
- SVMs for non-linearly separable data
- The kernel trick
- Mercer’s theorem
- Kernelizing other machine learning methods
  - Kernelized linear regression
  - Kernelized logistic regression
- If we have time: Gaussian Processes
Recall: Linear support vector machines

- Classification method for linearly separable data
- Designed to maximize the margin of the data: the minimum distance between any instance and the decision boundary
- Last time: phrase this as a quadratic program, and solve the dual
- Solution can be represented as a linear combination of a set of instances (support vectors)
- Both the set of support vectors and their coefficients are obtained automatically as the solution to the quadratic program.
- If the data is not linearly separable, or if we want to avoid overfitting: soft margins
Recall: Soft margin

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

\[ y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 \]

- The soft margin approach relaxes these constraints:

\[ y_i(\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq 1 - \zeta_i \text{ where } \zeta_i \geq 0 \]

- How can we interpret $\zeta_i$?
  - 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.
Recall: Soft margin SVMs

- Optimization problem:
  \[
  \min_{\mathbf{w}, w_0, \zeta_i} ||\mathbf{w}||^2 + C \sum_i \zeta_i \\
  \text{s.t. } y_i (\mathbf{w} \cdot \mathbf{x}_i + w_0) \geq (1 - \zeta_i) \\
  \zeta_i \geq 0
  \]

  where the first term is the margin, and the second term penalizes constraint violations.

- \( C > 0 \) is a user-chosen cost associated with constraint violation, and help to control overfitting.

- As in the separable case, the solution is of the form:
  \[
  h_{\mathbf{w}, w_0}(\mathbf{x}) = \text{sgn} \left( \sum_{i=1}^{m} \alpha_i y_i (\mathbf{x}_i \cdot \mathbf{x}) + w_0 \right)
  \]
A linear boundary might be too simple to capture the class structure.

One way of getting a nonlinear decision boundary in the input space is to find a linear decision boundary in an expanded space (e.g., for polynomial regression.)

Thus, \( x_i \) is replaced by \( \phi(x_i) \), where \( \phi \) is called a feature mapping.
Margin optimization in feature space

- Replacing $x_i$ with $\phi(x_i)$, the optimization problem to find $w$ and $w_0$ becomes:
  \[
  \min_{w, w_0, \zeta} \|w\|^2 + C \sum \zeta_i \\
  \text{w.r.t. } w, w_0, \zeta_i \\
  \text{s.t.} \ y_i(w \cdot \phi(x_i) + w_0) \geq (1 - \zeta_i) \\
  \zeta_i \geq 0
  \]

- Dual form:
  \[
  \max_{\alpha_i} \sum_{i=1}^m \alpha_i - \frac{1}{2} \sum_{i,j=1}^m y_i y_j \alpha_i \alpha_j \phi(x_i) \cdot \phi(x_j) \\
  \text{w.r.t. } \alpha_i \\
  \text{s.t.} \ 0 \leq \alpha_i \leq C \\
  \sum_{i=1}^m \alpha_i y_i = 0
  \]
Feature space solution

• The optimal weights, in the expanded feature space, are
  \[ w = \sum_{i=1}^{m} \alpha_i y_i \phi(x_i). \]

• Classification of an input \( x \) is given by:
  \[
  h_{w,w_0}(x) = \text{sgn} \left( \sum_{i=1}^{m} \alpha_i y_i \phi(x_i) \cdot \phi(x) + w_0 \right)
  \]

⇒ Note that to solve the SVM optimization problem in dual form and to make a prediction, we only ever need to compute \textit{dot-products of feature vectors}. 

COMP-652, Lecture 10 - October 14, 2009
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 $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ which corresponds to a dot product for some feature mapping $\phi$:

  $$K(x_1, x_2) = \phi(x_1) \cdot \phi(x_2) \text{ for some } \phi.$$

- Conversely, by choosing feature mapping $\phi$, we implicitly choose a kernel function.

- Recall that $\phi(x_1) \cdot \phi(x_2) = \cos \angle(x_1, x_2)$ where $\angle$ denotes the angle between the vectors, so a kernel function can be thought of as a notion of similarity.
Example: Quadratic kernel

- Let $K(x, z) = (x \cdot z)^2$.
- Is this a kernel?

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

$$= \sum_{i,j \in \{1\ldots n\}} (x_i x_j) (z_i z_j)$$

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

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

Feature vector includes all squares of elements and all cross terms.
- Note that computing $\phi$ takes $O(n^2)$ but computing $K$ takes only $O(n)$!
Polynomial kernels

- More generally, \( K(x, z) = (x \cdot z)^d \) is a kernel, for any positive integer \( d \):
  \[
  K(x, z) = \left( \sum_{i=1}^{n} x_i z_i \right)^d
  \]
- If we expanded the sum above in the obvious way, we get \( n^d \) terms (i.e. feature expansion)
- Terms are monomials (products of \( x_i \)) with total power equal to \( d \).
- If we use the primal form of the SVM, each of these will have a weight associated with it!
- **Curse of dimensionality**: it is very expensive both to optimize and to predict with an SVM in primal form
- However, **evaluating the dot-product of any two feature vectors can be done using \( K \) in \( O(n) \)!
The “kernel trick”

• If we work with the dual, we do not actually have to ever compute the feature mapping \( \phi \). We just have to compute the similarity \( K \).

• That is, we can solve the dual for the \( \alpha_i \):\
  \[
  \max_{\alpha_i} \sum_{i=1}^{m} \alpha_i - \frac{1}{2} \sum_{i,j=1}^{m} y_i y_j \alpha_i \alpha_j K(x_i, x_j) \\
  \text{w.r.t.} \quad \alpha_i \\
  \text{s.t.} \quad 0 \leq \alpha_i \leq C \\
  \sum_{i=1}^{m} \alpha_i y_i = 0
  \]

• The class of a new input \( x \) is computed as:

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

• Often, \( K(\cdot, \cdot) \) can be evaluated in \( O(n) \) time—a big savings!
Some other (fairly generic) kernel functions

- $K(x, z) = (1 + x \cdot z)^d$ – feature expansion has all monomial terms of $\leq d$ total power.
- Radial basis/Gaussian kernel:

$$K(x, z) = \exp\left(-\frac{||x - z||^2}{2\sigma^2}\right)$$

The kernel has an infinite-dimensional feature expansion, but dot-product can still be computed in $O(n)$!
- Sigmoidal kernel:

$$K(x, z) = \tanh(c_1 x \cdot z + c_2)$$
Example: Gaussian kernel

Note the non-linear decision boundary
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 → 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

<table>
<thead>
<tr>
<th></th>
<th>Bayes</th>
<th>Rocchio</th>
<th>C4.5</th>
<th>k-NN</th>
<th>SVM (poly) (d=1)</th>
<th>SVM (poly) (d=2)</th>
<th>SVM (poly) (d=3)</th>
<th>SVM (poly) (d=4)</th>
<th>SVM (poly) (d=5)</th>
<th>SVM (poly) (\gamma=0.6)</th>
<th>SVM (poly) (\gamma=0.8)</th>
<th>SVM (poly) (\gamma=1.0)</th>
<th>SVM (poly) (\gamma=1.2)</th>
</tr>
</thead>
<tbody>
<tr>
<td>earn</td>
<td>95.9</td>
<td>96.1</td>
<td>96.1</td>
<td>97.3</td>
<td>98.2</td>
<td>98.4</td>
<td>98.5</td>
<td>98.4</td>
<td>98.3</td>
<td>98.5</td>
<td>98.5</td>
<td>98.4</td>
<td>98.3</td>
</tr>
<tr>
<td>acq</td>
<td>91.5</td>
<td>92.1</td>
<td>85.3</td>
<td>92.0</td>
<td>92.6</td>
<td>94.6</td>
<td>95.2</td>
<td>95.2</td>
<td>95.2</td>
<td>95.0</td>
<td>95.3</td>
<td>95.3</td>
<td>95.4</td>
</tr>
<tr>
<td>money-fx</td>
<td>62.9</td>
<td>67.6</td>
<td>69.4</td>
<td>78.2</td>
<td>66.6</td>
<td>72.5</td>
<td>75.4</td>
<td>74.9</td>
<td>76.2</td>
<td>74.0</td>
<td>75.4</td>
<td>76.3</td>
<td>75.9</td>
</tr>
<tr>
<td>grain</td>
<td>72.5</td>
<td>79.5</td>
<td>83.1</td>
<td>89.9</td>
<td>91.3</td>
<td>93.1</td>
<td>92.4</td>
<td>91.3</td>
<td>89.9</td>
<td>93.1</td>
<td>91.9</td>
<td>91.9</td>
<td>90.6</td>
</tr>
<tr>
<td>crude</td>
<td>81.0</td>
<td>81.5</td>
<td>75.5</td>
<td>85.7</td>
<td>86.6</td>
<td>87.3</td>
<td>88.6</td>
<td>88.0</td>
<td>87.8</td>
<td>88.9</td>
<td>89.0</td>
<td>88.9</td>
<td>88.2</td>
</tr>
<tr>
<td>trade</td>
<td>50.0</td>
<td>77.4</td>
<td>59.2</td>
<td>77.4</td>
<td>66.2</td>
<td>75.5</td>
<td>76.6</td>
<td>77.3</td>
<td>77.1</td>
<td>76.9</td>
<td>78.0</td>
<td>77.8</td>
<td>76.8</td>
</tr>
<tr>
<td>interest</td>
<td>58.0</td>
<td>72.5</td>
<td>49.1</td>
<td>74.0</td>
<td>66.8</td>
<td>63.3</td>
<td>67.9</td>
<td>73.1</td>
<td>76.2</td>
<td>74.4</td>
<td>75.0</td>
<td>76.2</td>
<td>76.1</td>
</tr>
<tr>
<td>ship</td>
<td>78.7</td>
<td>83.1</td>
<td>80.9</td>
<td>79.2</td>
<td>82.4</td>
<td>85.4</td>
<td>86.0</td>
<td>86.5</td>
<td>86.0</td>
<td>85.4</td>
<td>86.5</td>
<td>87.6</td>
<td>87.1</td>
</tr>
<tr>
<td>wheat</td>
<td>60.6</td>
<td>79.4</td>
<td>85.5</td>
<td>76.6</td>
<td>83.1</td>
<td>84.5</td>
<td>85.2</td>
<td>85.9</td>
<td>83.8</td>
<td>85.2</td>
<td>85.9</td>
<td>85.5</td>
<td>85.9</td>
</tr>
<tr>
<td>ccm</td>
<td>47.3</td>
<td>62.2</td>
<td>87.7</td>
<td>77.9</td>
<td>86.0</td>
<td>86.5</td>
<td>85.3</td>
<td>85.7</td>
<td>83.9</td>
<td>85.1</td>
<td>85.7</td>
<td>85.7</td>
<td>84.5</td>
</tr>
<tr>
<td>microavg.</td>
<td>72.0</td>
<td>79.9</td>
<td>79.4</td>
<td>82.3</td>
<td>84.2</td>
<td>85.1</td>
<td>85.9</td>
<td>86.2</td>
<td>85.9</td>
<td>86.4</td>
<td>86.5</td>
<td>86.3</td>
<td>86.2</td>
</tr>
</tbody>
</table>

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.

SVMs are better than any other classifier
Getting SVMs to work in practice

- Two important choices:
  - Kernel (and kernel parameters)
  - Regularization parameter $C$
- Together, these control overfitting: always do an internal parameter search, using a validation set!
- Overfitting symptoms:
  - Low margin
  - Large fraction of instances are support vectors
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 wrongly labelled
- But no biological plausibility!
- Hard to interpret if the percentage of instances that are recruited as support vectors is high
Complexity

- Quadratic programming is expensive in the number of training examples
- Platt’s SMO algorithm is quite fast though, and other fancy optimization approaches are available
- Best packages can handle $20,000+$ instances, but not more than $100,000$
- On the other hand, number of attributes can be very high (strength compared to neural nets)
- Evaluating a SVM is *slow if there are a lot of support vectors*.
- Dictionary methods attempt to select a subset of the data on which to train.
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
Kernels beyond SVMs

A lot of current research has to do with defining new kernels functions, suitable to particular tasks / kinds of input objects

- 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!
Example: 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 $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$, under what conditions can $K(x, z)$ be written as a dot product $\phi(x) \cdot \phi(z)$ for some feature mapping $\phi$?
• We want a general recipe, which does not require explicitly defining $\phi$ every time
Kernel matrix

• Suppose we have an arbitrary set of input vectors $x_1, x_2, \ldots x_m$

• The *kernel matrix (or Gram matrix)* $K$ corresponding to kernel function $K$ is an $m \times m$ matrix such that $K_{ij} = K(x_i, x_j)$ (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

$$K_{ij} = \phi(x_i) \cdot \phi(x_j) = \phi(x_j) \cdot \phi(x_i) = K_{ji}$$
Proving the second claim

If $K$ is a valid kernel, then the kernel matrix is positive semidefinite.

Proof: Consider an arbitrary vector $z$

\[
\begin{align*}
z^T K z &= \sum_i \sum_j z_i K_{ij} z_j \\
&= \sum_i \sum_j z_i (\phi(x_i) \cdot \phi(x_j)) z_j \\
&= \sum_i \sum_j z_i \left( \sum_k \phi_k(x_i) \phi_k(x_j) \right) z_j \\
&= \sum_k \sum_i \sum_j z_i \phi_k(x_i) \phi_k(x_j) z_j \\
&= \sum_k \left( \sum_i z_i \phi_k(x_i) \right)^2 \geq 0
\end{align*}
\]
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 $K_{ij} = K(x_i, x_j)$ is symmetric and positive semidefinite.

• Mercer’s theorem states that the reverse is also true: Given a function $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$, $K$ is a kernel if and only if, for any data set, the corresponding kernel matrix is symmetric and positive semidefinite.

• The reverse direction of the proof is much harder.

• This result gives us a way to check if a given function is a kernel, by checking these two properties of its kernel matrix.

• Kernels can also be obtained by combining other kernels (see homework), or by learning from data.

• Kernel learning may suffer from overfitting (kernel matrix close to diagonal).
Kernelizing other machine learning algorithms

- Many other machine learning algorithms have a “dual formulation”, in which dot-products of features can be replaced with kernels.
- Two examples now:
  - Logistic regression
  - Linear regression
- Later: kernel PCA
Linear regression with feature vectors

- Find the weight vector $w$ which minimizes the (regularized) error function:
  \[ J(w) = \frac{1}{2}(\Phi w - y)^T(\Phi w - y) + \frac{\lambda}{2}w^T w \]
- The solution takes the form:
  \[ w = -\frac{1}{\lambda} \sum_{i=1}^{m} (w^T \phi(x_i) - y_i) \phi(x_i) = \sum_{i=1}^{m} a_i \phi(x_i) = \Phi^T a \]
  where $a$ is a vector of size $m$ (number of instances) with $a_i = -\frac{1}{\lambda}(w^T \phi(x_i) - y_i)$
- Main idea: use $a$ instead of $w$ as parameter vector
- Note that this is similar to re-formulating a weight vector in terms of a linear combination of instances, but we are not using the primal-dual mechanism in a literal sense
Re-writing the error function

• Instead of $J(w)$ we have $J(a)$:

$$J(a) = \frac{1}{2} a^T \Phi \Phi^T \Phi \Phi^T a - a^T \Phi \Phi^T y + \frac{1}{2} y^T y + \frac{\lambda}{2} a^T \Phi \Phi^T a$$

• Note that $\Phi \Phi^T = K$, the kernel matrix!

• Hence, we can re-write this as:

$$J(a) = \frac{1}{2} a^T K K A - A^T K y + \frac{1}{2} y^T y + \frac{\lambda}{2} a^T K a$$

• By setting the gradient to 0 we get:

$$a = (K + \lambda I_m)^{-1} y$$
Making predictions with dual-view regression

• For a new input \( x \), the prediction is:

\[
h(x) = w^T \phi(x) = a^T \Phi \phi(x) = k(x)^T (K + \lambda I_m)^{-1} y
\]

where \( k(x) \) is an \( m \)-dimensional vector, with the \( i \)th element equal to \( K(x, x_i) \)
• That is, the \( i \)th element has the similarity of the input to the \( i \)th instance
• Again, the feature mapping is not needed either to learn or to make predictions!
• This approach is useful if the feature space is very large.
Logistic regression

• The output of a logistic regression predictor is:

\[ h_w(x) = \frac{1}{1 + e^{w^T \phi(x) + w_0}} \]

• Again, we can define the weights in terms of support vectors: \( w = \sum_{i=1}^{m} \alpha_i \phi(x_i) \)

• The prediction can now be computed as:

\[ h(x) = \frac{1}{1 + e^{\sum_{i=1}^{m} \alpha_i K(x_i, x) + w_0}} \]

• \( \alpha_i \) are the new parameters (one per instance) and can be derived using gradient descent (see homework)
Kernels in Bayesian regression

- The kernel view can be applied to Bayesian regression too
- Recall that in the Bayesian view, we have a prior over the parameters, $w$
- The data induces a posterior distribution
- At any point, we can sample a parameter vector (i.e., a function) from the distribution
- Advantage: we get information about the variability of the prediction, in addition to the mean value
Example: Linear regression with features and prior

• Suppose that the weight vector $w$ has a normal prior of mean zero:

$$P(w) = \mathcal{N}(w|0, \alpha^{-1}I)$$

where $\alpha$ is the precision (inverse variance) of the distribution

• What is the probability distribution of the vector of predictions $h = \Phi w$?

• Because $h$ is a linear combination of normally distributed variables, it is also normal, so it is enough to compute the mean and covariance:

$$E(h) = E(\Phi w) = \Phi E(w) = 0$$

$$E(hh^T) = E(\Phi ww^T \Phi^T) = \Phi E(ww^T) \Phi^T = \frac{1}{\alpha} \Phi \Phi^T = K$$

where $K$ is a kernel matrix

• This is an example of a Gaussian process
Gaussian processes

• In general, a Gaussian process is a *probability distribution over functions* $h$ such that the set of values $h(x_i)$ evaluated at any arbitrary set of points $x_i$ have a jointly Gaussian distribution
• The key property of the Gaussian process is that the mean and covariance are sufficient to specify the distribution
• Gaussian processes are increasingly used in regression as well as other parts of machine learning
Gaussian process for regression

\[ f \sim \mathcal{GP} \]
\[ f \sim \mathcal{N}(0, K), \quad K_{ij} = k(x_i, x_j) \]

where \( f_i = f(x_i) \)

Noisy observations:
\[ y_i \mid f_i \sim \mathcal{N}(f_i, \sigma_n^2) \]
Gaussian Process posterior

• The prior over observations and targets is Gaussian
• Hence, the posterior for any output point will also be Gaussian
• The posterior over functions is a Gaussian Process.
• Let $\beta$ be the precision of the target noise
• To find the conditional distribution of the output $h(x)$ given the data, we partition the kernel matrix of the point and the data as:

$$\begin{pmatrix}
K & k(x) \\
k(x)^T & K(x, x) + \frac{1}{\beta}
\end{pmatrix}$$

• The mean and variance of the predictions are, respectively

$$k(x)^T K^{-1} y \text{ and } K(x, x) + \frac{1}{\beta} - k(x)^T K^{-1} k(x)$$