Machine Learning and Photonics

Week 7

Ergun Simsek

University of Maryland Baltimore County

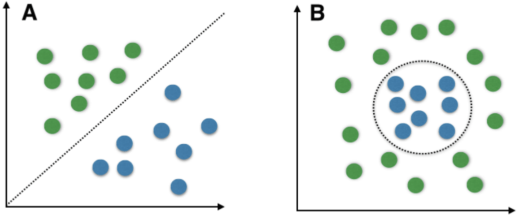
simsek@umbc.edu

Support Vector Machine

March 13, 2023

1 Support Vector Machine

- Introduction
- Formulation of Support Vectors
- Hard and Soft Margin
- Slack Variable
- Kernel Method

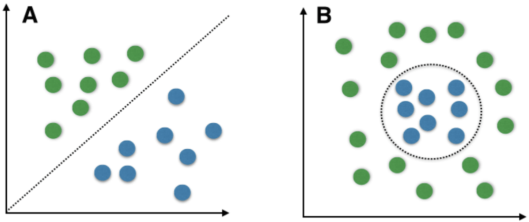


Introduction to SVM

- SVM is one of the most robust classification methods.
- Unlike the logistic regression that takes a probabilistic approach toward predicting categorical target variables, SVM takes a purely **geometric approach**.
- The focus of today's lecture is however on binary classifications in the context of supervised learning.

Basic Idea of Binary SVM

- Partition the feature space into two components, one for each class of a binary classification problem.
- The partitioning in the feature space is done in one of the two following ways.
  1. **Support Vector Classifier**: Use straight boundaries to partition the feature space into two regions by inserting a *hyperplane*
  2. **Support Vector Machine**: The boundaries of the two regions are allowed to have curvature
- The data scientists are typically not very careful about the distinction between SVCs and SCMs, and they may use them interchangeably.



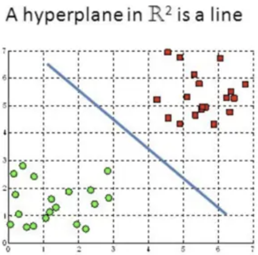
Formulation of Support Vectors

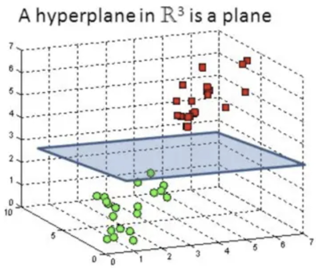
- Assume that we have a *binary classification* problem with target variable y, and d continuous features $\vec{x} \in \mathbb{R}^d$.
- The target variable y takes values $\{-1, +1\}$ corresponding to the two classes.
- The values of the features $\vec{x}$ and the target y are recorded for n observations to form a dataset.

Hyperplanes

A support vector classifier simply divides the feature space into two parts by inserting a a hyperplane.

- A hyperplane in a 2-dimensional space is simply a straight line.
- In the 3-dimensional case, a hyperplane is simply a Euclidean plane.
- In higher dimensions ($d > 3$), a hyperplane is a generalization of the concept of plane in three dimensions.





Recall that for two vectors $\vec{v}$ and $\vec{w}$ in the d-dimensional space $\mathbb{R}^d$, the dot product $\vec{v} \cdot \vec{w}$ is defined by

$$\vec{v} \cdot \vec{w} = v_1 w_1 + v_2 w_2 + \cdots + v_d w_d = \sum_{i=1}^{d} v_i w_i \ . \tag{1}$$

As is evident from (1), the result (output) of the dot product is a *scalar* (*i.e.* a real number). You can easily calculate the dot product through numpy if you will.

Example: Let $\vec{v} = \langle 1, 2, -3, 3, 5 \rangle$ and $\vec{w} = \langle -1, 3, 1.5, 2.5, -4 \rangle$ be vectors in $\mathbb{R}^5$. What is $\vec{v} \cdot \vec{w}$?

:
```python
import numpy as np

v = np.array([1, 2, -3, 3, 5])
w = np.array([-1, 3, 1.5, 2.5, -4])

print('v.w =', np.dot(v, w))
```

```
v.w = -12.0
```

Now let $\vec{w}$ be a fixed vector in the feature space, $\mathbb{R}^d$. We define the following scalar function f

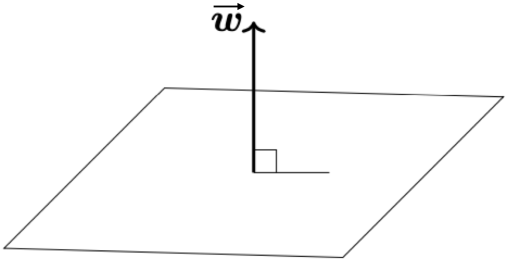
$$
\begin{aligned}
f :\ & \mathbb{R}^d \longrightarrow \mathbb{R} \\
f(\vec{x}) &= \vec{w} \cdot \vec{x} + b \, ,
\end{aligned} \tag{2}
$$

where b is a scalar ($b \in \mathbb{R}$). A hyperplane in $\mathbb{R}^d$ is now defined as the set of all points $\vec{x}$ in feature space $\mathbb{R}^d$ such that $f(\vec{x}) = 0$:

$$
\text{hyperplane} \; = \{\vec{x} \in \mathbb{R}^d \,|\, f(\vec{x}) = 0\} \; . \tag{3}
$$

Remember this equation! It says that if we have a hyperplane, then $f(\vec{x}) = 0$.

The fixed vector $\vec{w}$ is a vector *normal (perpendicular) to the hyperplane*.



Hence, Eq. (3) not only defines a hyperplane, it additionally defines a direction ($\vec{w}$).

- In order to classify an instance $\vec{x}^{(i)}$, we have to see on what side of the hyperplane the instance is located.
- The direction defines the positive and negative sides of the hyperplane.
- To classify the *i*-th example, we calculate $f(\vec{x}^{(i)})$.
  - If $f(\vec{x}^{(i)}) > 0$, then the *i*-th instance is labeled positive, and
  - if $f(\vec{x}^{(i)}) < 0$, it is labeled negative.

When we train the support vector classifier, we want to ensure that

$$\begin{cases} f(\vec{x}^{(i)}) > 0 & \text{when} \quad y^{(i)} = +1 \,, \\ f(\vec{x}^{(i)}) < 0 & \text{when} \quad y^{(i)} = -1 \,. \end{cases} \tag{4}$$

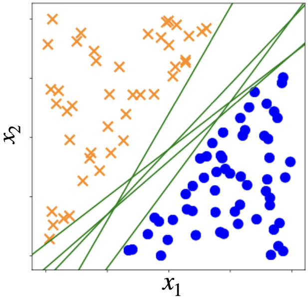
The two equations above can be combined into one single equation as follows

$$y^{(i)}\left(f(\vec{x}^{(i)})\right) > 0 \quad \text{for} \quad i = 1, 2, \cdots, n \,. \tag{5}$$

Question: Does this requirement (*i.e.* equation (5)) determine a hyperplane as a classifier uniquely?

Formulation of Support Vectors (Cont...)

- Consider a linearly separable classification problem with only two features x_1 and x_2.
- In this case, the feature space is a copy of $\mathbb{R}^2$, and a hyperplane simply corresponds a straight line.
- There are infinitely many choices for the classifying hyperplane!



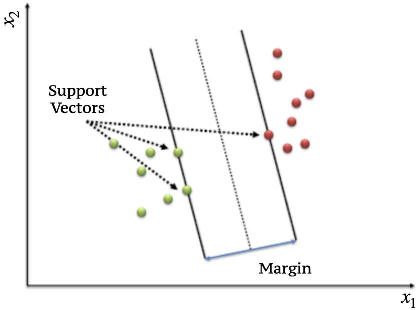
Question: Which hyperplane should we choose?
To be able to compare different hyperplanes with one another, we need to define the concept of **margin**

Introduction to "Margin" and Normalization

- Margin: The distance of the separating hyperplane from the closest instance in the dataset.
- With this definition, we encounter a conceptual problem: How do we make sense of distance when different features carry different physical dimensions? E.g.,
  - Feature-1: Thickness of a semiconducting film (10^{-5} m – 10^{-9} m)
  - Feature-2: Doping concentration (0 – 10^{20} cm^{-3}).
- When there are orders of magnitude differences among features, then transformations should be applied ($\log$, $\exp$, etc.)
- Once the features have magnitudes comparable to each other, then we should standardize them.
- *Data standardization*: Instead working with $\vec{x}^{(i)}$, we work with the standardized instances $\frac{x_j^{(i)}-\mu_j}{\sigma_j}$ for each feature $j = 1, 2, \cdots, d$, where μ_j and σ_j represent the mean and the standard deviation of the j^{th} feature, respectively.

(Hard) Margin

- Assume that we are working with a standardized dataset
- Suppose a separating hyperplane is given.
- The *distance of the closest instance of the dataset to the given hyperplane* is said to be the **margin** of the hyperplane.
- Each instance of the dataset which possesses the least distance (*i.e.* the margin) to the separating hyperplane is said to be a **support vector**.

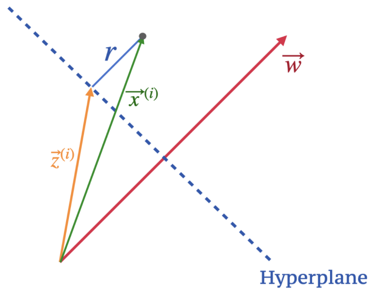


How do we find the (Hard) Margin?

- Suppose a hyperplane and its normal vector $\vec{w}$ are given.
- Assume that the i^{th} instance, $\vec{x}^{(i)}$, of the dataset is a support vector.
- $\vec{r}$ is a vector that is parallel with the normal vector $\vec{w}$.

$$\vec{x}^{(i)} = \vec{z}^{(i)} + r\hat{w} = \vec{z}^{(i)} + r\frac{\vec{w}}{|\vec{w}|}\ , \qquad (6)$$

where $\hat{w} = \frac{\vec{w}}{|\vec{w}|}$ is the unit vector along the normal vector $\vec{w}$



Since the margin is the closest distance to the separating hyperplane, we can express the constraints in (5) as

$$y^{(i)}\left(f(\vec{x}^{(i)})\right) \geq r \quad \text{for} \quad i = 1, 2, \cdots, n \,. \tag{7}$$

Between two separating hyperplanes, the better one is the one which comes with the greater margin.

Thus, we arrive at the following constrained optimization problem:

$$\begin{aligned} &\underset{\vec{w},b}{\text{argmax}} \quad r \\ &\text{subject to:} \begin{cases} y^{(i)}\left(f(\vec{x}^{(i)})\right) \geq r, & \text{for} \quad i = 1, 2, \cdots, n \\ |\vec{w}| = 1 \\ r > 0 \end{cases} \end{aligned} \tag{8}$$

We can rewrite this optimization problem in a slightly different form.

Let's say we have this function $f(\vec{\alpha})$, which takes the dot product of $\vec{w}$ with $\vec{\alpha}$ and then add scalar b.

Let's perform input $\vec{x}^{(i)}$ to this function, i.e., take the dot product on the two sides of $\vec{x}^{(i)} = \vec{z}^{(i)} + r\hat{w} = \vec{z}^{(i)} + r\frac{\vec{w}}{|\vec{w}|}$ with vector $\vec{w}$ and add scalar b to get

$$\begin{aligned} & \vec{w} \cdot \vec{x}^{(i)} + b = \vec{w} \cdot \left(\vec{z}^{(i)} + r\frac{\vec{w}}{|\vec{w}|}\right) + b = \left(\vec{w} \cdot \vec{z}^{(i)} + b\right) + r\frac{\vec{w} \cdot \vec{w}}{|\vec{w}|} \\ \implies \quad & f(\vec{x}^{(i)}) = f(\vec{z}^{(i)}) + r|\vec{w}| \\ & \text{Remember that if } f(\vec{z}^{(i)}) \text{ is a point on the hyperplane, then it needs to be zero} \\ \implies \quad & r = \frac{C^{(i)}}{|\vec{w}|}, \end{aligned} \tag{9}$$

where $C^{(i)}$ is a constant of the support vector (*i.e.* $C^{(i)} = f(\vec{x}^{(i)})$).

How do we find the (Hard) Margin? *(Cont...)*

$r = \frac{c^{(i)}}{|\vec{w}|}$

- Therefore, instead of maximizing r in Eq. (8), we can maximize $\frac{1}{|\vec{w}|}$, and relax the constraint $|\vec{w}| = 1$ in Eq. (8).
- Since in ML, we typically prefer to minimize a function (rather than maximizing), we can minimize $\frac{1}{2}|\vec{w}|^2$ (instead of maximizing $\frac{1}{|\vec{w}|}$), as the minima of $\frac{1}{2}|\vec{w}|^2$ are exactly the maxima of $\frac{1}{|\vec{w}|}$.

How do we find the (Hard) Margin? *(Cont...)*

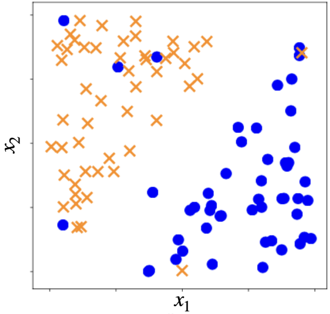
- We can do for further convenience is to divide the two sides of the inequality $y^{(i)}\left(f(\vec{x}^{(i)})\right) \geq r$ in Eq. (8) by r, and absorb the r in the definition of the normal vector w and the scalar b.
- We finally arrive at the following optimization problem:

$$\begin{aligned} &\underset{\vec{w},b}{\operatorname{argmin}} \quad \frac{1}{2}|\vec{w}|^2 \\ &\text{subject to: } y^{(i)}\left(f(\vec{x}^{(i)})\right) \geq 1, \quad \text{for} \quad i = 1, 2, \cdots, n\,. \end{aligned} \tag{10}$$

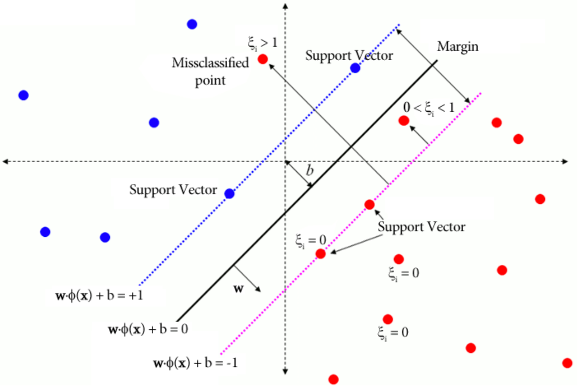
Comment: *SVM is computationally a very expensive algorithm*!

Hard Margin vs. Soft Margin

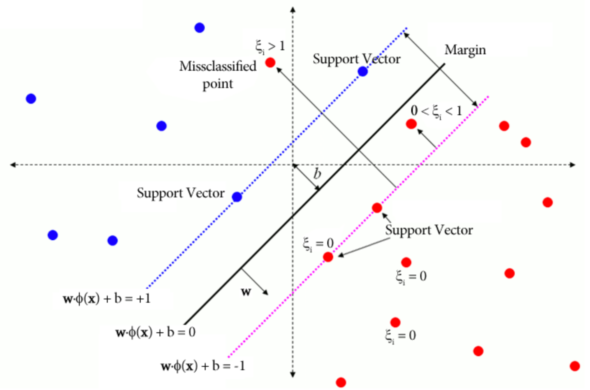
- The previous formulation is effective and can offer an optimal separating hyperplane when the dataset associated with the binary classification is *linearly separable*.
- But what if the dataset is not linearly separable? In that case, there won't exist a hyperplane which separates the two classes perfectly.
- We need to develop a *tolerance for misclassifications*!



- Soft margin allows misclassifications.
- To formalize the soft margin SVM, we introduce a **slack variable**, $\xi^{(i)}$, associated with each instance $(\vec{x}^{(i)}, y^{(i)})$ of the dataset.
- Geometrically, $\xi^{(i)}$ measures the distance of the i^{th} instance in the feature space
  - from the positive margin if the actual label of the i^{th} instance is positive (*i.e.* $y^{(i)} = +1$), and
  - from the negative margin if the actual label of the i^{th} instance is negative (*i.e.* $y^{(i)} = -1$).



- The slack variable $\xi^{(i)}$ is positive semidefinite (*i.e.* $\xi^{(i)} \geq 0$).
- If $0 < \xi^{(i)} \leq 1$, then the i^{th} instance violates the margin, but is still on the right side of the hyperplane.
- If $\xi^{(i)} > 1$, then the i^{th} instance is on the right side of the hyperplane and leads to a misclassification for the algorithm.



Now, with the help of slack variables, we can introduce a new version of the optimization, Eq. (10), which allows for misclassifications when we deal with nonlinearly separable cases.

Soft Margin SVM

The new optimization problem in the presence of the slack variables is given by

$$\underset{\vec{w},b,\xi^{(1)},\xi^{(2)},\cdots,\xi^{(n)}}{\operatorname{argmin}} \quad \frac{1}{2}|\vec{w}|^2$$
$$\text{subject to:} \begin{cases} y^{(i)}\left(f(\vec{x}^{(i)})\right) \geq 1-\xi^{(i)}, & \text{for} \quad i=1,2,\cdots,n \\ \sum_{i=1}^{n}\xi^{(i)} \leq C\,, & \\ \xi^{(i)} \geq 0, & \text{for} \quad i=1,2,\cdots,n \end{cases} \tag{11}$$

where C is a hyperparameter that determines the number and severity of the violations to the margin that we will tolerate.

- When $C = 0$, no violation is tolerated and $\xi^{(1)} = \xi^{(2)} = \cdots = \xi^{(n)} = 0$.
- When $C > 0$, the algorithm learns (through the training process) how to optimally spend the margin violation to get the minimum $|\vec{w}|^2$.
- The greater C is, the more margin violation is tolerated.

Cost function of Soft Margin SVM

One can incorporate the constraints in (11) into a cost function $\mathcal{L}$ as follows:

$$\mathcal{L}(\vec{w}, b, \xi^{(i)}, \alpha_i, \gamma_i) = \frac{1}{2}|\vec{w}|^2 - \sum_{i=1}^{n} \alpha_i \left(y^{(i)} f(\vec{x}^{(i)}) - 1 + \xi^{(i)} \right) + C \sum_{i=1}^{n} \xi^{(i)} - \sum_{i=1}^{n} \gamma_i \xi^{(i)} , \quad (12)$$

where the last three terms in above incorporate the three constraints in (11). Setting $\frac{\partial \mathcal{L}}{\partial w_i} = 0$, $\frac{\partial \mathcal{L}}{\partial b} = 0$, and $\frac{\partial \mathcal{L}}{\partial \xi^{(i)}} = 0$, we can find a cost function, $\mathcal{D}(\alpha_i, \gamma_i)$, purely in terms of the Lagrange multipliers

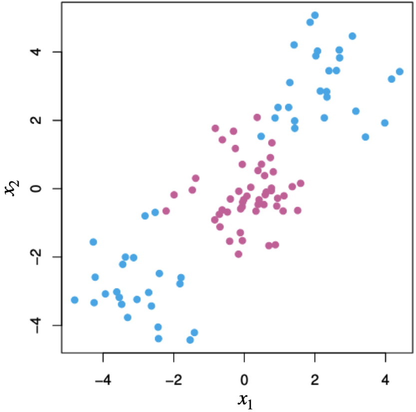
$$\mathcal{D}(\alpha_i, \gamma_i) = \frac{1}{2} \sum_{i=1}^{n} \sum_{j=1}^{n} \alpha_i \alpha_j y^{(i)} y^{(j)} \vec{x}^{(i)} \cdot \vec{x}^{(j)} - \sum_{i=i}^{n} \alpha_i ,$$

$$\text{subject to: } \sum_{i=1}^{n} \alpha_i y^{(i)} = 0 . \quad (13)$$

- The cost function $\mathcal{D}(\alpha_i, \gamma_i)$ is sometimes referred to as the **dual SVM cost function**.
- The solution of the dual SVM problem (*i.e.* solutions for α_i and γ_i through minimization of $\mathcal{D}(\alpha_i, \gamma_i)$) would then determine the solution to the original SVM problem (12) through $\vec{w} = \sum_{j=1}^{n} \alpha_j y^{(j)} \vec{x}^{(j)}$.
- Note that α_j is nonzero *only when the j^{th} instance is a support vector*.

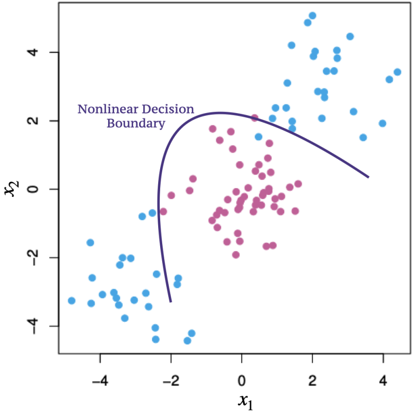
Support Vector Machine (Kernel Method)

- In the previous section, we studied the support vector classifier in which the boundary between classes is given by a hyperplane.
- However, we often encounter cases where a hyperplane cannot perform an reasonably well classification job, exactly because the boundary between the two classes is nonlinear.
- We need to allow for nonlinear boundaries between classes!



Support Vector Machine (Kernel Method) *(Cont...)*

- Kernel method allows nonlinear boundary definition.
- Key idea: compute the inner products between the images of all pairs of data in the feature space.
- The kernel trick is *computationally much cheaper* than the explicit computation of the coordinates.



First, we define a function g which considers the dot product of the argument vector $\vec{x}$ with all data points as follows:

$$\begin{aligned} & g : \mathbb{R}^d \longrightarrow \mathbb{R} \\ & g(\vec{x}) = b + \sum_{i=1}^{n} \alpha_i \, \vec{x} \cdot \vec{x}^{(i)} = b + \vec{x} \cdot \Big(\sum_{i=1}^{n} \alpha_i \, \vec{x}^{(i)} \Big) \; . \end{aligned} \tag{14}$$

Note that (14) reduces to defining equation of the hyperplane (3) upon identification $\vec{w} = \sum_{i=1}^{n} \alpha_i \, \vec{x}^{(i)}$.

Then we define the *linear*, *polynomial*, *radial*, and *sigmoid* kernel functions, $K : \mathbb{R}^d \times \mathbb{R}^d \longrightarrow \mathbb{R}$, for a pair of vectors $\vec{x}^{(i)}$ and $\vec{x}^{(j)}$

$$
\begin{aligned}
K_{lin}(\vec{x}^{(i)}, \vec{x}^{(j)}) &= \vec{x}^{(i)} \cdot \vec{x}^{(j)} \ , \\
K_{poly}(\vec{x}^{(i)}, \vec{x}^{(j)}) &= \left(r + \gamma\, \vec{x}^{(i)} \cdot \vec{x}^{(j)}\right)^{\ell} \ , \\
K_{rad}(\vec{x}^{(i)}, \vec{x}^{(j)}) &= e^{-\gamma |\vec{x}^{(i)} - \vec{x}^{(j)}|^2} \ , \\
K_{sig}(\vec{x}^{(i)}, \vec{x}^{(j)}) &= \tanh\left(r + \gamma\, \vec{x}^{(i)} \cdot \vec{x}^{(j)}\right) \ ,
\end{aligned}
\tag{15}
$$

where γ in the radial kernel is a *positive constant* (hyperparameter), and for the rest of the kernels, γ and r are real-valued hyperparameters.

Now to define the boundary between classes, we generalize the function f in (2) to

$$\begin{aligned} f_{nonlin} : \ & \mathbb{R}^d \longrightarrow \mathbb{R} \\ f_{nonlin}(\vec{x}) &= b + \sum_{i=1}^{n} \alpha_i K(\vec{x}, \vec{x}^{(i)}) \ , \end{aligned} \tag{16}$$

where α_i are constants (We typically set $\alpha_i \neq 0$ only for the support vectors).

The nonlinear boundary between classes is then defined by the same equation as (3) except f is replaced by f_{nonlin} defined in (16)

$$\text{boundary between classes } = \left\{\vec{x} \in \mathbb{R}^d \mid f_{nonlin}(\vec{x}) = 0\right\} . \tag{17}$$

1. The linear kernel, K_{lin}, is equivalent to the support vector classifier where the boundary between different classes is introduced by a hyperplane.
2. The polynomial kernel, K_{poly}, is a polynomial function of degree ℓ (Note that for $\ell = 1$, K_{poly} reduces to K_{lin}). Note that the greater the dot product $\vec{x}^{(i)} \cdot \vec{x}^{(j)}$ is, the greater the value of the kernel function will be.

3. The radial kernel works in a local manner: When the Euclidean distance between the pair of points $\vec{x}^{(i)}$ and $\vec{x}^{(j)}$ is large, the value of the radial kernel function is negligible.
   - For a test point $\vec{x}^*$, the farther points to $\vec{x}^*$ do not play any role in predicting the class of $\vec{x}^*$
   - Recall that the class of $\vec{x}^*$ is determined by the sign of $f_{nonlin}(\vec{x}^*)$.
   - The radial kernel has a very *local behavior*, and only nearby observations to a test point play a role in determining the class of the test point.
4. The sigmoid kernel is used when one desires to restrict the kernel function to vary in a bounded interval (Recall that $\tanh x \in (-1, 1), \ \forall x \in \mathbb{R}$).

What if are there more than two classes?

- Unfortunately, the concept of separating hyperplanes does not lend itself naturally to more than two classes.
- However, a number of simple proposals have been made to extend SVM to the multinomial classification problems.
  - One-vs-one
  - One-vs-all

When there are more than two classes: One vs. One

- We consider all possible pairs of classes (*i.e.* $\frac{1}{2}K(K-1)$ distinct pairs of classes).
- For each pair, one constructs a binary SVM classifier.
- A test observation is classified using each of the $\frac{1}{2}K(K-1)$ SVM classifiers, and we tally the number of times that the test observation is assigned to each of the K classes.
- The final class of the test observation is determined through a voting process (*i.e.* the most frequently assigned class).

This is what scikit-learn does for multinomial classification.

When there are more than two classes: One vs. All

- We first choose a class k (from the total K classes).
- We then construct an SVM classifier to decide whether a test observation belongs to class k or to the rest of $K-1$ classes.
  - Note that this is a binary classification, and SVM can be readily applied.
- We can repeat this process for any of the K classes by constructing K SVM classifiers.
  - Note that each of the K SVM classifiers has its own separating hyperplane $f_k(\vec{x})$.
  - To assign the ultimate class of a test point $\vec{x}^*$, we calculate $f_k(\vec{x}^*)$ for $k = 1, 2, \cdots, K$.
  - The final class of $\vec{x}^*$ is the class that leads to the greatest value for $f_k(\vec{x}^*)$.

Some Disadvantages of SVMs

- SVM is computationally very expensive, and hence it is not a suitable classification algorithm for very large datasets.
- SVM does not perform well when there is a considerable amount of noise in the dataset.
- SVM typically underperforms when the number instances of the training dataset is less than the number of features.
- For situations where a probabilistic approach is more suitable, SVM cannot be helpful, as it takes a purely geometric approach towards the classification problem.