EFC3 - Exercise 3

Rafael Claro Ito (R.A.: 118430)

November 2019

1 Source files

All code cited and all figures showed here can be found at the following GitHub repository:
`https://github.com/ito-rafael/IA006C-MachineLearning/tree/master/efc3`
In this repository, one can found the following files:

- Jupyter Notebook
  - MLP.ipynb
  - SVM.ipynb
- LaTeX
  - efc3.tex

The notebook "MLP" implements a multi-layer perceptron network used for binary classification. Here, we use different numbers of neurons, plot the decision regions and also calculate some metrics to evaluate the overall performance.

The notebook "SVM" uses the same dataset and also implements a binary classifier, but it does this using a support vector machine. Here, we plot the decision regions as well, but this time we vary the value of the hyperparameter C and also the value gamma used in a RBF kernel.

2 Part I - Error backpropagation

$$u_1 = 1 \cdot v_{00} + x_1 \cdot v_{10} + x_2 \cdot v_{20}$$
$$u_2 = 1 \cdot v_{01} + x_1 \cdot v_{11} + x_2 \cdot v_{21}$$
$$u_3 = 1 \cdot v_{02} + x_1 \cdot v_{12} + x_2 \cdot v_{22}$$

$$s_1 = f(u_1)$$
$$s_2 = f(u_2)$$
$$s_3 = f(u_3)$$

$$y_1 = 1 \cdot w_{00} + s_1 \cdot w_{10} + s_2 \cdot w_{20} + s_3 \cdot w_{30}$$
$$y_2 = 1 \cdot w_{01} + s_1 \cdot w_{11} + s_2 \cdot w_{21} + s_3 \cdot w_{31}$$

$$J = e_1^2 + e_2^2$$

$$\delta_3 = \frac{\partial J}{\partial u_3} = \frac{\partial[(d_1 - y_1)^2 + (d_2 - y_2)^2]}{\partial u_3}$$
$$\delta_3 = \frac{\partial (d_1 - y_1)^2}{\partial u_3} + \frac{\partial (d_2 - y_2)^2}{\partial u_3}$$
$$\delta_3 = \frac{\partial (d_1 - y_1)^2}{\partial (d_1 - y_1)} \cdot \frac{\partial (d_1 - y_1)}{\partial u_3} + \frac{\partial (d_2 - y_2)^2}{\partial (d_2 - y_2)} \cdot \frac{\partial (d_2 - y_2)}{\partial u_3}$$
$$\delta_3 = 2(d_1 - y_1) \cdot \left(-\frac{\partial y_1}{\partial u_3}\right) + 2(d_2 - y_2) \cdot \left(-\frac{\partial y_2}{\partial u_3}\right)$$
$$\delta_3 = -2(d_1 - y_1) \cdot \frac{\partial y_1}{\partial s_3} \cdot \frac{\partial s_3}{\partial u_3} - 2(d_2 - y_2) \cdot \frac{\partial y_2}{\partial s_3} \cdot \frac{\partial s_3}{\partial u_3}$$

$$\delta_3 = -2(d_1 - y_1) \cdot \frac{\partial (1 \cdot w_{00} + s_1 \cdot w_{10} + s_2 \cdot w_{20} + s_3 \cdot w_{30})}{\partial s_3} \cdot \frac{\partial f(u_3)}{\partial u_3} - 2(d_2 - y_2) \cdot \frac{\partial (1 \cdot w_{01} + s_1 \cdot w_{11} + s_2 \cdot w_{21} + s_3 \cdot w_{31})}{\partial s_3} \cdot \frac{\partial f(u_3)}{\partial u_3}$$

$$\delta_3 = -2(d_1 - y_1)\dot{f}(u_3)w_{30} - 2(d_2 - y_2)\dot{f}(u_3)w_{31}$$

$$\frac{\partial J}{\partial v_{12}} = \frac{\partial J}{\partial u_3} \cdot \frac{\partial u_3}{\partial v_{12}}$$
$$\frac{\partial J}{\partial v_{12}} = \frac{\delta_3 \cdot \partial(1 \cdot v_{02} + x_1 \cdot v_{12} + x_2 \cdot v_{22})}{\partial v_{12}}$$
$$\frac{\partial J}{\partial v_{12}} = \delta_3 \cdot x_1$$
$$\boxed{\frac{\partial J}{\partial v_{12}} = -2x_1\dot{f}(u_3)[w_{30}(d_1 - y_1) + w_{31}(d_2 - y_2)]}$$

3 Part II - Binary Classification with MLP and SVMs

3.1 Multi-layer Perceptron (MLP)

3.1.1 a) MLP network

The MLP network was implemented with the scikit-learn "MLPClassifier" class. This model optimizes the log-loss function. The parameters of the network were constantly modified in order to tune the classifier. The final configuration was:

Parameters:

- number of neurons: 50
- activation: relu
- solver: adam
- alpha: 0.01
- batch_size: 200
- random_state: 42
- tol: 1e-3
- max_iter: 1
- warm_start: True
- beta_1: 0.9
- beta_2: 0.95
- epsilon: 1e-8
- n_iter_no_change: 10

To determine the number of neurons, we trained the network with different number of neurons, choosing a reasonable value with a good generalization (not few neurons, avoiding an underfitting, not too many, avoiding overfitting, this will be clear in Figure 7). The batch size was chosen to be 200, which represents 20% of the training data, this means that for each epoch we would have 5 updates in the algorithm. The rectifier function was chosen to be the activation function, so each unit is what is called a ReLU. The solver used is adam, with parameters values of 0.9, 0.95 and 1e-8 for beta_1, beta_2 and epsilon, respectively. We are also using a seed to generate pseudo-random numbers, so the results showed here can all be exactly replicated. Another important thing to note is that the max_iter parameter is 1 because we are training the network iteration by iteration. This is done in order to plot the progression of the cost function during the training and during the validation. For this to work, we use warm_start as True. The tolerance used is 1e-3 and the alpha (regularization coefficient) is 0.01.

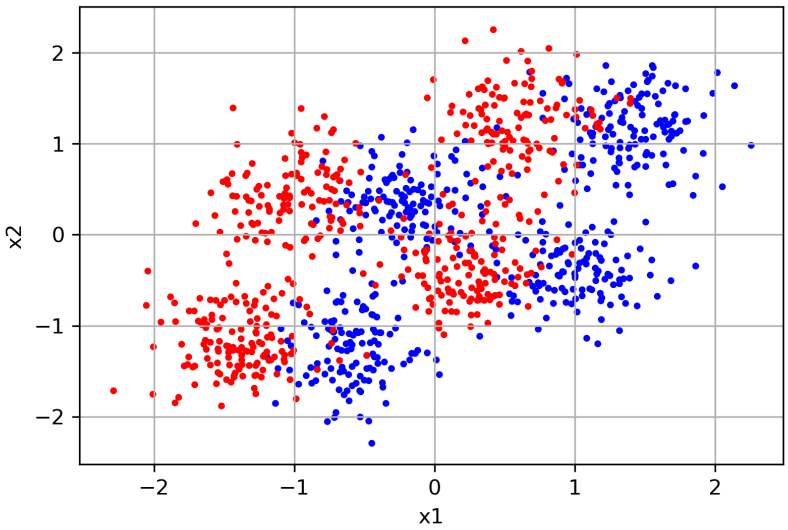


Figure 2: Training set

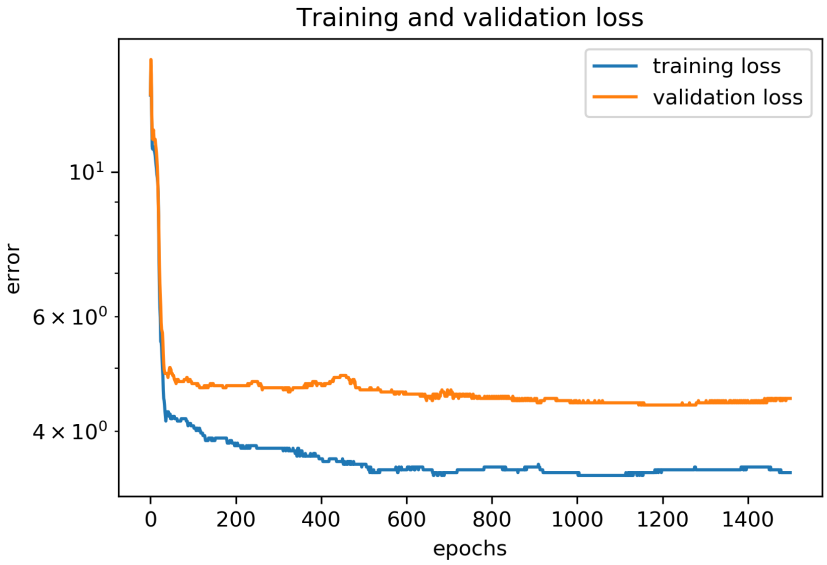


Figure 3: Progression of cost function in training and validation

The network was trained with the parameters showed previously. Here we explored the early stopping as a cross validation method along the validation set. The training data can be seen in Figure 2 and the progression of the cost function for training and validation can be seen in Figure 3.

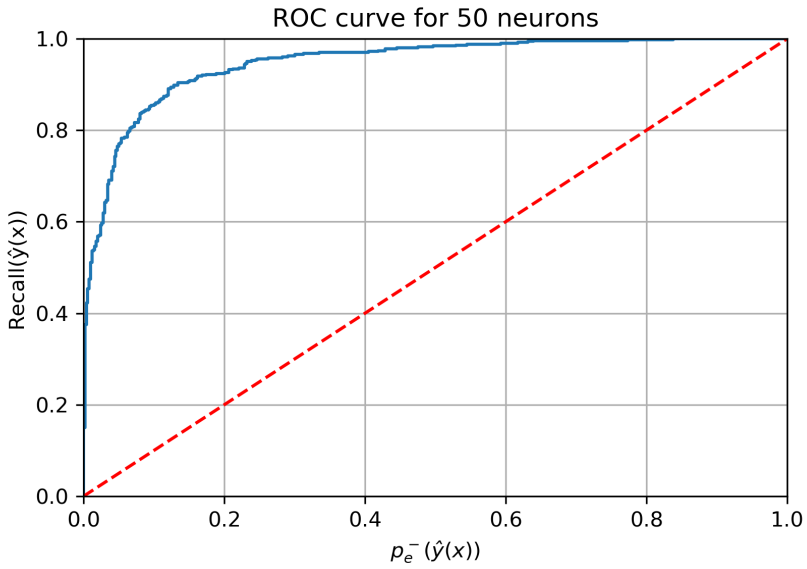


Figure 4: ROC curve for 50 neurons

In Figure 4 we can see the ROC curve for this classifier.

3.1.2 b) MLP decision regions

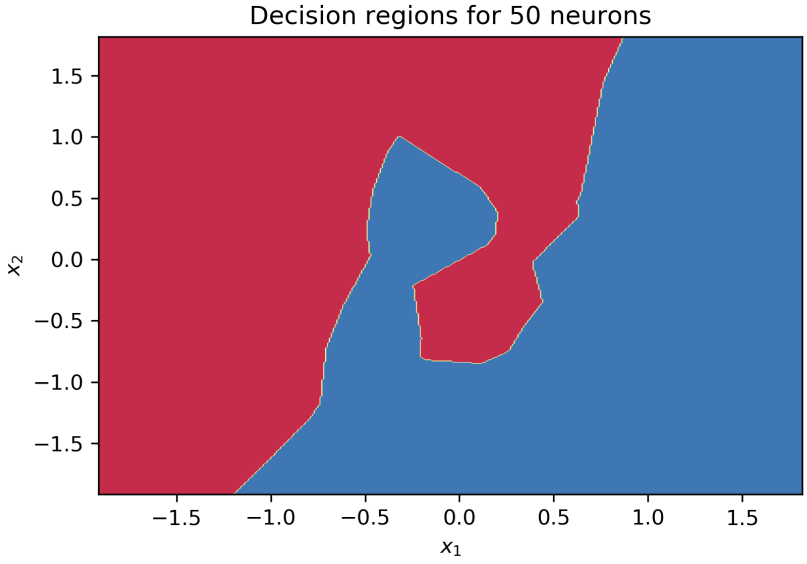


Figure 5: Decision regions for MLP with 50 neurons

A function was created to plot the decision regions. This function receives as parameters the dataset and the model. The dataset is used only to define the limits of the plot and the predict method of the model is used to determine the label of each division of the grid. The Figure 5 shows the decision regions for this MLP classifier.

3.1.3 c) Applying MLP model to test set

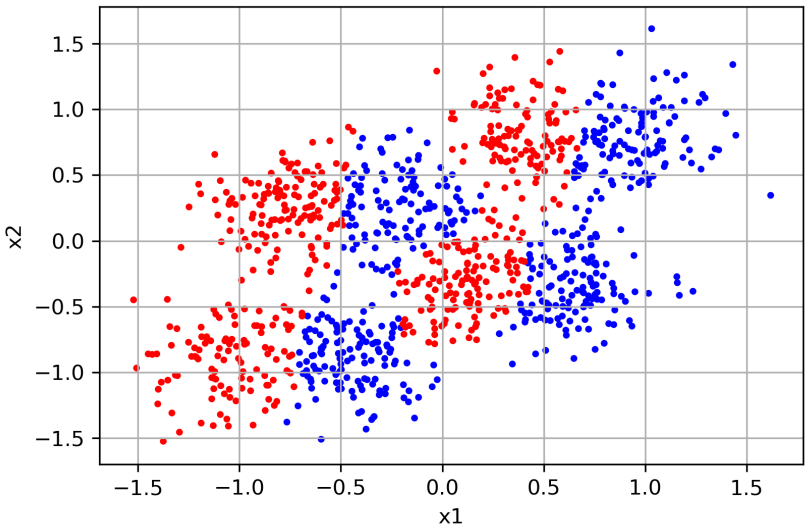
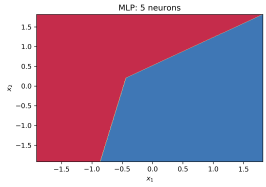


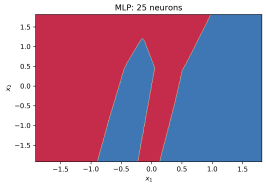
Figure 6: MLP predicted output for test set

We then applied the trained network in the test set. The output generated can be seen in Figure 6. The accuracy was 0.879, indicating an error of 12.1%.

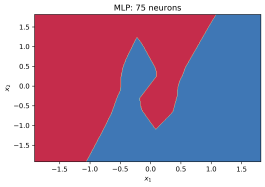
3.1.4 d) Experimenting network with different number of neurons



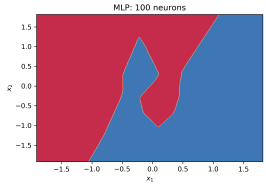
(a) 5 neurons



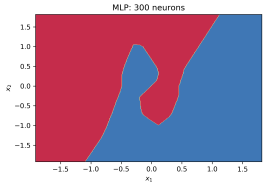
(b) 25 neurons



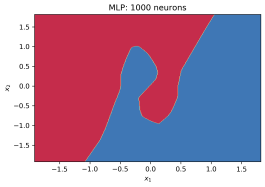
(c) 75 neurons



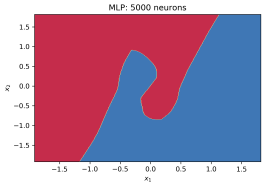
(d) 100 neurons



(e) 300 neurons

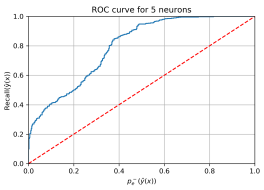


(f) 1000 neurons

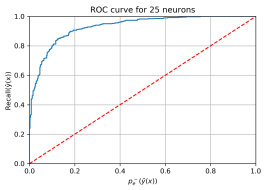


(g) 5000 neurons

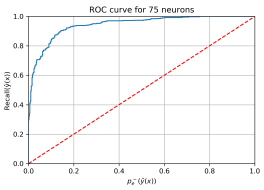
Figure 7: Decision regions for MLP with different number of neurons



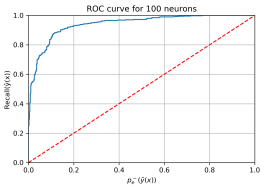
(a) 5 neurons



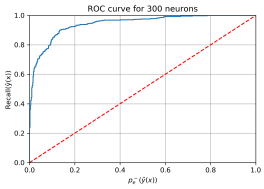
(b) 25 neurons



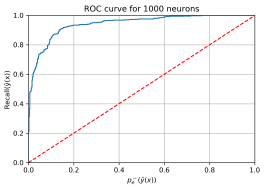
(c) 75 neurons



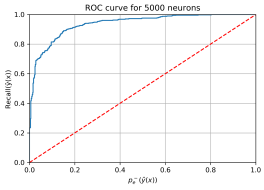
(d) 100 neurons



(e) 300 neurons



(f) 1000 neurons



(g) 5000 neurons

Figure 8: ROC curves for MLP with different number of neurons

We then tried networks with different number of neurons. 5, 25, 75, 100, 300, 1000 and 5000 neurons were utilized. Figure 7 shows the decisions regions for these classifiers and Figure 8 shows the ROC curves.

As expected, for networks with few neurons (5 or 25), the model does not have flexibility for separate the classes (underfitting). The opposite is also true, the network with 5000 neurons "learned" too much from the dataset compromising the generalization (overfitting). The networks with 75, 100 and 300 neurons presented satisfied results.

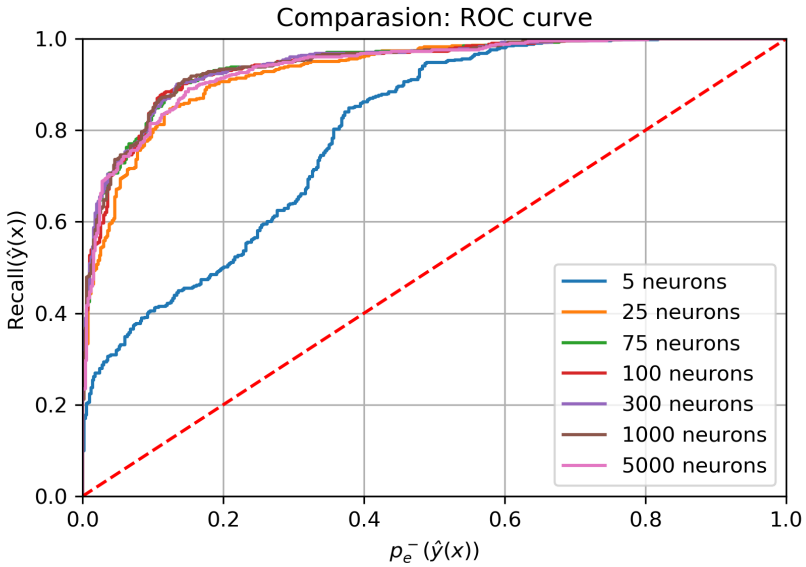


Figure 9: Comparison of ROC curves

In Figure 9 we plot all ROC curves under a single plot. This way we can compare the classifiers.

	Accuracy	F1-score	ROC area
5 neurons	0.733	0.762	0.7951
25 neurons	0.858	0.858	0.9305
50 neurons	0.881	0.881	0.9477
75 neurons	0.870	0.868	0.9430
100 neurons	0.870	0.866	0.9415
300 neurons	0.872	0.870	0.9434
1000 neurons	0.877	0.875	0.9427
5000 neurons	0.860	0.858	0.9376

Table 1: Comparison of results between networks with different number of neurons

We also present in Table 1 other metrics used to evaluate the classifiers. Accuracy in test set, F1-score and area under the ROC curve were calculated. Since the dataset is quite balanced, we are using the simple F1-score, instead of micro and macro.

3.2 Support Vector Machine (SVM)

3.2.1 e) SVM decision regions and support vectors

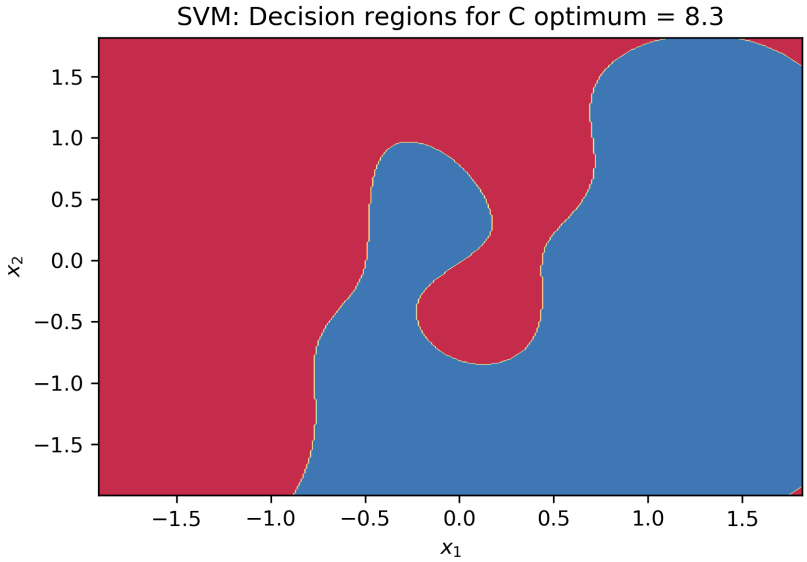


Figure 10: Decision regions

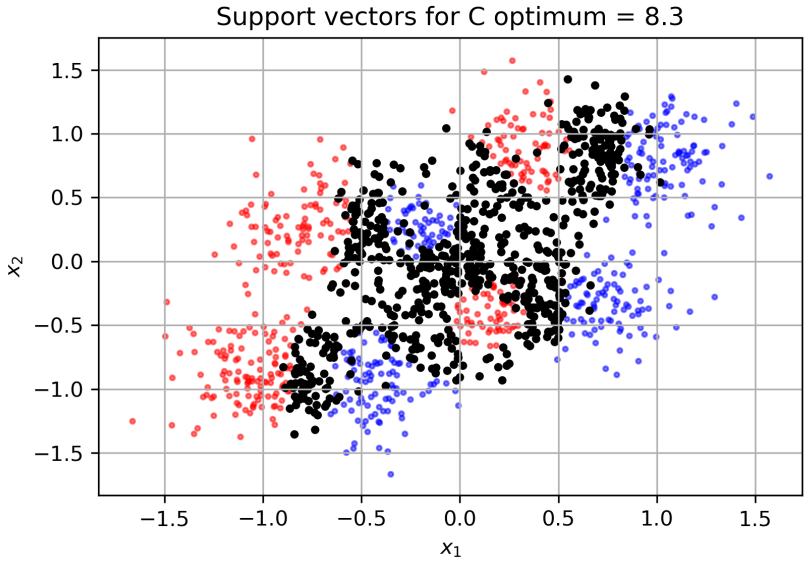


Figure 11: Support vectors

We now developed a SVM to do the classification. To find the hyperparameter C we used cross-validation sweeping C from 0.1 until 10 in steps of 0.1. The C for minimum error in validation set found was 8.3. The Figure 10 shows the decision regions for this value of C plotted with the same function used in the MLP. In Figure 11 we can see the support vectors identified.

3.2.2 f) Applying SVM model to test set

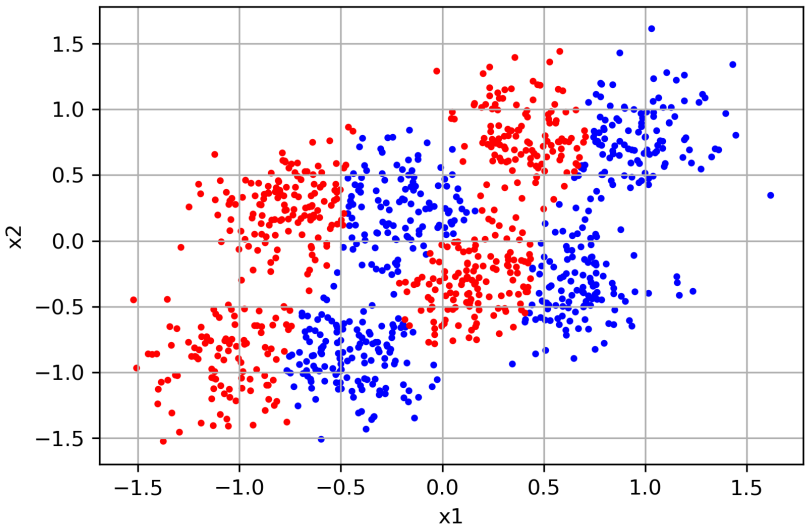
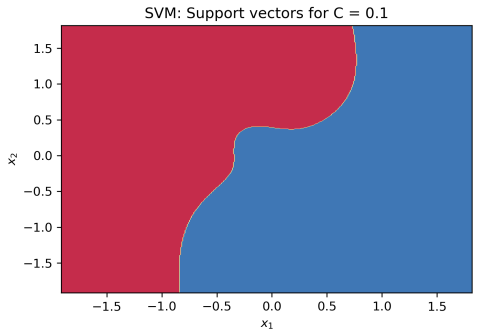


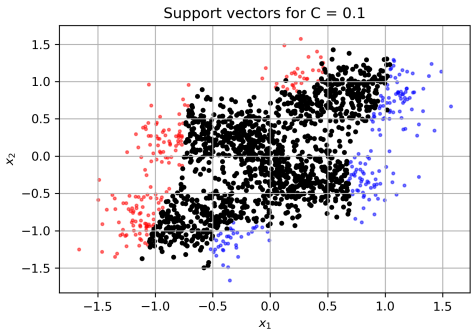
Figure 12: SVM predicted output for test set

We then used the model along with the test set. Figure 12 shows the output generated for each data. The accuracy obtained was 0.877, indicating an error of 12.3%.

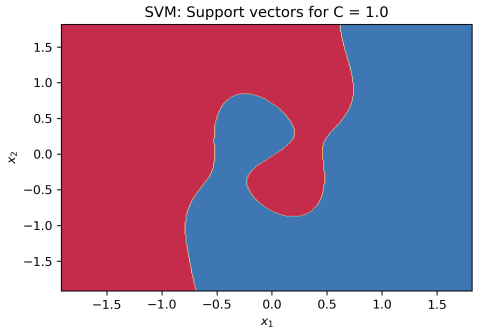
3.2.3 g) Experimenting different kernel parameters and values of C



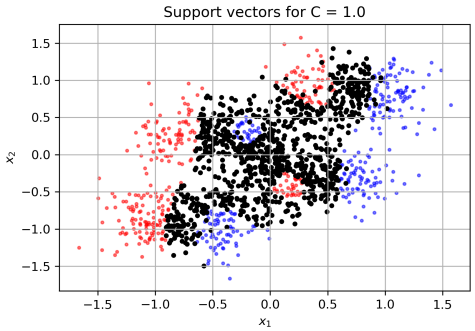
(a) Decision regions for C = 0.1



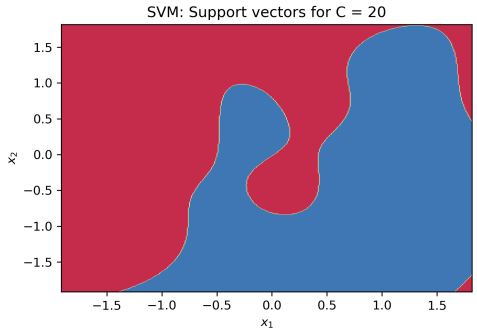
(b) Support vectors for C = 0.1



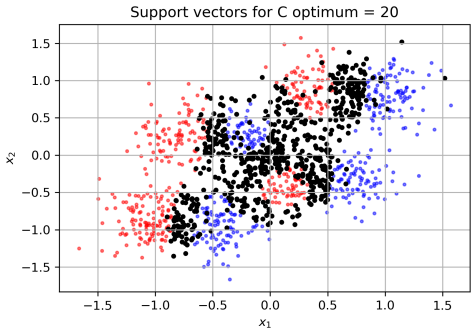
(c) Decision regions for C = 1.0



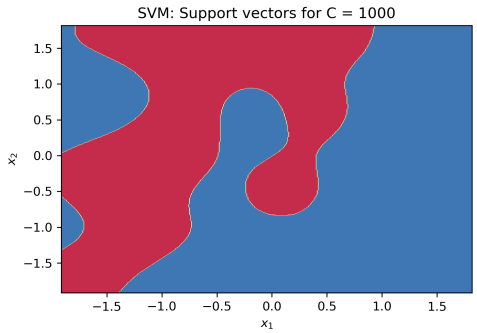
(d) Support vectors for C = 1.0



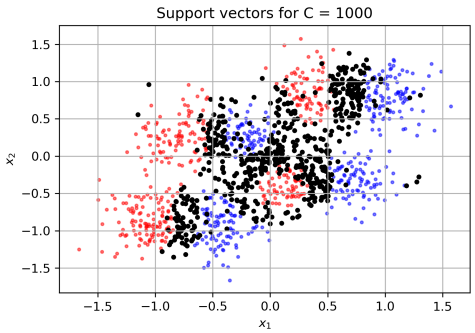
(e) Decision regions for C = 20



(f) Support vectors for C = 20



(g) Decision regions for C = 1000

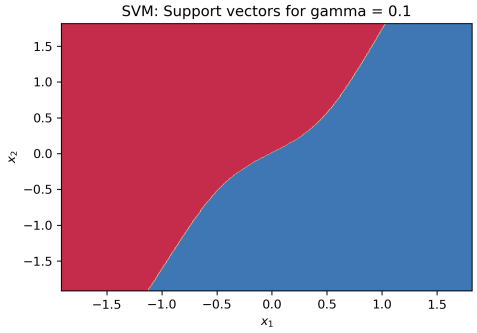


(h) Support vectors for C = 1000

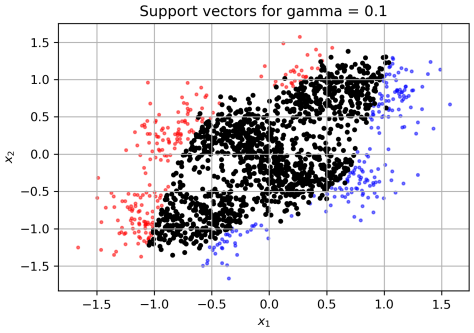
Figure 13: Decision regions and respective support vectors for different values of hyperparameter C

We now used different values of the hyperparameter C to see what impact this would cause in the classifier. In Figure 13 we can see the decision regions and the respective support vectors for values of C equal to 0.1, 1, 20 and 1000.

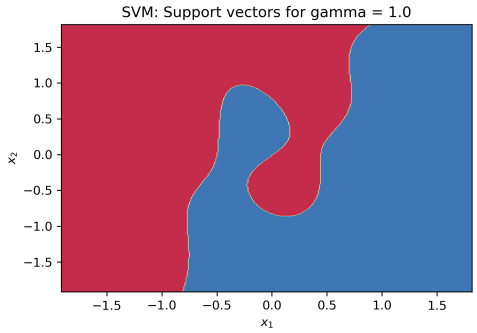
We can see that for small values of C, we don't give flexibility to the model, suggesting an underfitting. In fact, what we are doing is giving a small weight for the points that invade the border. This produces a rough division in the dataset. We can see that the support vectors are all data near the border. The opposite, occurs for example with C = 1000. We are penalizing so much the points that invade the border, that the model will try to fit all data from the training set, leading to an overfitting. This will try to keep only the data very close to the border to be the support vectors.



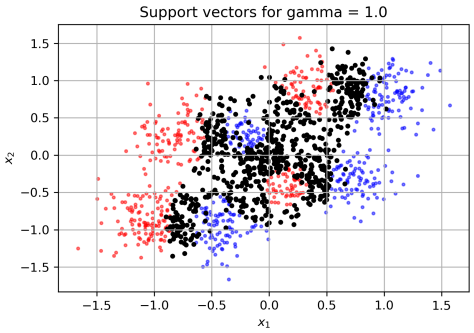
(a) Decision regions for gamma = 0.1



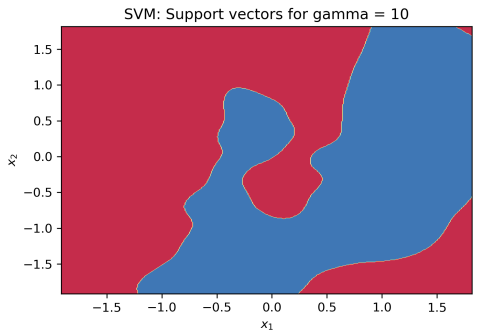
(b) Support vectors for gamma = 0.1



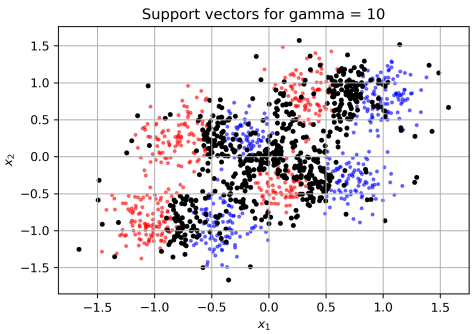
(c) Decision regions for gamma = 1.0



(d) Support vectors for gamma = 1.0



(e) Decision regions for gamma = 10



(f) Support vectors for gamma = 10

Figure 14: Decision regions and respective support vectors for different values of hyperparameter gamma in kernel function

We now change the parameter gamma of the Radial Basis Function (RBF) kernel. Values of 0.1, 1 and 10 were used. Figure 14 shows the decision regions and the support vectors identified.

A similar effect observed with hyperparameter C can be seen here. For small values of gamma, we have a very soft margin, producing a rough classifier and suggesting an underfitting. For gamma equal to 10, the model bent itself to fit almost all data. This

leads to a significant generalization loss, indicating an overfitting. Gamma = 1.0 seems to generate a reasonable classifier.