Homework Five, for Thu 2/18 CSE 250B

Your homework must be typeset, and the PDF file must be uploaded to Gradescope by midnight on the due date.

1. *Voting Perceptrons.* The perceptron algorithm, as presented in class, will not converge when given data that is not linearly separable.

   One work-around is to simply halt the algorithm after a fixed number (say, T) of passes through the data, and to output the vector w at that time. But this has a drawback: it might be that an update occurred right before termination, and caused the final w to point in a poor direction.

   A better idea is to combine the various w's obtained during the training process. In particular, a vector w that survived for quite a long time is likely to be good. This is the motivation for the *voted perceptron* algorithm. Given data $(x^{(1)}, y^{(1)}), \ldots, (x^{(n)}, y^{(n)}) \in \mathbb{R}^p \times \{-1, 1\}$, it operates as follows:

   - $\ell = 1$, $c_\ell = 0$
   - $w_1 = 0$
   - Repeat T times:
     - Randomly permute the data points
     - For $i = 1$ to n:
       - If $(x^{(i)}, y^{(i)})$ is misclassified by w_ℓ:
         - $w_{\ell+1} = w_\ell + y^{(i)} x^{(i)}$
         - $c_{\ell+1} = 1$, $\ell = \ell + 1$
       - Else:
         - $c_\ell = c_\ell + 1$

   At the end of this process, we have a collection of linear separators $w_1, w_2, \ldots, w_\ell$ as well as their "survival times" $c_1, \ldots, c_\ell$. To classify a new point x, we take the weighted majority vote:

   $$\text{sign}\left(\sum_{j=1}^{\ell} c_j \,\text{sign}(w_j \cdot x)\right).$$

   An alternative rule is the simpler *averaged perceptron*: predict $\text{sign}(w \cdot x)$, for

   $$w = \sum_{j=1}^{\ell} c_j w_j.$$

   (a) Implement the voted perceptron algorithm and try it out on the small 2-d data set `data1.txt`. (Remember to add a constant-valued feature to the data.) Try $T = 10$ or thereabouts. Show the final decision boundary. Is it linear?

   (b) After a few passes through the data, the number of classifiers ℓ might grow inconveniently large. Can you devise a good way to downsample them so that at most L different values of w are ever stored? Give pseudocode, and redo the previous problem (part (b)) using your technique. Try a higher value of T to assess how well your downsampling is working.

   (c) Give pseudocode for the averaged perceptron that avoids keeping track of all w's seen (it should maintain at most two w's). Show the decision boundary it achieves on the small data set, with $T = 10$.

2. *Kernelized perceptron.* Implement the kernel perceptron algorithm for the quadratic and RBF kernels. Show the boundaries obtained for the small data sets `data1.txt` and `data2.txt`. For the RBF kernel, show boundaries for a few settings of the scale parameter σ.