250B Problem Set 5

William Fedus

March 5, 2015

1 Voting Perceptrons

Here we employ the perceptron model for the classification of points with input in $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \{-1, 1\}$.. Additionally, we append a constant valued feature to the input space $\mathcal{X}$ to embed it in $\mathbb{R}^3$ so that the solutions for our hyperplane w may be found as intersecting the origin.

1.1 Voting Perceptrons

One issue with the classic perceptron model is that for data that is not linearly separable, the algorithm will never converge as some data will consistently be misclassified. In order to still use this powerful algorithm and not instate an arbitrary stopping condition, we employ a voting perceptron algorithm where we store a collection of l linear separating hyperplanes $\{w_1, w_2, \ldots, w_l\}$ which persist for $\{c_1, c_2, \ldots, c_l\}$ iterations of the algorithm. Those separators w_j that have a high persistence c_j will have a high weight or 'vote' in the final classifier. In particular, a new point x will be classified according to the weighted majority vote in Equation 1

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

The resulting decision boundary we find from the voting perceptron algorithm is displayed in Figure 1 after using $T = 10$ iterations.

With this algorithm, we see that our resulting decision boundary is non-linear and it achieves a classification accuracy of 0.987 on this particular iteration. Note that the boundary and the classification accuracy is subject to fluctations based on the randomized aspect of the algorithm.

1.2 Voting Perceptrons with Limited Model Storage

One issue with the voting perceptron algorithm is that the number of stored hyperplanes w can become large as we run more iterations of the algorithm. To alleviate this issue, we select a value L to be the maximum number of models

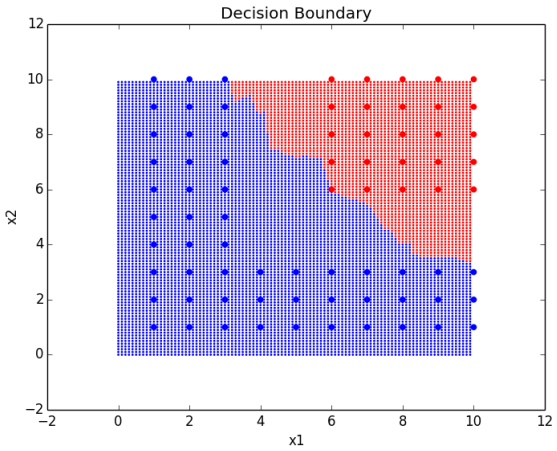

Figure 1: Resulting decision boundary on dataset1.txt where we using $T = 10$ iterations of the voting perceptron model.

w_j we wish to store. We will save the first L models as before, but now if we exceed the number L, we simply write over the oldest saved model. In this way, we simply keep the L most recent models and predict off of these.

Using this simple selection procedure, only allowing $L = 20$ models to be stored, we find a classification accuracy of 0.974 after $T = 100$ iterations of the voting perceptron with limited memory storage. The resulting decision boundary is displayed in Figure 2.

1.3 Averaged Perceptrons

Finally, as an additional iteration on these basic procedures, we use the averaged perceptron, which instead of storing all the models, it only keeps the current model w with the update $w = w + y^{(i)}x^{(i)}$ if i is misclassified and also a running cumulative sum as given by Equation 2.

$$w = \sum_{j=1}^{l} c_j w_j \tag{2}$$

Then the classification of a point x becomes the much simpler rule of $\text{sign}(w \cdot x)$ where the averaged w is given by Equation 2

Using this approach, we achieve a 0.908 classification accuracy and the resulting decision boundary on dataset1 is shown in Figure 3.

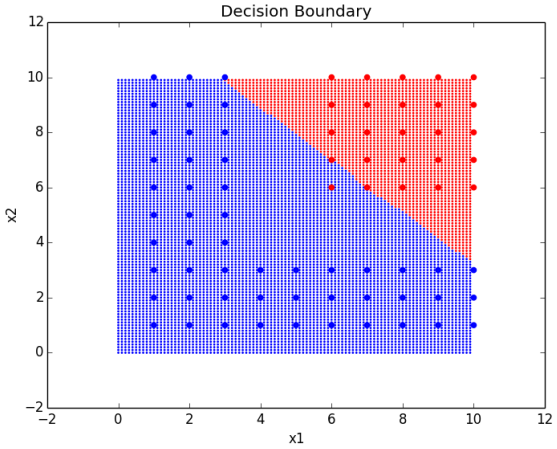

Figure 2: Resulting decision boundary on dataset1.txt where we using $T = 100$ iterations of the voting perceptron model and we restrict the total models to $L = 20$.

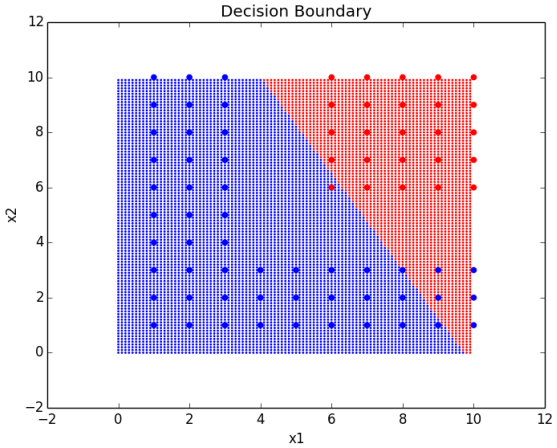

Figure 3: Decision boundary using the averaged perceptron.

2 Kernalized Perceptrons

Finally, we implement quadratic and RBF kernel perceptron algorithms for the classification of points with input in $\mathcal{X} = \mathbb{R}^2$ and $\mathcal{Y} = \{-1, 1\}$. This approach maps our points into a higher dimensional space as seen in Equation 3

$$\{x_1, x_2, \ldots, x_n\} \mapsto \{\phi(x_1), \phi(x_2), \ldots, \phi(x_n)\} \tag{3}$$

and gives us the added benefit that we may find more sophisticated classification schemes in our original $\mathbb{R}^2$ space while using a simple generalization of the basic perceptron algorithm as seen in Algorithm 1.

Algorithm 1 Kernelized Perceptron

```
1: procedure KERNELIZED PERCEPTRON (PRIMAL)
2:     w = 0
3:     while some  y · (w · φ(x)) < 0 do
4:         w = w + y · φ(x)
```

As seen in the primal form, we require the calculation of $w \cdot \phi(x)$, which could be potentially costly given our enhanced feature space may be prohibitively large, or potentially infinite. However, by representing this problem in the Dual form, we may express w as a linear combination of $\phi(x_i)$ and we can efficiently compute the dot product $\phi(x_i) \cdot \phi(x_j)$ without ever writing out the full vectors in the enhanced feature space. In fact, we use this dot product as the definition of our kernel function as seen in Equation 4

$$k(x_i, x_j) \equiv \phi(x_i) \cdot \phi(x_j) \tag{4}$$

and the quadratic kernel is given in Equation 5

$$k(x_i, x_j) \equiv (1 + x_i \cdot x_j)^2 \tag{5}$$

and the RBF kernel is given in Equation 6

$$k(x_i, x_j) \equiv e^{||x_i - x_j||^2 / 2\sigma^2} \tag{6}$$

The results of our classification using the quadratic kernel are seen in Figures 4 and 5.

Our classification results using the RBF kernel are seen in Figures 6, 7, 8, 9, 10 and 11.

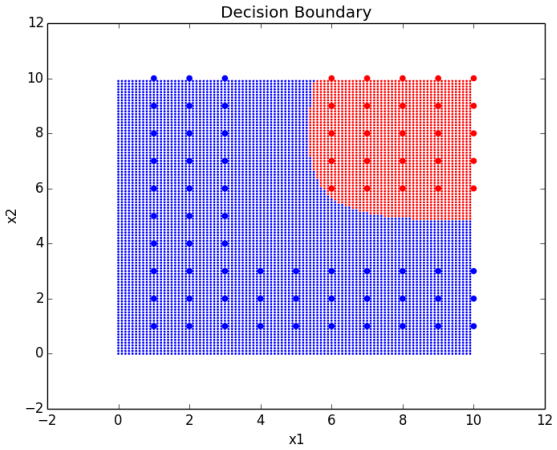

Figure 4: Decision boundary given by the quadratic kernel on dataset1.txt.

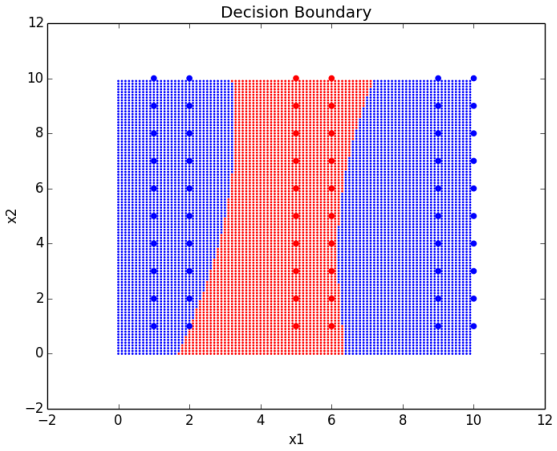

Figure 5: Decision boundary given by the quadratic kernel on dataset2.txt

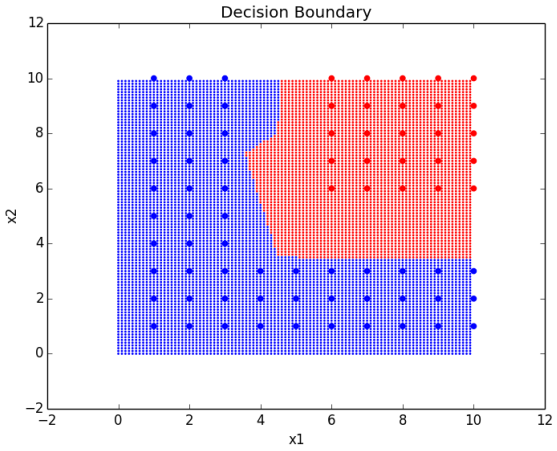

Figure 6: Decision boundary given by the RBF kernel on dataset1.txt with $\sigma = 0.1$.

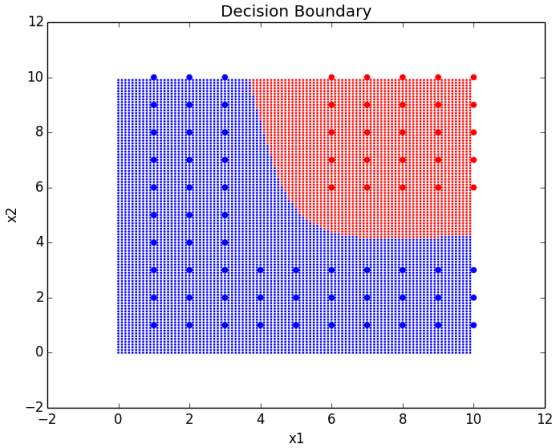

Figure 7: Decision boundary given by the RBF kernel on dataset1.txt with $\sigma = 1$.

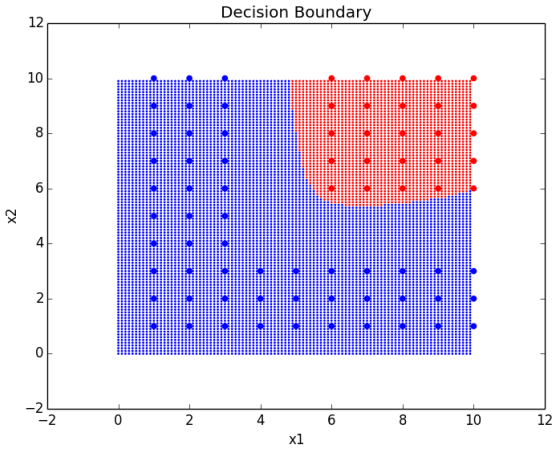

Figure 8: Decision boundary given by the RBF kernel on dataset1.txt with $\sigma = 10$.

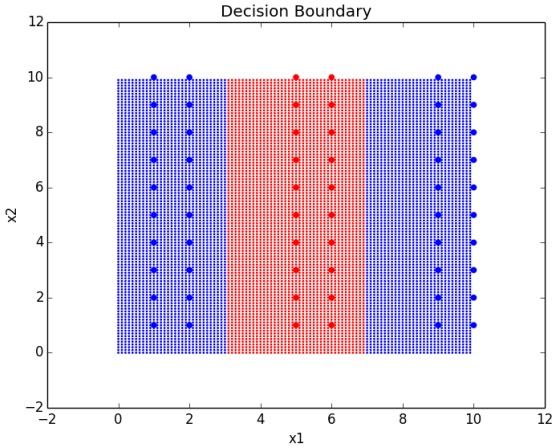

Figure 9: Decision boundary given by the RBF kernel on dataset2.txt with $\sigma = 0.1$.

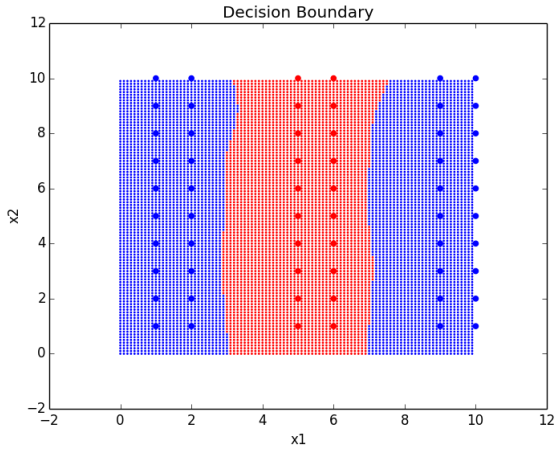

Figure 10: Decision boundary given by the RBF kernel on dataset2.txt with $\sigma = 1$.

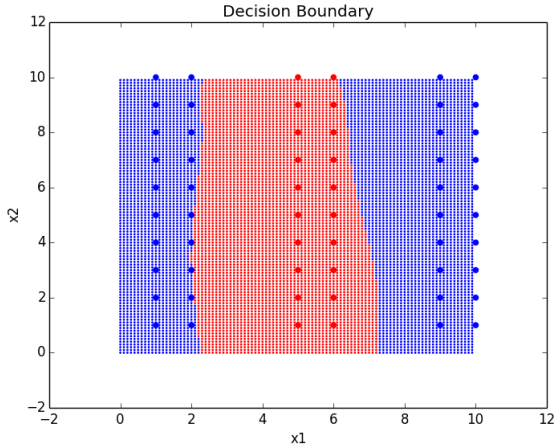

Figure 11: Decision boundary given by the RBF kernel on dataset2.txt with $\sigma = 10$.