Lecture 10: Non-linear support vector machines. Kernels. Gaussian Processes

- 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 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.

Recall: Soft margin SVMs

- Optimization problem:

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

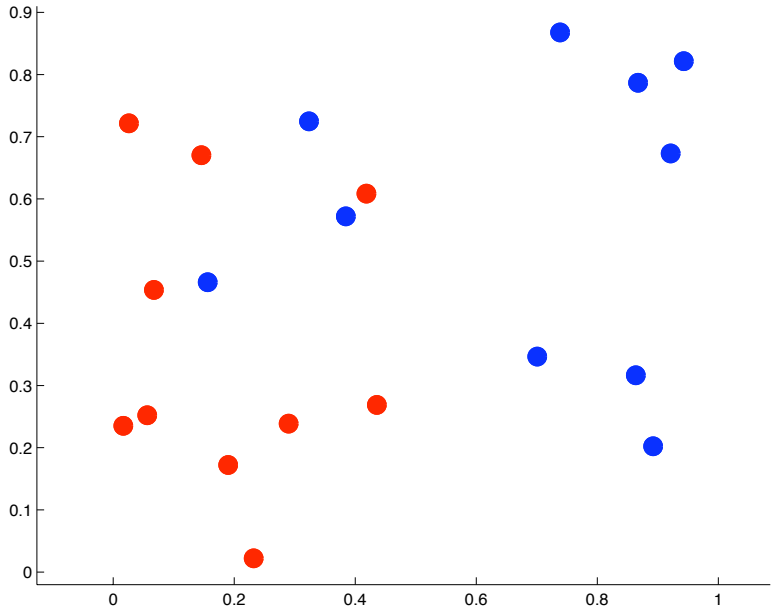
  where the first term is the margin, and the second term penalizes constraint violations

- $C > 0$ 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:

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

Non-linearly separable data



- 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, $\mathbf{x}_i$ is replaced by $\phi(\mathbf{x}_i)$, where ϕ is called a *feature mapping*

Margin optimization in feature space

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

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

- 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 space 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 of an input $\mathbf{x}$ is given by:

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

$\Rightarrow$ Note that to solve the SVM optimization problem in dual form and to make a prediction, we only ever need to compute *dot-products of feature 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 $K : \mathbb{R}^n \times \mathbb{R}^n \mapsto \mathbb{R}$ which corresponds to a dot product for some feature mapping ϕ:

$$K(\mathbf{x}_1, \mathbf{x}_2) = \phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) \text{ for some } \phi.$$

- Conversely, by choosing feature mapping ϕ, we implicitly choose a kernel function
- Recall that $\phi(\mathbf{x}_1) \cdot \phi(\mathbf{x}_2) = \cos \angle(\mathbf{x}_1, \mathbf{x}_2)$ 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 $K(\mathbf{x}, \mathbf{z}) = (\mathbf{x} \cdot \mathbf{z})^2$.
- Is this a kernel?

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

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

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

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

Polynomial kernels

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

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

- 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 ϕ. We just have to compute the similarity K.
- That is, we can solve the dual for the α_i:

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

- The class of a new input $\mathbf{x}$ is computed as:

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

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

Some other (fairly generic) kernel functions

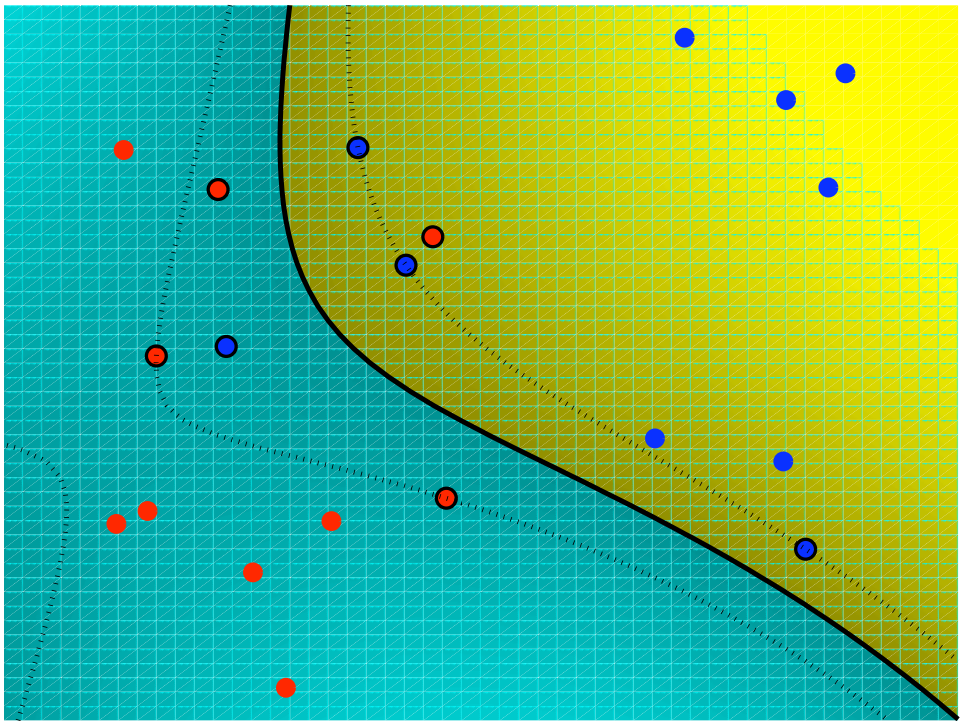
- $K(\mathbf{x}, \mathbf{z}) = (1 + \mathbf{x} \cdot \mathbf{z})^d$ – feature expansion has all monomial terms of $\leq d$ total power.
- Radial basis/Gaussian kernel:

$$K(\mathbf{x}, \mathbf{z}) = \exp(-\|\mathbf{x} - \mathbf{z}\|^2 / 2\sigma^2)$$

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

$$K(\mathbf{x}, \mathbf{z}) = \tanh(c_1 \mathbf{x} \cdot \mathbf{z} + c_2)$$

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 $\rightarrow$ 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.

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 wronlgy 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 $K(\mathbf{x}, \mathbf{z})$ be written as a dot product $\phi(\mathbf{x}) \cdot \phi(\mathbf{z})$ for some feature mapping ϕ?
- We want a general recipe, which does not require explicitly defining ϕ every time

Kernel matrix

- Suppose we have an arbitrary set of input vectors $\mathbf{x}_1, \mathbf{x}_2, \ldots \mathbf{x}_m$
- The *kernel matrix (or Gram matrix)* K corresponding to kernel function K is an $m \times m$ matrix such that $K_{ij} = K(\mathbf{x}_i, \mathbf{x}_j)$ (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 $K : \mathbb{R}^n \times \mathbb{R}^n \rightarrow \mathbb{R}$, *$K$ 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 is 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 $\mathbf{w}$ which minimizes the (regularized) error function:

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

- The solution takes the form:

$$\mathbf{w} = -\frac{1}{\lambda}\sum_{i=1}^{m}(\mathbf{w}^T\phi(\mathbf{x}_i) - y_i)\phi(\mathbf{x}_i) = \sum_{i=1}^{m} a_i\phi(\mathbf{x}_i) = \mathbf{\Phi}^T\mathbf{a}$$

  where $\mathbf{a}$ is a vector of size m (number of instances) with $a_i = -\frac{1}{\lambda}(\mathbf{w}^T\phi(\mathbf{x}_i) - y_i)$

- Main idea: *use $\mathbf{a}$ instead of $\mathbf{w}$ 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 $J(\mathbf{w})$ we have $J(\mathbf{a}$:

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

- Note that $\mathbf{\Phi}\mathbf{\Phi}^T = \mathbf{K}$, the kernel matrix!
- Hence, we can re-write this as:

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

- By setting the gradient to 0 we get:

$$\mathbf{a} = (\mathbf{K} + \lambda\mathbf{I}_m)^{-1}\mathbf{y}$$

Making predictions with dual-view regression

- For a new input $\mathbf{x}$, the prediction is:

$$h(\mathbf{x}) = \mathbf{w}^T \phi(\mathbf{x}) = \mathbf{a}^T \mathbf{\Phi} \phi(\mathbf{x}) = \mathbf{k}(\mathbf{x})^T (\mathbf{K} + \lambda \mathbf{I}_m)^{-1} \mathbf{y}$$

  where $\mathbf{k}(\mathbf{x})$ is an m-dimensional vector, with the ith element equal to $K(\mathbf{x}, \mathbf{x}_i)$

- That is, the ith element has the similarity of the input to the ith 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:

$$h_{\mathbf{w}}(\mathbf{x}) = \frac{1}{1 + e^{\mathbf{w}^T \phi(\mathbf{x}) + w_0}}$$

- Again, we can define the weights in terms of support vectors: $\mathbf{w} = \sum_{i=1}^{m} \alpha_i \phi(\mathbf{x}_i)$
- The prediction can now be computed as:

$$h(\mathbf{x}) = \frac{1}{1 + e^{\sum_{i=1}^{m} \alpha_i K(\mathbf{x}_i, \mathbf{x}) + w_0}}$$

- α_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, $\mathbf{w}$
- 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 $\mathbf{w}$ has a normal prior of mean zero:

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

  where α is the precision (inverse variance) of the distribution
- What is the probability distribution of the vector of predictions $\mathbf{h} = \mathbf{\Phi w}$?
- Because $\mathbf{h}$ is a linear combination of normally distributed variables, it is also normal, so it is enough to compute the mean and covariance:

$$E(\mathbf{h}) = E(\mathbf{\Phi w}) = \mathbf{\Phi} E(\mathbf{w}) = \mathbf{0}$$

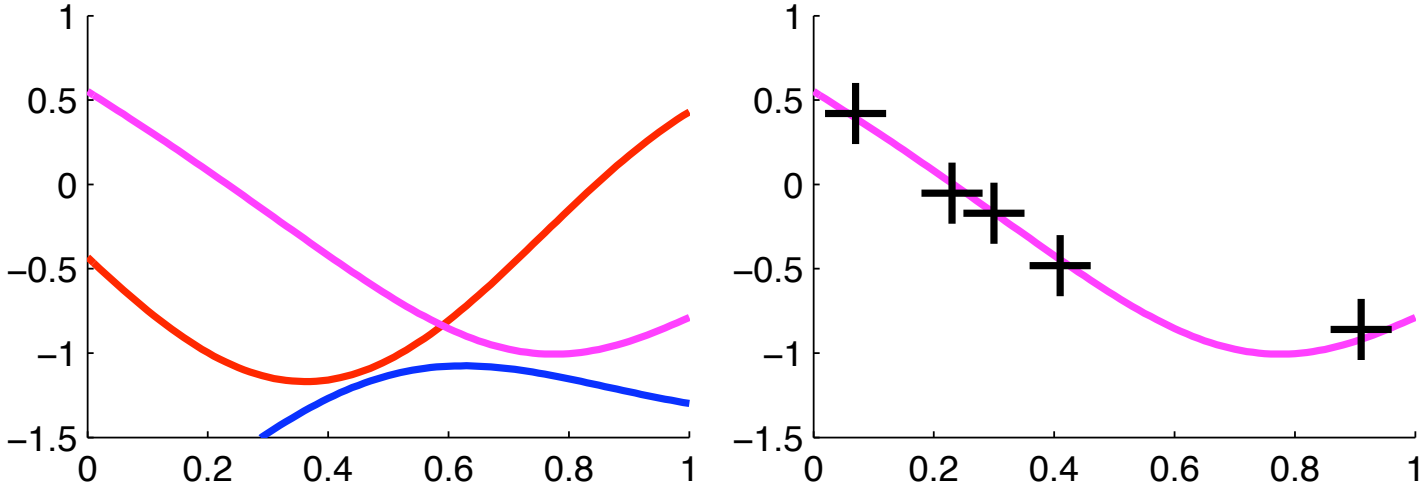
$$E(\mathbf{h}\mathbf{h}^T) = E(\mathbf{\Phi w w}^T \mathbf{\Phi}^T) = \mathbf{\Phi} E(\mathbf{w}\mathbf{w}^T)\mathbf{\Phi}^T = \frac{1}{\alpha}\mathbf{\Phi}\Phi^T = \mathbf{K}$$

  where $\mathbf{K}$ 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 $h(\mathbf{x}_i)$ evaluated at any arbitrary set of points $\mathbf{x}_i$ 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



$$f \sim \mathcal{GP}$$

$$\mathbf{f} \sim \mathcal{N}(0, K), \quad K_{ij} = k(x_i, x_j)$$

where $f_i = f(x_i)$

Noisy observations:

$$y_i | f_i \sim \mathcal{N}(f_i, \sigma_n^2)$$

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 β be the precision of the target noise
- To find the conditional distribution of the output $h(\mathbf{x})$ given the data, we partition the kernel matrix of the point and the data as:

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

- The mean and variance of the predictions are, respectively

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