COMP 652: Machine Learning

Lecture 13

Today

- Nonseparable data
- Feature expansions and the kernel trick

Recall one version of the linear SVM problem

- Recall, we formulated the quadratic program (QP) for finding the maximum-margin perceptron (a.k.a. linear SVM) as:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) \geq 1 \end{aligned}$$

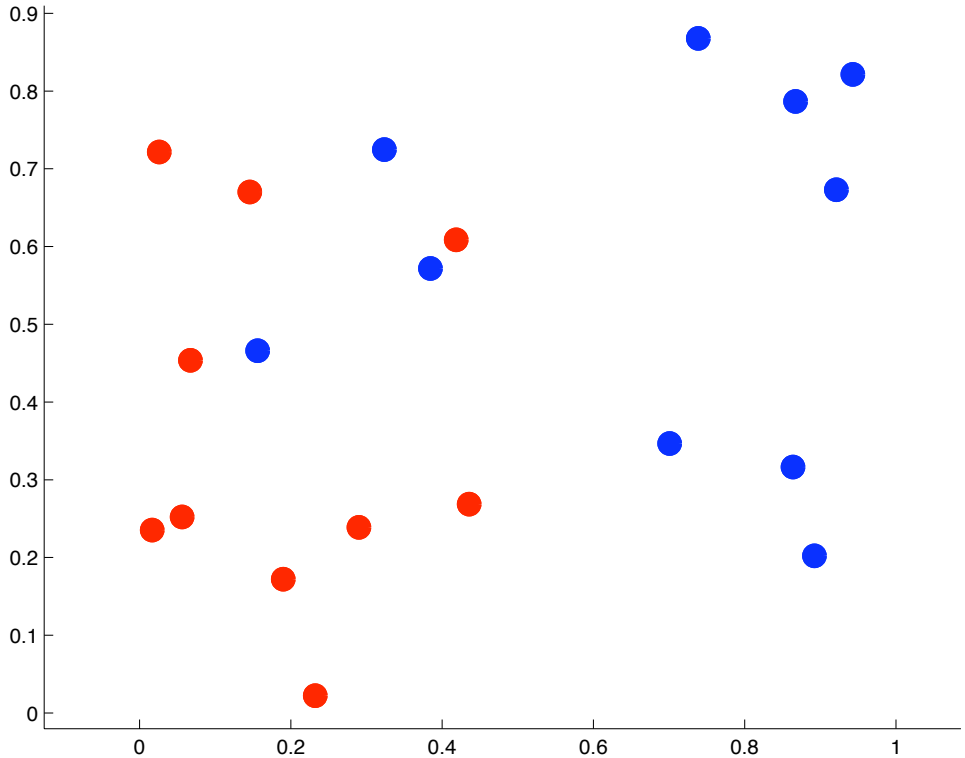
- The margin is $2M$ where $M = 1/\|\mathbf{w}\|$.

- The constraints, $y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) \geq 1$, ensure that each example is on the right side of the decision boundary at distance at least M.

- What if the data is not linearly separable?

Non-linearly separable data

- For example, consider this data set:



- If we try to use the previous formulation of the problem, the optimization package will tell us something like "no feasible solution". That is, the constraint cannot all be satisfied.

Recall one version of the linear SVM problem

- Recall, we formulated the quadratic program (QP) for finding the maximum-margin perceptron (a.k.a. linear SVM) as:

$$\begin{aligned} \text{min} \quad & \|\mathbf{w}\|^2 \\ \text{w.r.t.} \quad & \mathbf{w}, w_0 \\ \text{s.t.} \quad & y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) \geq 1 \end{aligned}$$

- The margin is $2M$ where $M = 1/\|\mathbf{w}\|$.

- The constraints, $y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) \geq 1$, ensure that each example is on the right side of the decision boundary at distance at least M.

- What if the data is not linearly separable?
  - Relax the constraints somehow
  - Try a different feature space, in which the data may be separable. (Maybe better for generalization too.)

Relaxing the constraints

- Given $\mathbf{w}, w_0$, an example $(\mathbf{x_i}, y_i)$ as at least distance $M = 1/\|\mathbf{w}\|$ on the right side of the margin if:

$$y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) \geq 1$$

- If we can't achieve that, suppose we require:

$$y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) \geq 1 - \zeta_i$$

  where $\zeta_i \geq 0$.

- How can we interpret ζ_i?
  - If $\zeta_i = 0$, then the original distance constraint is satisfied.
  - If $\zeta_i \in (0, 1)$, then the point is on the correct side of the decision boundary, but not as far as it should be.
  - If $\zeta_i = 1$, then the point is on the decision boundary.
  - If $\zeta_i > 1$ then the point is on the wrong side of the decision boundary.

Reformulating the problem

- Such constraints can always be satisfied – but we must penalize the ζ_i in order to encourage satisfaction of the original constraints.
- This suggests the optimization problem:

$$\begin{array}{rl} \text{min} & \|\mathbf{w}\|^2 + C \sum_i \zeta_i \\ \text{w.r.t.} & \mathbf{w}, w_0, \zeta_i \\ \text{s.t.} & y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) \geq (1 - \zeta_i) \\ & \zeta_i \geq 0 \end{array}$$

  where $C > 0$ is a user-chosen cost associated with constraint violation.
- Intuitively, this formulation tries to maximize the margin without violating the constraints too much.
- This formulation can also be solved by quadratic programming.

Solution

- As in the separable case, the solution for $\mathbf{w}$ is of the form

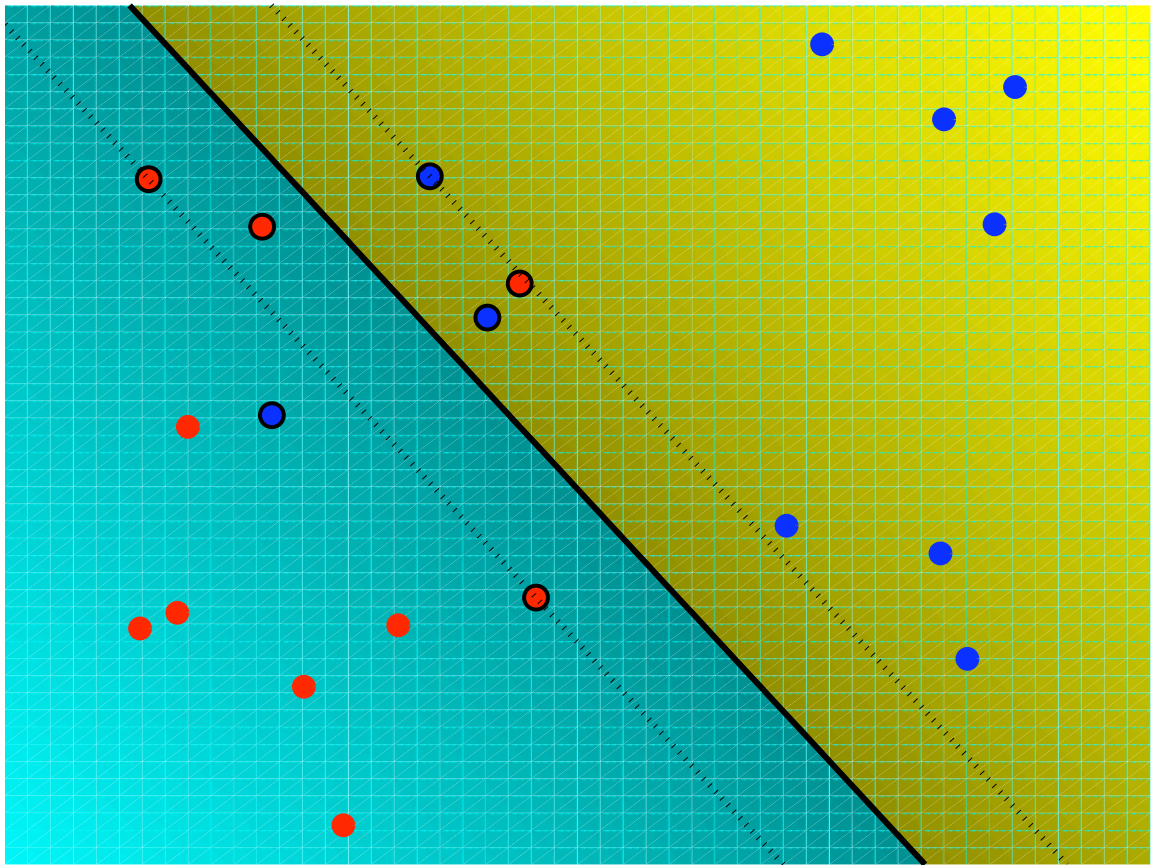
$$\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \mathbf{x_i}$$

  where the α_i are non-negative weighting factors. Thus,

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \mathbf{x_i} \cdot \mathbf{x} + w_0\right)$$

- α_i is positive if and only if $\mathbf{x_i}$ lies on the edge of the margin *or* on the wrong side of the margin. That is, if $y_i(\mathbf{w} \cdot \mathbf{x_i} + w_0) \leq 1$.
- Such an $\mathbf{x_i}$ is a *support vector*.

Example



Feature expansions and nonlinear decision boundaries

- Linear SVMs always produce a decision boundary that is a hyperplane. For some data this is not appropriate.
- One way of getting a nonlinear decision boundary in the input space is to find a linear decision boundary in an expanded space. (As we've seen, e.g., for polynomial regression.)
- Thus, $\mathbf{x_i}$ is replaced by $\phi(\mathbf{x_i})$, where ϕ is called a "feature mapping"

- What's the big deal?

Feature-expanded optimization problem

Replacing $\mathbf{x_i}$ with $\phi(\mathbf{x_i})$, the optimization problem to find $\mathbf{w}$ and w_0 becomes:

$$\begin{array}{rl} \text{min} & \|\mathbf{w}\|^2 + C\sum_i \zeta_i \\ \text{w.r.t.} & \mathbf{w}, w_0, \zeta_i \\ \text{s.t.} & y_i(\mathbf{w}\cdot\phi(\mathbf{x_i}) + w_0) \geq (1-\zeta_i) \\ & \zeta_i \geq 0 \end{array}$$

or in dual form:

$$\begin{array}{rl} \text{max} & \sum_{i=1}^m \alpha_i - \frac{1}{2}\sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \phi(\mathbf{x_i})\cdot\phi(\mathbf{x_j}) \\ \text{w.r.t.} & \alpha_i \\ \text{s.t.} & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^m \alpha_i y_i = 0 \end{array}$$

Feature-expanded solution

- The optimal weights, in the expanded feature space, are $\mathbf{w} = \sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x_i})$.
- Classification is by the rule:

$$h_{\mathbf{w},w_0}(\mathbf{x}) = \text{sgn}\left(\sum_{i=1}^{m} \alpha_i y_i \phi(\mathbf{x_i}) \cdot \phi(\mathbf{x}) + w_0\right)$$

$\Rightarrow$ Observation: to solve the fitting problem in dual form and to make a classification prediction, we only ever need to computed dot-products of feature-expanded vectors.

Kernel functions

- Whenever a learning algorithm (such as SVMs) can be written in terms of dot-products, it can be generalized to kernels.
- A *kernel* is any function $K : \Re^n \times \Re^n \mapsto \Re$ which corresponds to a dot product for some feature mapping ϕ. That is, $K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2)$ for some ϕ.
- Conversely, by choosing feature mapping ϕ, we are implicitly choosing a Kernel function
- Recalling that $\phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) = \cos A \|\mathbf{x}_1\| \|\mathbf{x}_2\|$ where A is the angle between the vectors, a Kernel function can be thought of as a notion of *similarity*.

$\Rightarrow$ We can substitute other notions of similarity (as long as they are Kernel functions)

Example: Quadratic kernel

- Let $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$.
- Is this a Kernel? Yes:

$$K(\mathbf{x}, \mathbf{z}) = \left(\sum_{i=1}^{n} x_i z_i\right)\left(\sum_{j=1}^{n} x_j z_j\right) = \sum_{i,j \in \{1 \ldots n\}} x_i z_i x_j z_j = \sum_{i,j \in \{1 \ldots n\}} (x_i x_j)(z_i z_j)$$

- Hence, it is a kernel, with feature mapping:

$$\phi(\mathbf{x}) = \langle x_1^2,\ x_1 x_2,\ \ldots,\ x_1 x_n,\ x_2 x_1,\ x_2^2,\ \ldots,\ x_n^2 \rangle$$

- The feature vector includes all squares of elements and all cross terms.
- Note some redundancy among the features – that's OK.
- Fitting the SVM puts a weight on each of these features.

Polynomial kernels

- More generally, $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^d$ is a Kernel, for any positive integer d.

$$K(\mathbf{x}, \mathbf{z}) = \left(\sum_{i=1}^{n} \mathbf{x}_i \mathbf{z}_i \right)^d$$

- If we expanded the sum above in the obvious way, we get n^d terms.
- Thus, this K is a kernel with a length n^d feature expansion.
- Terms are monomials (products of x_i) with total power equal to d.
- E.g., if $n = 2$ and $d = 3$, the (most direct) feature mapping is

$$\phi(\mathbf{x}) = \langle x_1^3, x_1^2 x_2, x_1^2 x_2, x_1 x_2^2, x_1^2 x_2, x_1 x_2^2, x_1 x_2^2, x_2^3 \rangle$$

A "curse of dimensionality"

- Suppose we want to use a polynomial Kernel $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^d = \phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ to fit an SVM.
- If we try to solve the SVM in primal form, we have to work with some very big feature vectors:

$$\begin{array}{rl} \text{min} & \|\mathbf{w}\|^2 + C \sum_i \zeta_i \\ \text{w.r.t.} & \mathbf{w}, w_0, \zeta_i \\ \text{s.t.} & y_i(\mathbf{w} \cdot \phi(\mathbf{x_i}) + w_0) \geq (1 - \zeta_i) \\ & \zeta_i \geq 0 \end{array}$$

- Even making a prediction is computationally intensive:

$$h_{\mathbf{w}, w_0}(\mathbf{x}) = \text{sgn}(\mathbf{w} \cdot \phi(\mathbf{x}) + w_0)$$

The "kernel trick"

- If we work with the dual, we don't actually have to ever compute the feature mapping ϕ. We just have to compute the similarity K.
- That is, we can solve the dual for the α_i:

$$\begin{array}{rl} \text{max} & \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j K(\mathbf{x_i}, \mathbf{x_j}) \\ \text{w.r.t.} & \alpha_i \\ \text{s.t.} & 0 \leq \alpha_i \leq C \\ & \sum_{i=1}^m \alpha_i y_i = 0 \end{array}$$

- Keeping in mind that the optimal weights are of the form $\mathbf{w} = \sum_{i=1}^m \alpha_i y_i \phi(\mathbf{x_i})$, we can make predictions as:

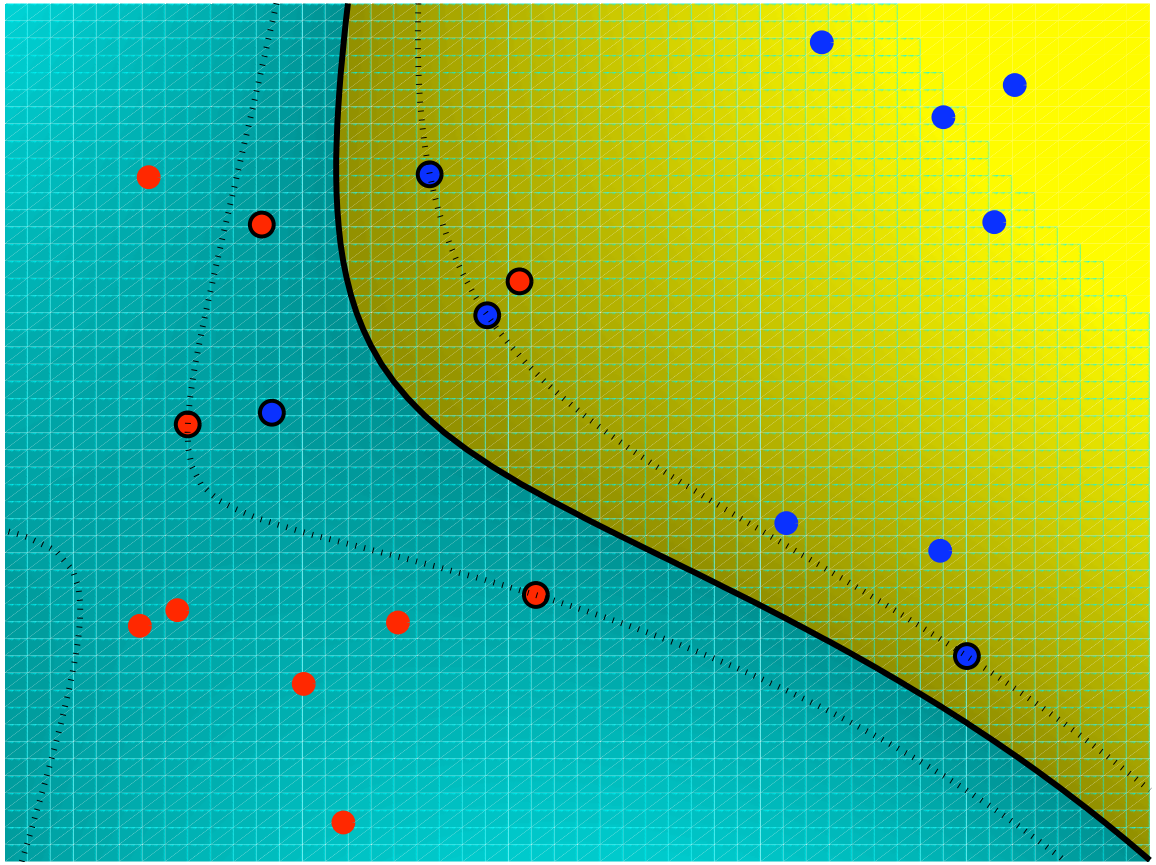
$$\begin{aligned} h_{\mathbf{w},w_0}(\mathbf{x}) &= \text{sgn}\left(\left(\sum_{i=1}^m \alpha_i y_i \phi(\mathbf{x_i})\right) \cdot \phi(\mathbf{x}) + w_0\right) \\ &= \text{sgn}\left(\sum_{i=1}^m \alpha_i y_i K(\mathbf{x_i}, \mathbf{x}) + w_0\right) \end{aligned}$$

- Often, $K(\cdot, \cdot)$ can be evaluated in $O(n)$ time—a big savings!

Some other (fairly generic) Kernel functions

- $K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})^d$ – feature expansion has all monomial terms of $\leq d$ total power.
- Radial basis/Gaussian $K(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2/s)$ – has infinite-dimensional feature expansion!
- Sigmoidal $K(\mathbf{x}, \mathbf{z}) = \tanh(c_1 \mathbf{x} \cdot \mathbf{z} + c_2)$

Example: Gaussian kernel



Application: Text classification

(Joachims, 1998)

- Evaluated several methods, including SVMs, on a suite of text classification problems
- Words were stemmed (e.g. learn, learning, learned $\rightarrow$ learn)
- Nondiscriminative stopwords and words occurring < 3 times ignored
- Of remaining words, considered a binary presence-absence feature
- 1000 features with greatest information gain retained, others discarded
- Each feature scaled by "inverse document frequency":

$$\log \frac{\#\text{ docs}}{\#\text{ docs with word } i}$$

Results

					SVM (poly) $d =$					SVM (rbf) $\gamma =$			
	Bayes	Rocchio	C4.5	k-NN	1	2	3	4	5	0.6	0.8	1.0	1.2
earn	95.9	96.1	96.1	97.3	98.2	98.4	**98.5**	98.4	98.3	**98.5**	98.5	98.4	98.3
acq	91.5	92.1	85.3	92.0	92.6	94.6	**95.2**	95.2	95.3	95.0	95.3	95.3	**95.4**
money-fx	62.9	67.6	69.4	78.2	66.9	72.5	75.4	74.9	**76.2**	74.0	75.4	**76.3**	75.9
grain	72.5	79.5	89.1	82.2	91.3	93.1	**92.4**	91.3	89.9	**93.1**	91.9	91.9	90.6
crude	81.0	81.5	75.5	85.7	86.0	87.3	88.6	**88.9**	87.8	**88.9**	89.0	88.9	88.2
trade	50.0	77.4	59.2	77.4	69.2	75.5	76.6	77.3	**77.1**	76.9	78.0	**77.8**	76.8
interest	58.0	72.5	49.1	74.0	69.8	63.3	67.9	73.1	**76.2**	74.4	75.0	**76.2**	76.1
ship	78.7	83.1	80.9	79.2	82.0	85.4	86.0	**86.5**	86.0	**85.4**	86.5	87.6	87.1
wheat	60.6	79.4	85.5	76.6	83.1	84.5	85.2	**85.9**	83.8	**85.2**	85.9	85.9	85.9
corn	47.3	62.2	87.7	77.9	86.0	86.5	85.3	**85.7**	83.9	**85.1**	85.7	85.7	84.5
microavg.	**72.0**	**79.9**	**79.4**	**82.3**	84.2	85.1	85.9	86.2	85.9	86.4	86.5	86.3	86.2
					combined: **86.0**					combined: **86.4**			

Figure 4: Precision/recall-breakeven point on the ten most frequent Reuters categories and microaveraged performance over all Reuters categories. k-NN, Rocchio, and C4.5 achieve highest performance at 1000 features (with $k = 30$ for k-NN and $\beta = 1.0$ for Rocchio). Naive Bayes performs best using all features.

Kernels for everyone!

A lot of SVM research has to do with defining new kernels functions, suitable to particular tasks / kinds of input objects, plus proving that they are indeed kernels.

- Information diffusion kernels (Lafferty and Lebanon, 2002)
- Diffusion kernels on graphs (Kondor and Jebara 2003)
- String kernels for text classification (Lodhi et al, 2002)
- String kernels for protein classification (e.g., Leslie et al, 2002)
- ... and others!

String kernels

- Very important for DNA matching, text classification, ...
- Example: in DNA matching, we use a sliding window of length k over the two strings that we want to compare
- The window is of a given size, and inside we can do various things:
  - Count exact matches
  - Weigh mismatches based on how bad they are
  - Count certain markers, e.g. AGT
- The kernel is the sum of these similarities over the two sequences
- How do we prove this is a kernel?

Establishing "kernelhood"

- Suppose someone hands you a function K. How do you know that it is a kernel?
- More precisely, given a function $K : \Re^n \times \Re^n \rightarrow \Re$, under what conditions can $K(\mathbf{x}, \mathbf{z})$ be written as a dot product $\phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ for some feature mapping ϕ?

Kernel matrix

- Suppose we have an arbitrary set of input vectors $\mathbf{x_1}, \mathbf{x_2}, \ldots \mathbf{x_m}$
- The **kernel matrix** K corresponding to kernel function K is an $m \times m$ matrix such that $K_{ij} = K(\mathbf{x_i}, \mathbf{x_j})$ (notation is overloaded on purpose).
- What properties does the kernel matrix K have?
- Claims:

  1. K is symmetric
  2. K is positive semidefinite

- Note that these claims are consistent with the intuition that K is a "similarity" measure (and will be true regardless of the data)

Proving the first claim

If K is a valid kernel, then the kernel matrix is symmetric

$$K_{ij} = \phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_j}) = \phi(\mathbf{x_j}) \cdot \phi(\mathbf{x_i}) = K_{ji}$$

Proving the second claim

If K is a valid kernel, then the kernel matrix is positive semidefinite
Proof: Consider an arbitrary vector $\mathbf{z}$

$$\begin{aligned}
\mathbf{z}^T K \mathbf{z} &= \sum_i \sum_j z_i K_{ij} z_j = \sum_i \sum_j z_i \left(\phi(\mathbf{x_i}) \cdot \phi(\mathbf{x_j})\right) z_j \\
&= \sum_i \sum_j z_i \left(\sum_k \phi_k(\mathbf{x_i}) \phi_k(\mathbf{x_j}) \right) z_j \\
&= \sum_k \sum_i \sum_j z_i \phi_k(\mathbf{x_i}) \phi_k(\mathbf{x_j}) z_j \\
&= \sum_k \left(\sum_i z_i \phi_k(\mathbf{x_i}) \right)^2 \geq 0
\end{aligned}$$

Mercer's theorem

- We have shown that if K is a kernel function, then for any data set, the corresponding kernel matrix K defined such that $K_{ij} = K(\mathbf{x_i}, \mathbf{x_j})$ is symmetric and positive semidefinite
- Mercer's theorem states that the reverse is also true!
  Given a function $K : \Re^n \times \Re^n \rightarrow \Re$, K is a kernel *if and only if*, for any data set, the corresponding kernel matrix is positive and semidefinite
- The reverse direction of the proof is much harder
- This result gives us a way to *check* is a given function is a kernel, by checking these two properties of its kernel matrix.

Symptoms of overfitting in SVMs

- Low margin
- Large number of support vectors
- If you look at the kernel matrix (also called Gram matrix) it is almost diagonal
  - This means each point is only similar to itself
  - So the kernel used is not really adequate (it does not generalize over the data

Getting this to work in practice

- Choice of kernel
- Choice of regularization parameter
- These control the overfitting: always check the margin/number of support vectors or do cross-validation

Interpretability

- More interpretable than neural nets if you look at the machine and the misclassifications
- E.g. Ovarian cancer data (Haussler) - 31 tissue samples of 3 classes, misclassified examples wronlgy labelled
- *No* biological plausibility

Complexity

- Quadratic programming is expensive in the number of training examples
- Platt's algorithm is quite fast though
- Best packages can handle 10,000-20,000 instances, not more
- On the other hand, number of attributes can be very high (strength compared to neural nets)
- Evaluating a SVM requires taking the dot product of an instance with the support vectors, which is *slow if there are a lot of support vectors*.
- Dictionary methods attempt to select data points and/or support vectors

Applications of SVMs

- The biggest strength of SVMs is dealing with large numbers of features (which relies on the kernel trick and the control of overfitting)
- Many successful applications in:
  - Text classification (e.g. Joachims, 1998)
  - Object detection (e.g. Osuna, Freund and Girosi, 1997)
  - Object recognition (e.g. Pontil and Verri, 1998)
  - Bioinformatics (e.g. lee et al, 2002)
- SVMs are considered by many the state-of-the art approach to classification
- Experimentally, SVMs and neural nets are roughly tied based on evidence to date, each has its own preferred applications

Summary of Perceptrons & Support vector machines

- A perceptron is basically just a linear decision boundary
- For linearly separable data, the gradient-like perceptron training rule can find weights that separate the data
- A linear support vector machine maximizes the margin of the decision boundary
- We can modify the SVM optimization to allow for misclassified examples
- We can define other notions of similarity — Kernels — the allow nonlinear decision boundaries (in the original space)
- The kernel trick allows us to work efficiently with Kernels even if they have large feature expansions — because we only ever need to compute the Kernel, not the actual features