All source code is hosted here.

Problem 4.1 (Support Vector Machines)

a) The given dataset has an assimetrical distribution of the samples between the two classes y_1 and y_2. This fact is relevant especially when the data are interpolated. The SVM, indeed, tends to favor the class with the higher number of instaces. This could be a problem, because, without the appropiate precautions, the algorithm, trying to minimize the number of violations, will put the hyperplane in a way that will be easy to classificate the new samples of the major class but with a high margin of error for the minor class. Two possible solutions are:

  (a) to attribute a major weight at the violations of data samples of the minor class; or
  (b) we can do a random subsampling of the major class in order to make the two classes even in number, bringing back to a simmetric situation.

b) Assuming that our dataset has two classes y_1 and y_2 the main idea of the SVM classifier is to find an hyperplane, with equation $w^T + b = 0$, that divides the samples of the two classes with the highest distance. Defining the **margin** as the distance between the closest samples of the two classes, our goal is to maximize this quantity. Because the dimension of the margin is given by $m = \frac{2}{||w||}$, we want to find two parameters w^* and b^* that maximize it (or equivalently minimize $||w||$). Found this hyperplane, we will be able to classify a new sample x with the decision function:

$$f(x) = sign(w^* x + b^*).$$

However, in most cases in reality, can not exist a hyperplane that perfectly divides the samples of the dataset (a possible reason could be the noise). To expand the SVM also at these cases, the idea is to allow violations of the margin but penalizing them (for this reason is called *soft margin*). So now, during the search of the hyperplane, we need to consider the trade-off between maximizing the dimension of the margin and ensuring that each sample lies on the correct side of the dimension boundary. In order to do that a hyper-parameter is introduced: *C*. It determines how much a violation of the margin will be penalized. So higher this parameter will be and more the method will prefer to avoid the violantions and have a more thin margin, and viceversa. In general, the points of the dataset that lies on the margin or violate it are called *support vectors*, meanwhile, the others, *inner points*. (How said in the answer of point (c), another hyper-parameter is the choice of the kernel).

c) The SVM can be adapted to work with datasets that cannot be separated by a hyperplane in its original space. The idea is to transform the original space of the dataset into a space with higher dimensionality and hope that, in this new space, the samples become linearly separable. The problem of this step is that could be computationally expensive (we have to find a suitable transformation, transform, calculate the inner product, antitrasform). But since we need only the inner product (that mesure the "*similarity*" or distance between two vectors) of the samples to resolve our optimization problem, we can use functions, that are called *kernels*, that returns the inner product of the samples in a higher space, without transform them. Thus, another hyper-parameter is the choice of the kernel to use (others hyper-parameters derive also from the kernel choosen).

d) In our code we tested 3 parameter of the supprot vector machine:

  (a) C: Regularization parameter : 0.5, 1 or 1.5
  (b) Kernel: the type of the kernel: Linear, Poly or RBF
  (c) Gamma: Kernel coefficient: scale or auto (only for RBF and Poly)

To find the best parameter set we tried all the possible combination of these parameters. Each parameter set was tested 100 times and the accuracy of the set was evaluated based on the average accuracy.
(Note that our solution is easily extendable with new parameters.)
In our case the best parameter set was: C=1 Kernel=Linear Gamma=Auto
Accuracy: 0.97

e) In this exercise we used the paramters found above. The decision boundaries of our classification:

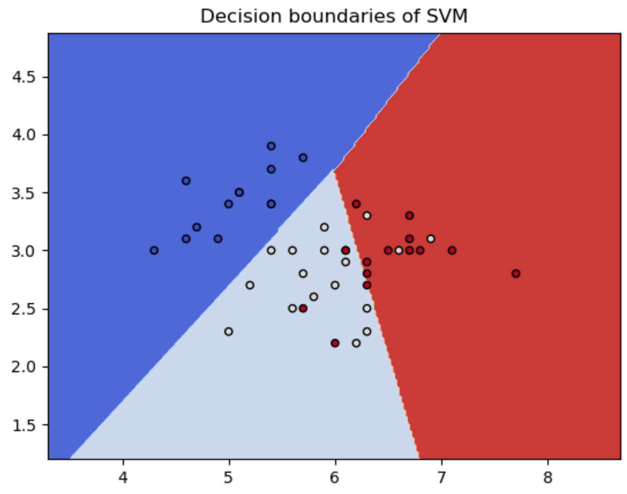


Abbildung 1: Decision boundaries of SVM on Iris dataset

Problem 4.2 (Support Vector Regression)

a) For the solution of this exercise a support vector regression has been applied to the dataset schwefel.cvs. The source code of this application can be found here.

The SVR application has been done with the following parameters for all 3 kernels:

- C = 10000
- epsilon = 0.1

The results of kernel application are below.

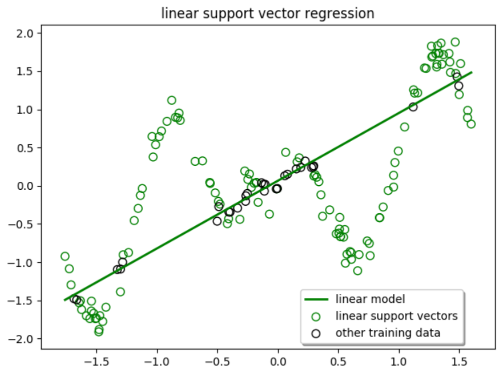


Abbildung 2: SVR application on schwefel.csv with a linear kernel

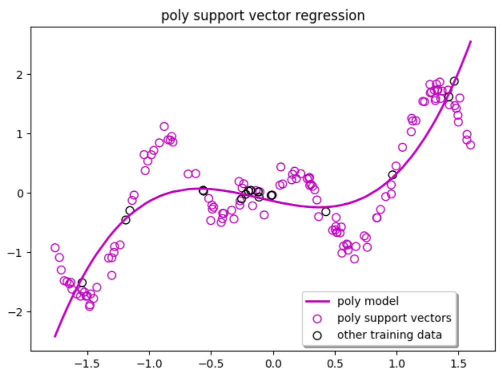


Abbildung 3: SVR application on schwefel.csv with a polynomial kernel (degree = 3)

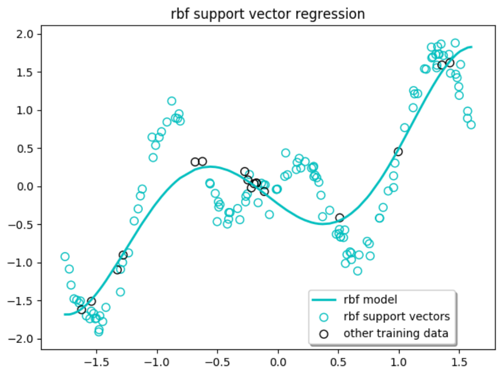


Abbildung 4: SVR application on schwefel.csv with a rbf kernel (gamma = 0.1)

What can be seen in these graphs are different hyperplanes learned by the SVR working on different kernel spaces. With the linear kernel the resulting hyperplane is forced to be a straight line (in our case, that is an R^2 space), and because of this, the resulting regression is not so close to the real data set. Meanwhile, using the polynomial and the RBF kernels we are able to determine better regressions. This come from the fact that with these kernels we can move to an higher dimensional space, and because of that the resulting hyperplane in R^2 will have a different shape (not forced to be a straight line). For both these two kernels we have to set their hyperparameters; with some tuning it’s possible to get an almost perfect regression function for the current dataset.

Epsilon

As it can be seen in these figures, the *epsilon* parameter decides the thikness of the regression tube. In the first figure *epsilon* is set to 0.01, so only points distant 0.01 or less are considered to be into the regression tube. A different situation can be seen in second picture where *epsilon* is 0.5 and so more points are considered into the tube.

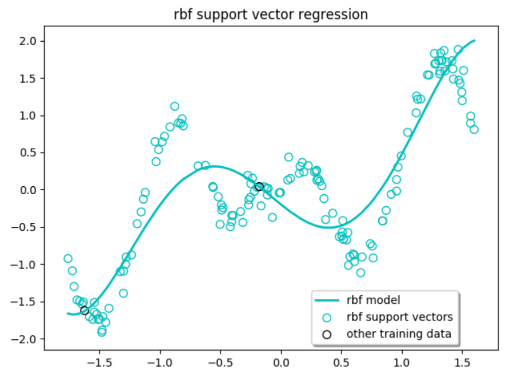


Abbildung 5: SVR application on schwefel.csv with a rbf kernel, epsilon=0.01

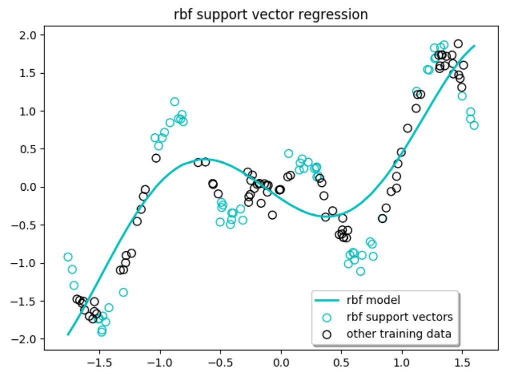


Abbildung 6: SVR application on schwefel.csv with a rbf kernel, epsilon=0.5

C

Here two different values of C parameter are tested: $C = 100$, $C = 1$. In the first figure the C parameter is greater and margin violation are more penalized than the second figure, it can be seen that the line tries to fit more the data point, in the second figure violations are less penalized so the line is nearly a straight line.

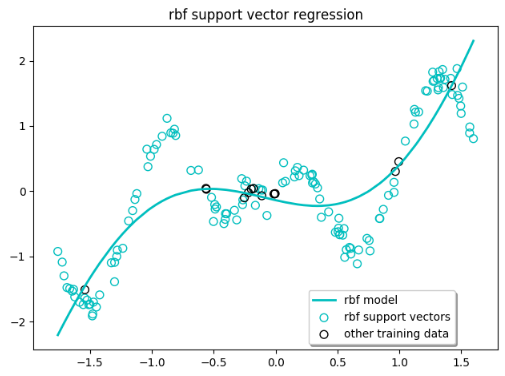


Abbildung 7: SVR application on schwefel.csv with a rbf kernel, C=100

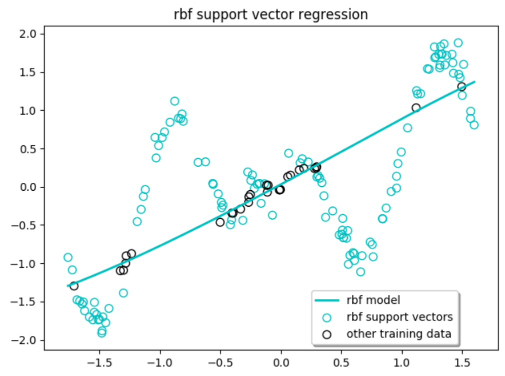


Abbildung 8: SVR application on schwefel.csv with a rbf kernel, C=1

Gamma

The gamma parameter defines how far the influence of a single training example reaches, it can be seen here that with high values the regression line fits better the dataset.

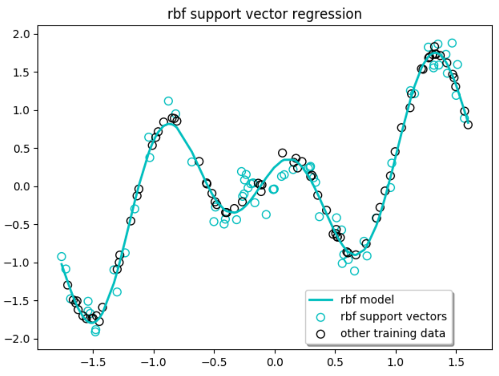


Abbildung 9: SVR application on schwefel.csv with a rbf kernel, gamma=1

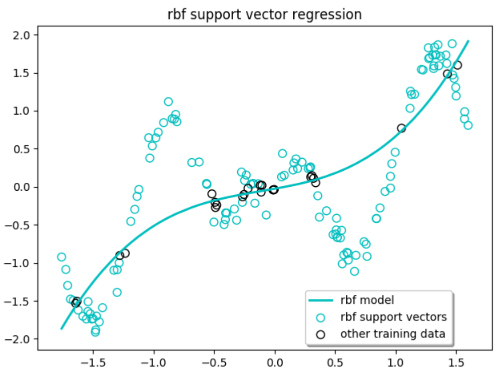


Abbildung 10: SVR application on schwefel.csv with a rbf kernel, gamma=0.01

b) The conditions a kernel has to satisfy are those given by Mercer's theorem: a kernel matrix is a symmeric and positive semidefinite matrix.

In order to prove that a Gaussian kernel is a proper kernel, we need to write the norm of vector $||x - x'||$ can be written as a vector inner product $J =< x - x', x - x' >$ that is a kernel as defined and proven in the paper given by Hans Hohenfeld on studip. So the Gaussian kernel is basically the exponential function of a kernel multiplied by a coefficient. By setting $\sigma = \sqrt{2}$ we have $K = \exp{(-J)}$ that can be written as a linear combination of J using a Taylor series:

$$K = \sum_{k=0}^{\infty} \frac{(-J)^k}{k!} = 1 + \frac{1}{2}(-J) + \frac{1}{4}(-J)^2 + \cdots .$$

that is a linear combination of kernels that is actually a kernel.

c) The kernel designed is a linear combination of a linear kernel and a RBF kernel applying this formula: $k_{custom} = \beta * k_{lin} + (1 - \beta) * k_{RBF}$ with $\beta = 0.9$. It has been chosen to give more importance to the linear kernel in order to avoid to consider too much noise in the regression tube. With the same parameters, seems to follow the dataset more compared to other kernels.

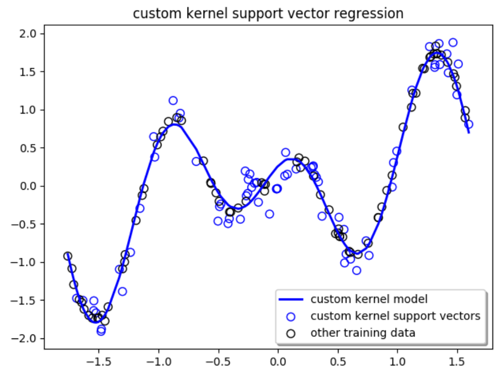


Abbildung 11: SVR application on schwefel.csv with a custom kernel, same parameters as point a

Problem 4.3 (Bayesian Optimization)

a) The first assumption to do in order to apply the bayesian optimization to an objective function is that the considered function has to be Lipschitz-continuous. The liptchitz-continuous property ensures that our objective function has no discontinuity points, so that, "moving" from the inputs x_1 to x_2, we will obtain a continous (not monotonic) increment or decrement (or even a stationarity) of the outputs from $f(x_1)$, $f(x_2)$. Also, we would like to have our function to be convex, because we are looking for the maximum of the funcion, but unfortunatly we have no way to knowing before begin the optimization process. With the only liptchitz-continuous property, we could still have the objective function very spiky, and with a function like this the bayesian optimization will turn to be not applicable. The approach consist into evaluate a very little set of points in the objective function (because the cost to evaluate is very hight) and try to predict others output based on the only knowledge of the outputs for the evaluated ones. Now it's easy to figure it out that without the liptchitz-continuous (at least) it will be impossible to predict other output values based on the observations we have done, because the output is not even continuous; and it's also easy to figure it out that more smooth is the function and more easy will be to predict the output based on the observations.

b) The algorithim, as we can almost understand on the previous answer, works in the following way. At each step of iteration what the algorithm gives back is the probabilities of outputs. This probability is given based on the analized data points, for whose we have the effective output (effected by the noise). So, what we need is a way to calculate the probability of the outputs (make predictions) given some analized (training) data, and a way to select the next data point to analize (in order to refine the predictions). In order to fulfit the first Gaussian Process Model is used. With a GP we can make prediction on outputs only looking to data we have already analized. This is a strong process because we can actually do prediction without having a model for our data. The explaination is a bit long, but we can say that every model is taken into account, but since every model could be considered gaussian, what we obtain by considering every model it's for sure, again, a gaussian. Because of his nature of a gaussian distribution, it is totally described by only a mean and a variance. The mean is not hard to calculate, so we can focus on analize what's the value of the variance. Without going too deep we can say that the variance is a matrix that takes into account the covariance matrix of the analized data points. We can assume (often it's like this) that for samples far away from the ones we have analized the covariance is low and than the uncertain of prediction is higher (and viceversa). What we need now is the second part of the algorith, to decide wich one is the next data point to evaluate in order to refine our prediction (i.e. update our covariance matrix and so our gaussian distribution). This step is called "Acquisition Functions". There are different way to do it, and we are going to describe these in the next.

c) As said before, here the topic is to find a function that gives us a way to select new point to analize (to improve the next predictions).
One of the possible approach is to consider to pick the points that have maximum probability to

give a higher value (our objective is to maximize). What we can write a probability of improvement

$$PI(x) = P(f(x) \geq f(x^+))$$

without going too deep in detail, the idea is to maximize this probability. We have also to add a parameter inside this probability function in order to avoid to pick values really close to the values that we have already analized; our function becomes

$$PI(x) = P(f(x) \geq f(x^+) + \xi)$$

and the psi allow us to move sufficiently far from our known points (remember that evaluating a sample cost much and that evaluating a point too close to the already known ones gives not so many new information). Notice that the probability of above is still described by a gaussian and that we have also the parameter ξ that has to be defined by the user.
Another idea is to select new point to evaluate using an "upper confidence bound" (we are assuming that we want to minimize in this case) and we will pick the new data point accordingly to the bound; this last described as

$$UCB = \mu(x) + \kappa\sigma(x)$$

What we do here is to maximize the upper confidence bound (that is a function) and we will pick the x for which we have the maximum as new point to evaluate. We are going to select as new data point to evaluate the one that gives back the highest UCB.
Note also in this case that the user has to define the parameter κ and it affects the optimization process.
Last but not least idea for selecting new point to evaluate is based on the "Expected Improvement". As the name suggest we would define somehow an "improvement function", and this is defined (by Mockus) as

$$I(x) = max\,\{0, f_{t+1}(x) - f(x^+)\}$$

This is set to 0 or to a positive number if we predict a value higher than the best value known. What we actually want is the x associated to that prediction.
Since we are considering gaussian process we can calculate the likelihood of improvement I as a normal density function:

$$\frac{1}{\sqrt{2\pi}\sigma(x)} exp(-\frac{(\mu(x) - f(x^+) - I)^2}{2\sigma^2(x)})$$

We stop here, saying that it's analytically possible to calculate the expected improvement and thus the point in which the improvement has his maximum.

d) Visualization of the bayesian optimization process with the upper confidence bound algorithm:

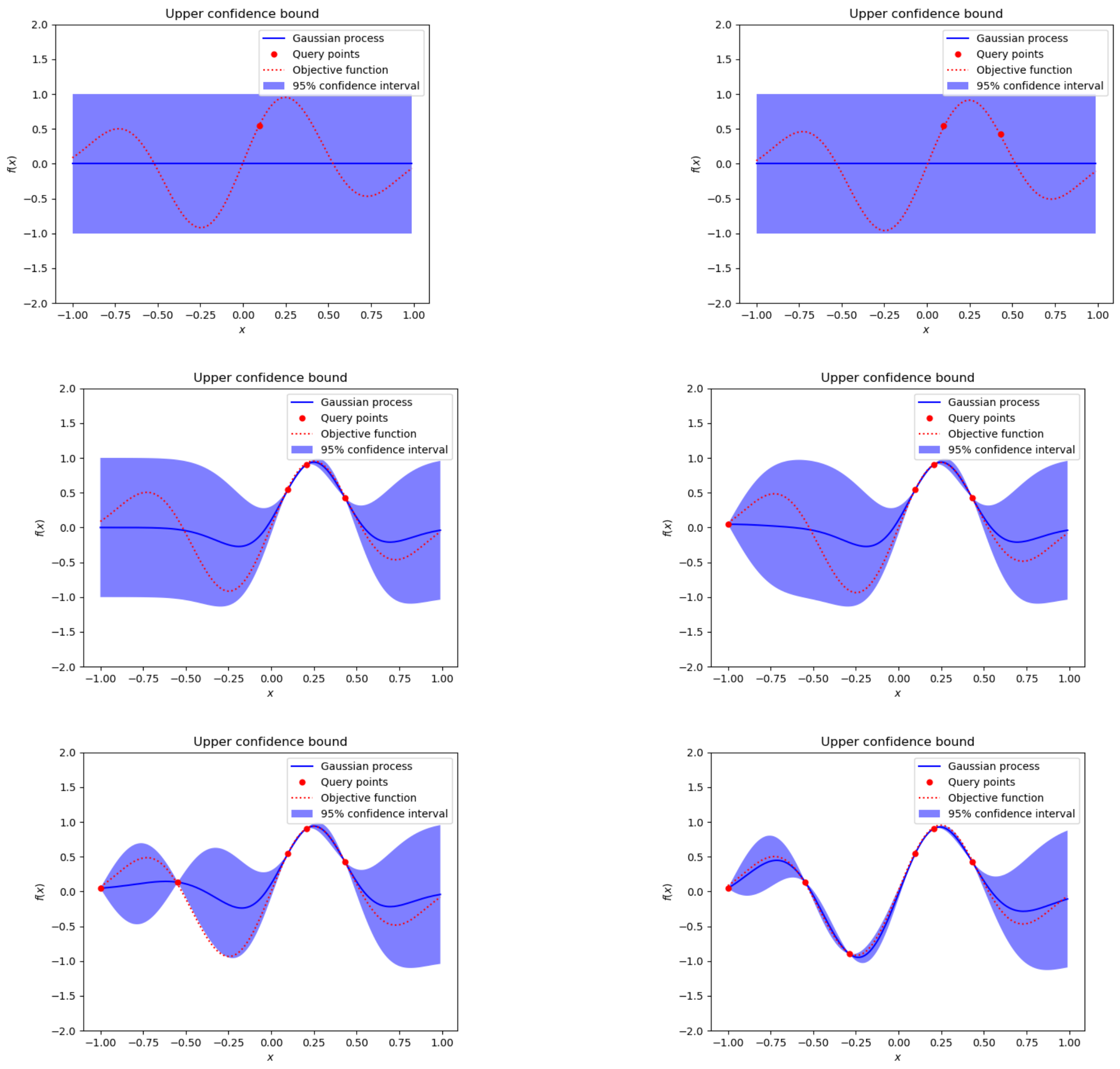
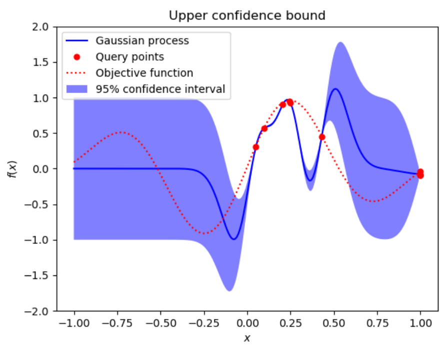
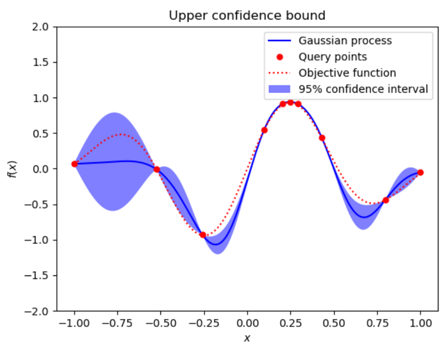


Abbildung 12: Upper confidence bound

At each iteration the algorithm choose the next point with less confidence to improve the gausian process. After some iteration the gausian process is simliar to our objective function. And then we can pick the query point with the highest output value as our maximum. We can see how the confidence is increasing after each iteration.
With the UCB we can specify the paramter k. With small k, we are decresing the value of the variance, and only considering the mean, and with big k, we have big variance weight.

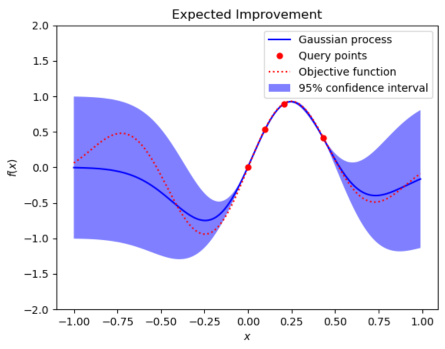


(a) Upper confidence bound with k = 0.01

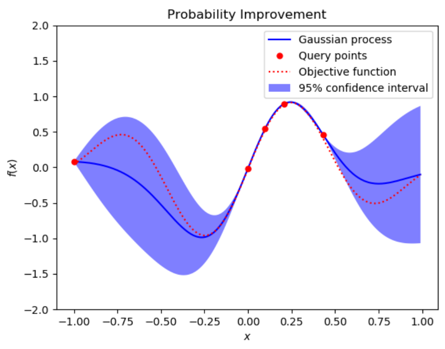


(b) Upper confidence bound with k = 5

e) The other alorithms are not trying to decrese the confidece, they are trying to find maximum value of the objective function by ”moving”the query points there. These algorithms have bigger confidence near the maximum, but less far from that point. The number of iterations in which they find the maximum value depends on the selection of the first random points, but they are faster than the UCB. (Like in the case of the PI, it started from a bad position, but then it found the maximum fast)



(a) Expected improvement



(b) Probability of improvement

Problem 4.4 (Sequential Minimal Optimization)

a) How we have seen in the first answer, in order to find a hyperplane that divides the samples of two classes, we have to resolve an optimiziation problem. Using a Lagrange, this optimization problem is transformed into the dual form where the objective function is dependent only on a set of Lagrange multipliers α_i and subject to the constraints: $0 \leq \alpha_i \leq C \ \forall i$, $\sum_i^N \alpha_i y_i = 0$. In particular, this kind of problems are called *Quadratic Problems* (QP), because consist on the optimization of quadratic objective function of several variables subject to linear constraints. In order to solve this QP, the *Karush - Kuhn - Tucker* (KKT) *conditions*, are necessary and sufficient for an optimal point. Basically, when all the Lagrange multipliers α_i will satisfy the KKT conditions, our problem will be solved.
However, solving this problem with the classical QP techiques could be difficult because of its huge

size. Infact, it involves a matrix that has a number of elements equal to the square of the number of training examples.
The basic idea of *Sequential Minimal Optimization algorithm* (SMO), differently from other methods, is to solve at each step the smallest QP problem. Because of the linear equality constraint, the smallest QP problem involves two Lagrange multipliers, otherwise, with only one multiplier, the SMO could not fulfill the constraint at each step. This idea is based on the *Osuna's Theorem*, that allows us to solve the QP main problem dividing it in a set of smaller QP sub-problems and guarantees the convergence of the method. So, the principals advantages of SMO compared with the other methods are:

(a) since at each step the algorithm consider only two Lagrange multipliers α_1 and α_2, the solution for α_1 and α_2 can be found analitically and so without using any iterative QP optimizer as part of the algorithm (differently from other methods).
(b) No extra matrices are involved in the algorithm, so it is less expensive with respect to the memory usage and less susceptible to numerical precision problems.

b) At each step the algorithm, with an heuristic function, choose two Lagrange multipliers α_1, α_2 and provides to optimize the problem, keeping fixed the others Lagrange multipliers. Notice that, from the constraint $0 \leq \alpha_i \leq C$, derives that α_1 and α_2 will lie within a box and, from the constraint $\sum_i^N y_i\alpha_i = 0$, derives that they will lie in a diagonal line because, reminding that the others Lagrange multipliers are fixed, has to be true that:

$$y_1\alpha_1^{old} + y_2\alpha_2^{old} = const. = y_1\alpha_1^{new} + y_2\alpha_2^{new}.$$

Said that, the algorithm looks first for α_2^{new} and after derives α_1^{new} from it. In particular, two different bounds can be applied to α_2^{new}, depending on whether $y_1 = y_2$ or $y_1 \neq y_2$:

(a) if $y_1 \neq y_2$:
$$L = max(0, \alpha_2^{old} - \alpha_1^{old}) \leq \alpha_2^{new} \leq min(C, C + \alpha_2^{old} - \alpha_1^{old}) = H$$
(b) if $y_1 = y_2$:
$$L = max(0, \alpha_2^{old} + \alpha_1^{old} - C) \leq \alpha_2^{new} \leq min(C, \alpha_2^{old} + \alpha_1^{old}) = H$$

At this point, we can calculate an unclipped value of α_2^{new} in this way:

$$\alpha_2^{new} = \alpha_2^{old} + \frac{y_2(E_1 - E_2)}{k}$$

where E_i is the error on the $i-th$ sample and k is the value of the second derivative calculated along the diagonal line (usually it is positive, but sometimes could happen that it is non-positive; in this cases the new α_2 is found differently); and clipping it to enforce the constraint $L \leq \alpha_2^{new} \leq H$:

$$\alpha_2^{new,clipped} = \begin{cases} H & if & \alpha_2^{new} \geq H \\ \alpha_2^{new} & if & L < \alpha_2^{new} < H \\ L & if & \alpha_2^{new} \leq L \end{cases}$$

Thus, we can calculate
$$\alpha_1^{new} = \alpha_1 + y_1y_2(\alpha_2 - \alpha_2^{new,clipped}).$$

c) How we said before, the choice of Lagrange multipliers α_1 and α_2 is realized through heuristics. The first multiplier is chosen alternating a single search on the entire training set of the multipliers that violates the KKT conditions and multiple search on the set of *non-bound samples* (Lagrange multipliers that are neither 0 nor C) that, again, violates the KKT conditions.
At this point, the choice of the second multiplier is done trying to maximize the positive progress that the SMO will achive in the optimization by α_1 and α_2. The progress amount is approximated by SMO with the quantity: $|E_1 - E_2|$, where E_i is the error on the $i - th$ non-bound sample that is kept cached by the algorithm. Depending on the error's value of the first multiplier chosen, SMO will choose the second one:

- if $E_1 \leq 0$, SMO will choose the sample with the maximum error E_2;

- if $E_1 > 0$, SMO will choose the sample with the minimum error E_2.

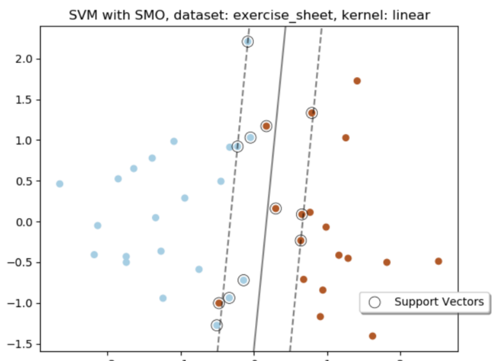
However, in some particular cases, this kind of heuristic could not work. For this reason the SMO provides a hierarchy of second choice heuristics:

(a) if the heuristic above doesn't find a multiplier that will make positive progress, the SMO iterates on the subset of non-bound sample searching for a multiplier that will make positive progress;

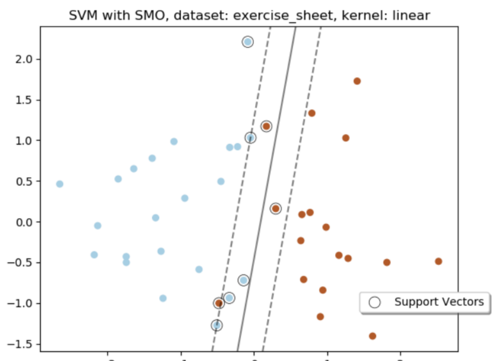
(b) if, after point *a*), the algorithm hasn't still found a second multiplier, it iterates on the entire training set;

(c) if, also the point *b*) has produced no result, the SMO skips the first choice and restart the procedure.

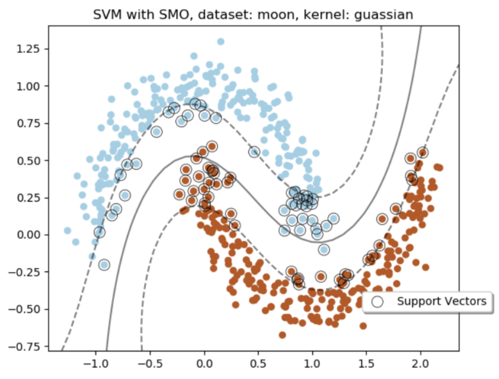
d) In the figures below are reported the plots produced by our implementation of SMO algorithm on the given dataset (15a, 15b) and on the *moons* dataset of *Scikit-learn* (15c 15d). We used fixed values for the tolerances (*tol* = 0.01, *eps* = 0.05) but different values for the *Regularization parameter*, C, and different types of *Kernel*.



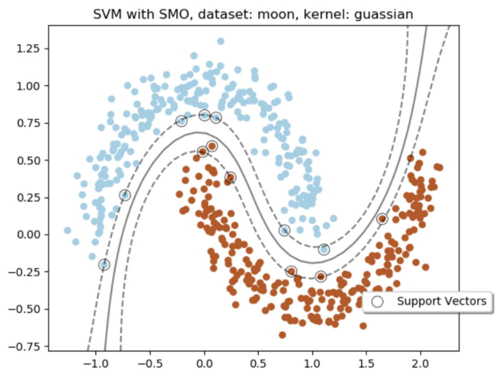
(a) $C = 1, Kernel = Linear$



(b) $C = 100, Kernel = Linear$



(c) $C = 1, Kernel = Gaussian$



(d) $C = 100, Kernel = Gaussian$

Abbildung 15: SMO plots