

## SVM

Both *perceptrons* and *support vector machines* (SVM) are linear classifiers. What are the differences between an SVM and Perceptron?

### Solution

A perceptron will converge to *some* hyperplane separating the classes - if one exists. An SVM will converge to the hyperplane with the maximal geometric margin, i.e. the plane maximizing the distance to the closest data points (the support vectors).

---

An SVM classifies an instance  $x$  as

$$\text{sgn}(\sum_i \alpha_i y^{(i)} K(x^{(i)}, x) + b)$$

Explain the different components in the formula. What are:

- $\alpha_i$
- $y^{(i)}$  and  $x^{(i)}$
- $K$
- $b$

### Solution

$x^{(i)}$  are the support vectors and  $y^{(i)}$  their classification (-1 or +1), while  $\alpha_i$ s represent their weights in the SVM. Alternatively  $i$  can run over all data points, but with  $\alpha_i$  set to zero for points that are not support vectors.

$K$  is the kernel function, and represents the inner product for vectors transformed to a different (typically higher-dimensional) space.  $b$  is the bias.

---

Typically, SVMs use the "kernel trick" to construct linear classifiers in a different space from the input space. Why is this important?

Prove that a weighted sum of inner products is a linear classifier, i.e. show that

$$\sum_i w_i x_i^T x$$

is a linear classifier of  $x$ .

### Solution

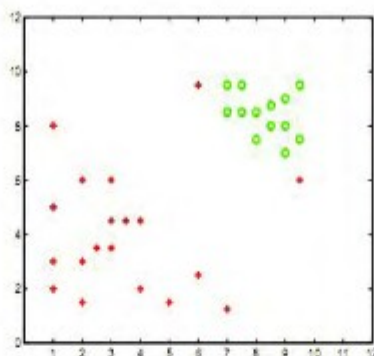
The kernel trick represents a transformation of the data into a higher dimensional space, but where we can construct inner products implicitly - that is, without calculating the actual transformation. This is important, as it allows us to perform non-linear classification.

$$\sum_i w_i x_i^T x = \sum_i w_i \sum_j x_{ij} x_j = \sum_j \sum_i w_i x_{ij} x_j$$

So we see that this is a vector product (and thus it represents a linear classifier) of the vector with components  $\sum_i w_i x_{ij}$  and the vector  $x$ .

---

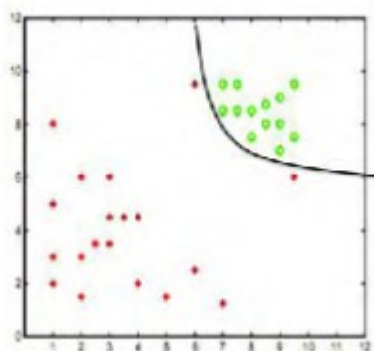
For this question assume that we are training an SVM with a quadratic kernel - i.e. our kernel function is a polynomial kernel of degree 2. This means the resulting decision boundary in the original feature space may be parabolic in nature. The dataset on which we are training is given below:



The slack penalty  $C$  will determine the location of the separating parabola. Please answer the following questions qualitatively.

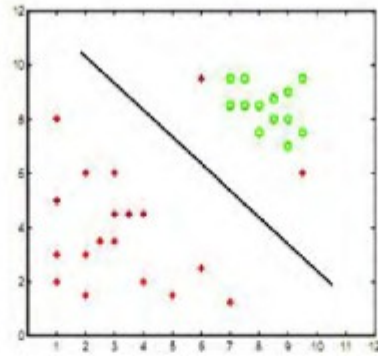
- (a) [5 points] Where would the decision boundary be for very large values of  $C$ ? (Remember that we are using a quadratic kernel). Justify your answer in one sentence and then draw the decision boundary in the figure below.

**Answer:** Since  $C$  is too large, we can't afford any misclassification. Also we want to minimize  $\|w\|$ , therefore the  $x^2$  constant is minimum, and hence among all the parabolas, we choose the minimum curvature one.

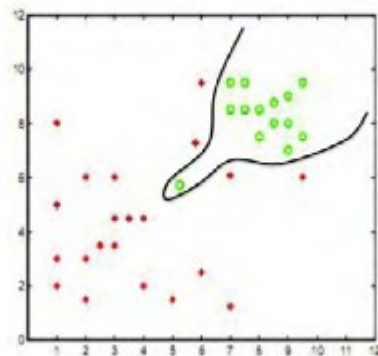


- (b) [5 points] Where would the decision boundary be for  $C$  nearly equal to 0? Justify your answer in one sentence and then draw the decision boundary in the figure below.

**Answer:** Since the penalty for mis-classification is too small, the decision boundary will be linear to have  $x^2$  constant equal to 0.



- c) [5 points] Now suppose we add three more data points as shown in figure below. Now the data are not quadratically separable, therefore we decide to use a degree-5 kernel and find the following decision boundary. Most probably, our SVM suffers from a phenomenon which will cause wrong classification of new data points. Name that phenomenon, and in one sentence, explain what it is.



**Answer:** Over-fitting. It happens when we use an unnecessarily complex model to fit the training data.

- (1) [2 pts] What is the primary motivation for using the kernel trick in machine learning algorithms?

If we want to map sample points to a very high-dimensional feature space, the kernel trick can save us from having to compute those features explicitly, thereby saving a lot of time.

(Alternative solution: the kernel trick enables the use of infinite-dimensional feature spaces.)

---

(a) (**True/False** - 1 pt ) Support vector machines, like logistic regression models, give a probability distribution over the possible labels given an input example.

**Answer:** False

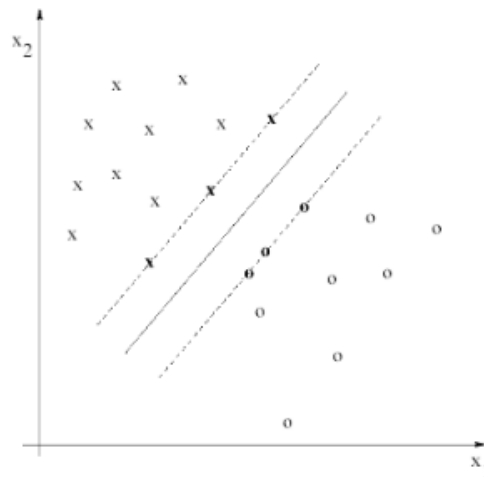
(b) (**True/False** - 1 pt ) We would expect the support vectors to remain the same in general as we move from a linear kernel to higher order polynomial kernels.

**Answer:** False ( There are no guarantees that the support vectors remain the same. The feature vectors corresponding to polynomial kernels are non-linear functions of the original input vectors and thus the support points for maximum margin separation in the feature space can be quite different. )

(c) (**True/False** - 1 pt ) The maximum margin decision boundaries that support vector machines construct have the lowest generalization error among all linear classifiers.

**Answer:** False ( The maximum margin hyperplane is often a reasonable choice but it is by no means optimal in all cases. )

(f) ( 2 pts ) What is the leave-one-out cross-validation error estimate for maximum margin separation in the following figure ? (we are asking for a number)



**Answer:** 0

Based on the figure we can see that removing any single point would not change the resulting maximum margin separator. Since all the points are initially classified correctly, the leave-one-out error is zero.

(g) ( 2 pts ) Now let us discuss a SVM classifier using a second order polynomial kernel. The first polynomial kernel maps each input data  $x$  to  $\Phi_1(x) = [x, x^2]^T$ . The second polynomial kernel maps each input data  $x$  to  $\Phi_2(x) = [2x, 2x^2]^T$ .

In general, is the margin we would attain using  $\Phi_2(x)$

- A. ( ) greater
- B. ( ) equal
- C. ( ) smaller
- D. ( ) any of the above

in comparison to the margin resulting from using  $\Phi_1(x)$  ?

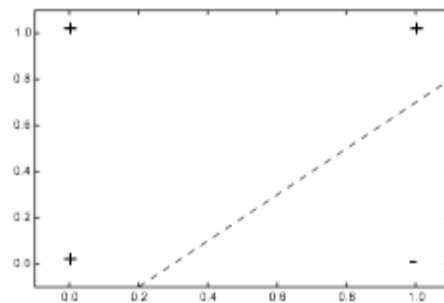
**Answer:** A.

**Perceptron.** Consider the following Boolean function:

$x_1$	$x_2$	$y = \neg x_1 \cup x_2$
0	0	1
0	1	1
1	0	0
1	1	1

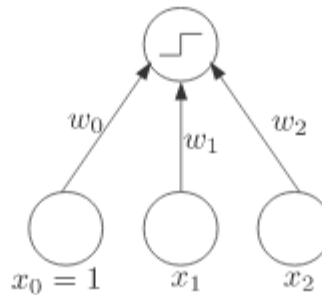
- (a) (2 points) Can this function be represented by a perceptron? Explain your answer.

**Solution:** Yes, because the function is linearly separable.



- (b) (4 points) If yes, draw a perceptron that represents it. Otherwise, build a multilayer neural network that will.

**Solution:** A perceptron has the following architecture:



$$w_0 = 1, w_1 = -1, w_2 = 2$$

Its output is given by: 1 if  $w_0 + w_1x_1 + w_2x_2 > 0$  and 0 otherwise.

This is one of many possible solutions.  $w_0, w_1, w_2$  must give the equation of a line that separates  $(1, 0)$  from  $(0, 0), (0, 1)$  and  $(1, 1)$ .