre-images [4].

2 Spectral Clustering

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

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

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

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

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

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

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

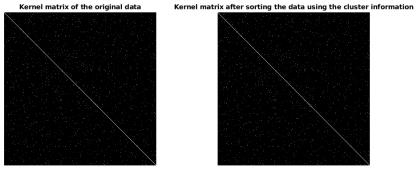
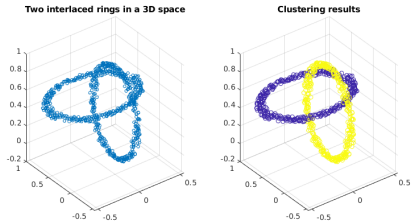
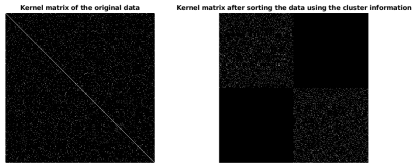
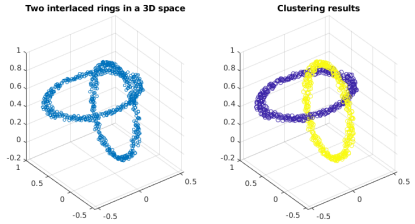
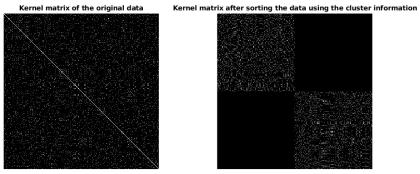
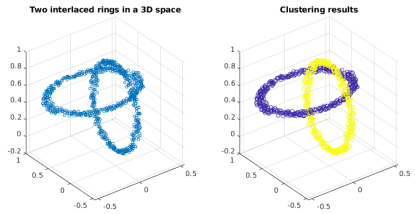
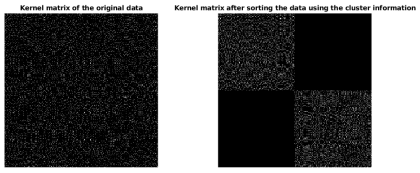
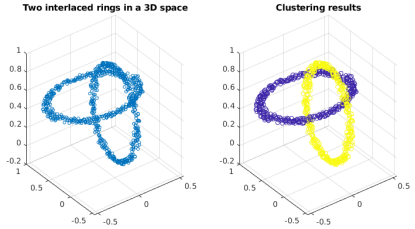
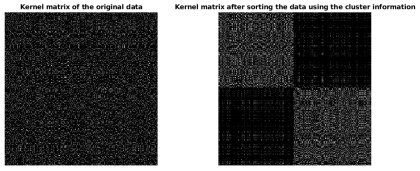
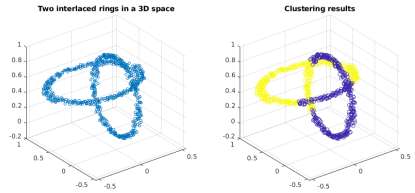
σ	Kernel Matrices	Clustering Results
$\sigma = 0.001$		
$\sigma = 0.005$		
$\sigma = 0.01$		
$\sigma = 0.02$		
$\sigma = 0.05$		

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

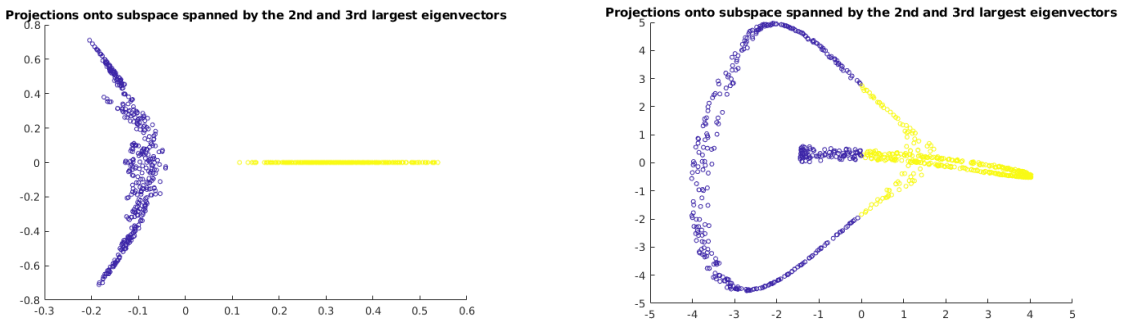


Figure 4: *(left)* Projection for $\sigma = 0.001$ *(right)* Projection for $\sigma = 0.05$

3 Fixed-size LS-SVM

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

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

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

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

Using the mapping

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

and the kernel function

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

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

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

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

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

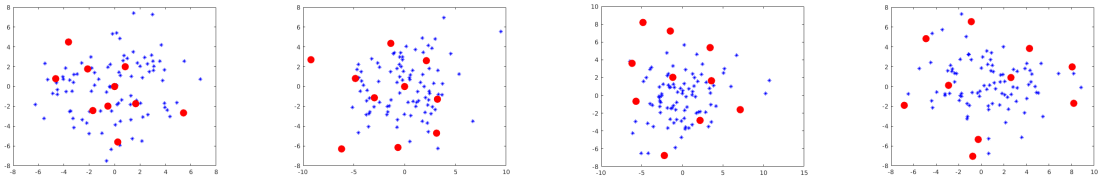


Figure 5: Determining support vector subsets of size 10, with $\sigma = 0.01$, 0.1, 1, and 10 (from left to right).

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

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

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

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

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

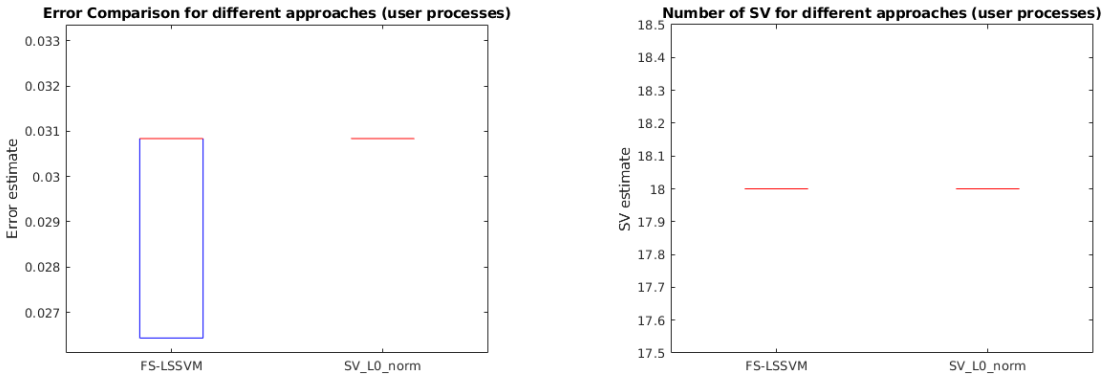


Figure 6: *(left)* Error comparison for FS-LSSVM and ℓ_0-approximation. *(right)* Number of support vectors.

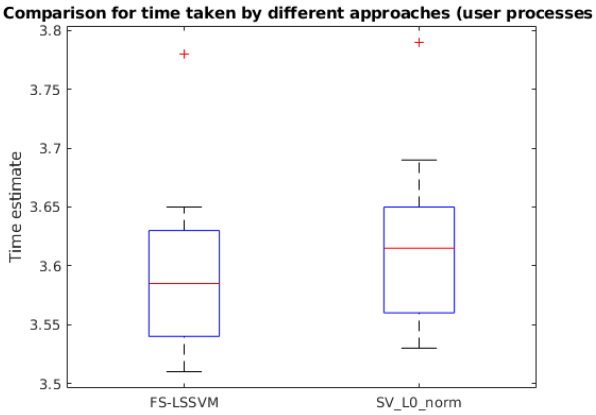


Figure 7: Time estimate for FS-LSSVM and ℓ_0-approximation

4 Homework Problems

4.1 Kernel Principal Components Analysis

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

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

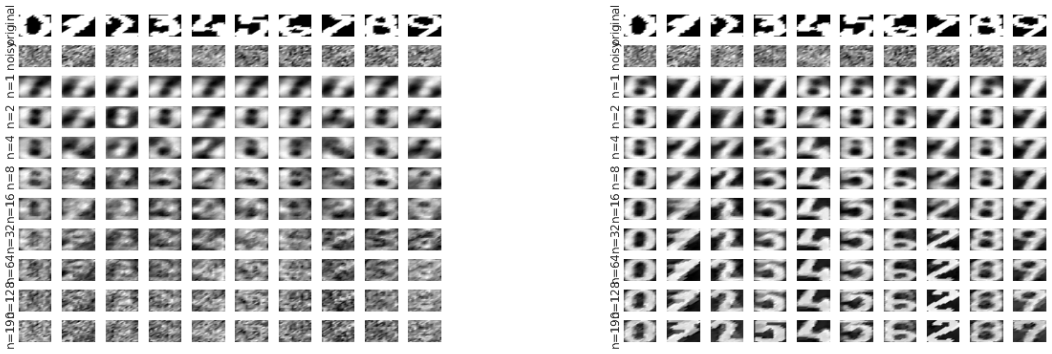


Figure 8: Denoising with PCA *(left)* and kPCA *(right)* with a noise factor of 1.

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

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

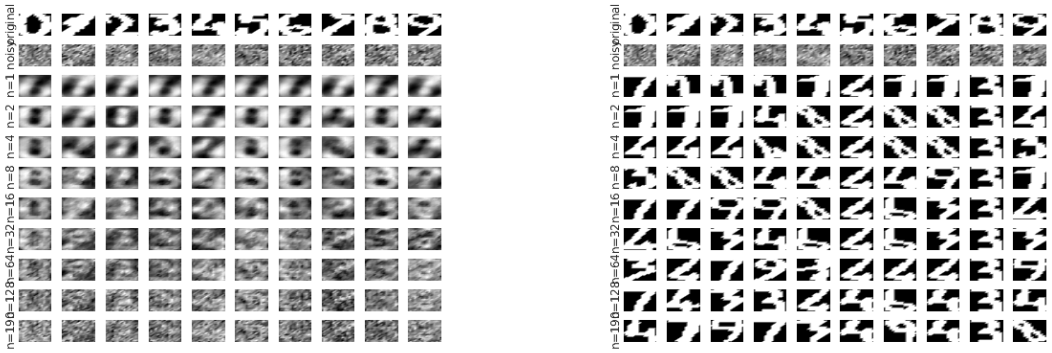


Figure 9: Denoising with PCA *(left)* and kPCA *(right)* with a sigma factor of 0.01.

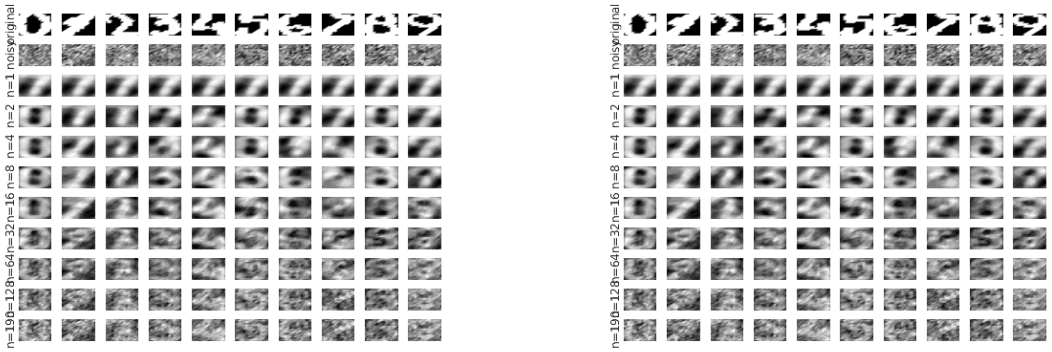


Figure 10: Denoising with PCA *(left)* and kPCA *(right)* with a sigma factor of 10.

4.2 Fixed-size LS-SVM

4.2.1 Shuttle

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

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

4.2.2 California

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

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

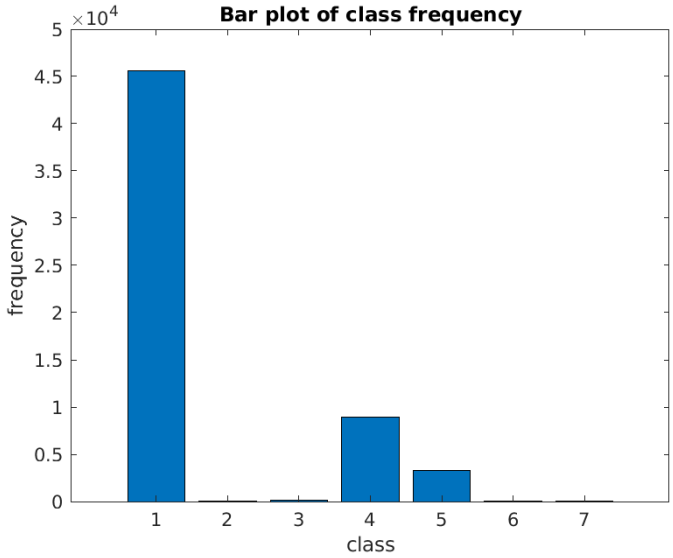


Figure 11: Bar plot showing class frequency for the shuttle dataset.

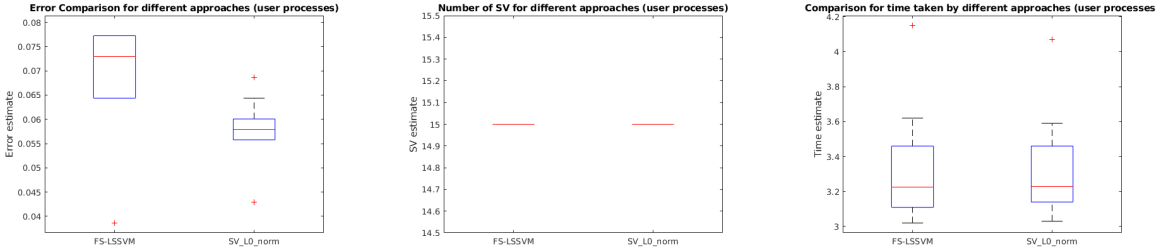


Figure 12: *(left)* Error comparison for FS-LSSVM and ℓ_0-approximation. *(middle)* Number of support vectors. *(right)* Computational cost.

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

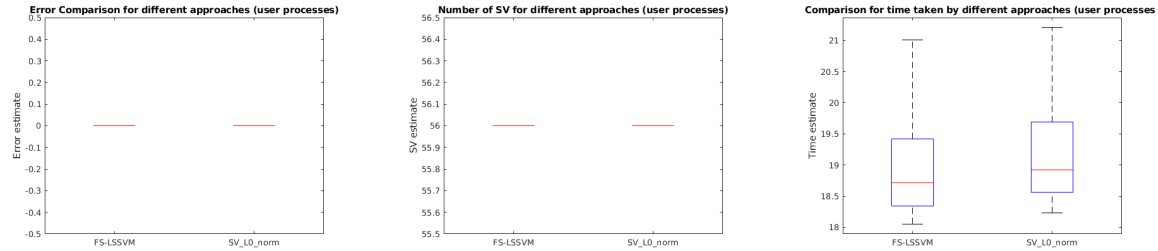


Figure 13: *(left)* Error comparison for FS-LSSVM and ℓ_0-approximation. *(middle)* Number of support vectors. *(right)* Computational cost.

References

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

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

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

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

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

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

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