CSC321: Introduction to Neural Networks and Machine Learning

Lecture 24: Non-linear Support Vector Machines

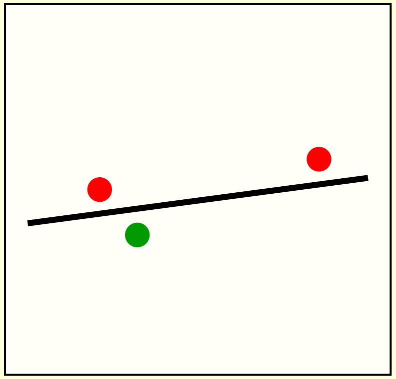
Geoffrey Hinton

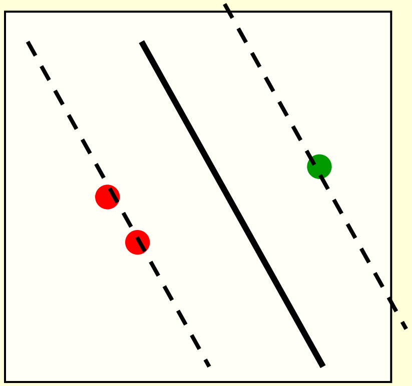
The story so far

- If we use a large set of non-adaptive features, we can often make the two classes linearly separable.
  - But if we just fit any old separating plane, it will not generalize well to new cases.
- If we fit the separating plane that maximizes the margin (the minimum distance to any of the data points), we will get much better generalization.
  - Intuitively, we are squeezing out all the surplus capacity that came from using a high-dimensional feature space.
- This can be justified by a whole lot of clever mathematics which shows that
  - large margin separators have lower VC dimension.
  - models with lower VC dimension have a smaller gap between the training and test error rates.

Why do large margin separators have lower VC dimension?

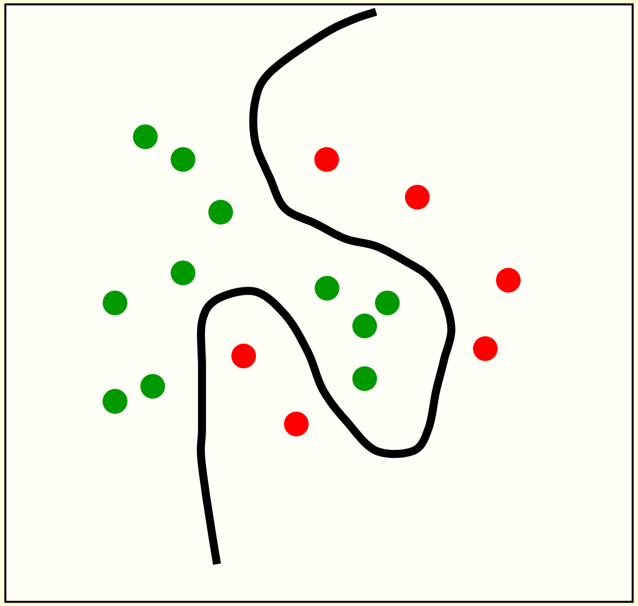
- Consider a set of N points that all fit inside a unit hypercube.
- If the number of dimensions is bigger than N-2, it is easy to find a separating plane for any labeling of the points.
  - – So the fact that there is a separating plane doesn’t tell us much. It like putting a straight line through 2 data points.
- But there is unlikely to be a separating plane with a margin that is big
  - – If we find such a plane its unlikely to be a coincidence. So it will probably apply to the test data too.





How to make a plane curved

- Fitting hyperplanes as separators is mathematically easy.
  - The mathematics is linear.
- By replacing the raw input variables with a much larger set of features we get a nice property:
  - A planar separator in the high-dimensional space of feature vectors is a curved separator in the low dimensional space of the raw input variables.



A planar separator in a 20-D feature space projected back to the original 2-D space

A potential problem and a magic solution

- If we map the input vectors into a very high-dimensional feature space, surely the task of finding the maximum-margin separator becomes computationally intractable?
  - The mathematics is all linear, which is good, but the vectors have a huge number of components.
  - So taking the scalar product of two vectors is very expensive.
- The way to keep things tractable is to use “the kernel trick”
- The kernel trick makes your brain hurt when you first learn about it, but its actually very simple.

What the kernel trick achieves

- All of the computations that we need to do to find the maximum-margin separator can be expressed in terms of scalar products between pairs of datapoints (in the high-dimensional feature space).
- These scalar products are the only part of the computation that depends on the dimensionality of the high-dimensional space.
  - So if we had a fast way to do the scalar products we would’nt have to pay a price for solving the learning problem in the high-D space.
- The kernel trick is just a magic way of doing scalar products a whole lot faster than is possible.

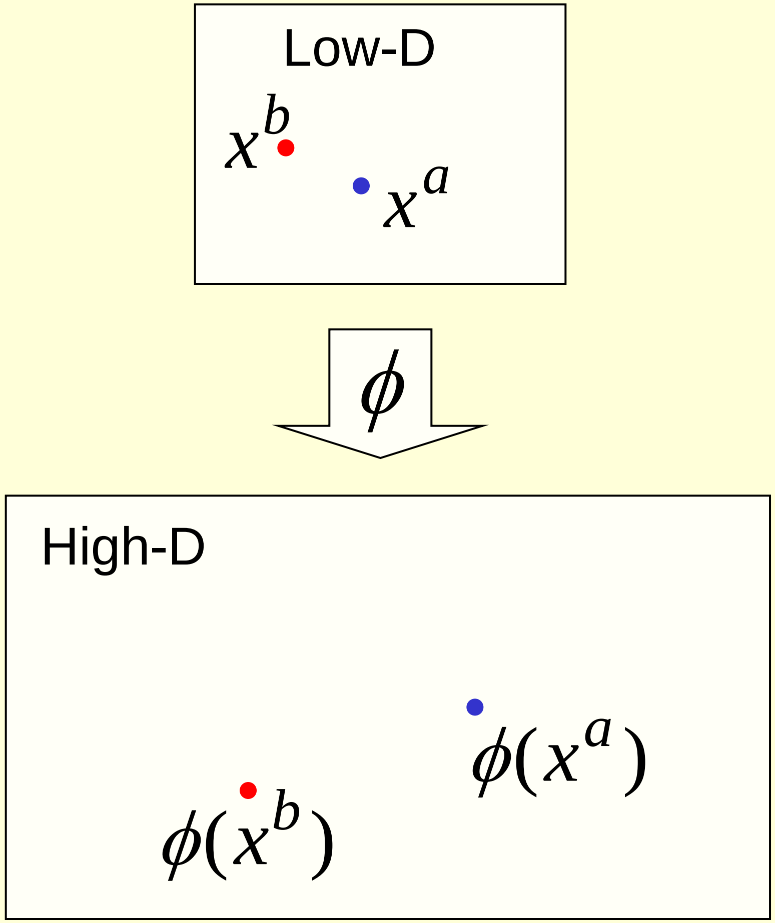
The kernel trick

- For many mappings from a low-D space to a high-D space, there is a simple operation on two vectors in the low-D space that can be used to compute the scalar product of their two images in the high-D space.

$$K(x^a, x^b) = \phi(x^a).\phi(x^b)$$

Letting the kernel do the work

doing the scalar product in the obvious way



Dealing with the test data

- If we choose a mapping to a high-D space for which the kernel trick works, we do not have to pay a computational price for the high-dimensionality when we find the best hyper-plane.
  - We cannot express the hyperplane by using its normal vector in the high-dimensional space because this vector would have a huge number of components.
  - Luckily, we can express it in terms of the support vectors.
- But what about the test data. We cannot compute the scalar product $\mathbf{w} \, . \, \phi(\mathbf{x})$ because its in the high-D space.

Dealing with the test data

- We need to decide which side of the separating hyperplane a test point lies on and this requires us to compute a scalar product.
- We can express this scalar product as a weighted average of scalar products with the stored support vectors
  - This could still be slow if there are a lot of support vectors .

The classification rule

- The final classification rule is quite simple:

$$bias + \sum_{s \, \varepsilon \, SV} w_s K(x^{test}, x^s) \quad > \; 0$$

↑

The set of
support vectors

- All the cleverness goes into selecting the support vectors that maximize the margin and computing the weight to use on each support vector.
- We also need to choose a good kernel function and we may need to choose a lambda for dealing with non-separable cases.

Some commonly used kernels

Polynomial: $K(\mathbf{x},\mathbf{y}) = (\mathbf{x}.\mathbf{y} + 1)^{p}$

Gaussian radial basis function $K(\mathbf{x},\mathbf{y}) = e^{-\|\mathbf{x}-\mathbf{y}\|^{2}/2\sigma^{2}}$

Parameters that the user must choose

Neural net: $K(\mathbf{x},\mathbf{y}) = \tanh(k\,\mathbf{x}.\mathbf{y} - \delta)$

For the neural network kernel, there is one “hidden unit” per support vector, so the process of fitting the maximum margin hyperplane decides how many hidden units to use. Also, it may violate Mercer’s condition.

Performance

- Support Vector Machines work very well in practice.
  - The user must choose the kernel function and its parameters, but the rest is automatic.
  - The test performance is very good.
- They can be expensive in time and space for big datasets
  - The computation of the maximum-margin hyper-plane depends on the square of the number of training cases.
  - We need to store all the support vectors.
- SVM's are very good if you have no idea about what structure to impose on the task.
- The kernel trick can also be used to do PCA in a much higher-dimensional space, thus giving a non-linear version of PCA in the original space (PCA will be explained later)