

[Hint: For part (e), the answer is that the K there *is* indeed a kernel. You still have to prove it, though. (This one may be harder than the rest.) This result may also be useful for another part of the problem.]

2. [10 points] Kernelizing the Perceptron

Let there be a binary classification problem with $y \in \{-1, 1\}$. The perceptron uses hypotheses of the form $h_\theta(x) = g(\theta^T x)$, where $g(z) = \text{sign}(z) = 1$ if $z \geq 0$, -1 otherwise. In this problem we will consider a stochastic gradient descent-like implementation of the perceptron algorithm where each update to the parameters θ is made using only one training example. However, unlike stochastic gradient descent, the perceptron algorithm will only make one pass through the entire training set. The update rule for this version of the perceptron algorithm is given by

$$\theta^{(i+1)} := \begin{cases} \theta^{(i)} + \alpha y^{(i+1)} x^{(i+1)} & \text{if } h_{\theta^{(i)}}(x^{(i+1)}) y^{(i+1)} < 0 \\ \theta^{(i)} & \text{otherwise,} \end{cases}$$

where $\theta^{(i)}$ is the value of the parameters after the algorithm has seen the first i training examples. Prior to seeing any training examples, $\theta^{(0)}$ is initialized to $\vec{0}$.

Let K be a Mercer kernel corresponding to some very high-dimensional feature mapping ϕ . Suppose ϕ is so high-dimensional (say, ∞ -dimensional) that it's infeasible to ever represent $\phi(x)$ explicitly. Describe how you would apply the “kernel trick” to the perceptron to make it work in the high-dimensional feature space ϕ , but without ever explicitly computing $\phi(x)$. [Note: You don't have to worry about the intercept term. If you like, think of ϕ as having the property that $\phi_0(x) = 1$ so that this is taken care of.] Your description should specify

- How you will (implicitly) represent the high-dimensional parameter vector $\theta^{(i)}$, including how the initial value $\theta^{(0)} = \vec{0}$ is represented (note that $\theta^{(i)}$ is now a vector whose dimension is the same as the feature vectors $\phi(x)$);
- How you will efficiently make a prediction on a new input $x^{(i+1)}$. I.e., how you will compute $h_{\theta^{(i)}}(x^{(i+1)}) = g(\theta^{(i)T} \phi(x^{(i+1)}))$, using your representation of $\theta^{(i)}$; and
- How you will modify the update rule given above to perform an update to θ on a new training example $(x^{(i+1)}, y^{(i+1)})$; i.e., using the update rule corresponding to the feature mapping ϕ :

$$\theta^{(i+1)} := \theta^{(i)} + \alpha \mathbf{1}\{\theta^{(i)T} \phi(x^{(i+1)}) y^{(i+1)} < 0\} y^{(i+1)} \phi(x^{(i+1)}).$$

[Hint: our discussion of the representer theorem may be useful.]