**k-Nearest Neighbors (knn)**

1. Implement the knn classification algorithm such that it can work with different distance functions and with different values of ‘k’. In the first part of this question apply the knn to the spambase dataset with different values of ‘k’ i.e., 1,3,5, and 7, using Euclidean distance.
2. For the knn classifier the distance metric used has to be selected based on the dataset. For the raw USPS dataset compare the final classification accuracy using both the Euclidean and Cosine distance, for different values of k i.e., 1, 3, 5, and 7. Discuss the differences in the accuracies and the underlying causes for these differences.
3. Adapt the knn classifier to work with kernels. Apply the kernelized version with the same values of k to both the Iris and USPS dataset using the RBF and polynomial degree 2 and 3 kernel. Contrast the performance of the kernelized version with that of the non-kernelized version using Euclidean distance.

**Kernels for learning non-linear decision boundaries**

1. Implement the dual version of the perceptron, keeping in mind that we need to work with kernels. Apply both the primal and the dual perceptron to the Iris dataset that has three classes. Contrast the performance and the final weight vector calculated using both the primal and the dual version to distinguish between ‘setosa’ and ‘versicolor’ and also between ‘setosa’ and ‘virginica’. Do you see any differences or would you consider that both the algorithms are identical with respect to the final classifier?
2. Use the RBF and polynomial degree-2 and degree-3 polynomial kernels to develop and evaluate a one-vs-rest classification scheme for the Iris dataset. Compare the performance of the kernelized dual perceptron with that of the non-kernelized version, which classes benefit from the use of kernels and why?
3. In this part we are going to work with three two-dimensional binary classification datasets that you can easily visualize. Use the dual perceptron with the linear, polynomial (degree 2 and 3), and RBF kernel to learn four different weight vectors per dataset. Visualize the decision boundaries for each individual classifier. Discuss which kernel(s) is suitable for each dataset, and in cases where more than one kernel is applicable which kernel would you choose and why?
4. Visualize the Spiral toy dataset, and select two kernels that you think are appropriate for this dataset (linear, polynomial (any degree), or RBF). Discuss why you chose the kernels, and which one from your selected duo would outperform the other. Support you claims by estimating the performance of the dual perceptron using your selections based on a ten-fold cross-validation scheme.

**Optional:**

If a classification algorithm can be posed as an optimization problem of the form:

|  |  |
| --- | --- |
| $$min\_{w} Loss\left(y,f(w,x)\right)+λ Penalty(w)$$ | (1) |

where $w$ are model parameters, $f(w,x)$ is classifier output, and $λ$ is a regularization parameter. Then under some “weak” conditions on the loss and penalty function, we can represent the solution of the minimization problem as1:

|  |  |
| --- | --- |
| $$\hat{w}=\sum\_{i=1}^{N}α\_{i}y\_{i}ϕ(x\_{i})$$ | (2) |

i.e., a linear combination of the input data instances. This implies that such algorithms can be easily kernelized.

1. Consider a binary classification case where the labels take on the values -1 and 1. How can we use regularized linear regression for predicting the output labels in this case?
2. What is the loss function that linear regression minimizes in this case?
3. If we use regularized linear regression, how can we write the final weight vector as Equation-2. What are the alphas in this case, and is linear regression kernelizable?
4. Compare your developed loss function from part (b) with the Hinge and Log loss. Can we consider linear regression as a max-margin learning algorithm? (Hint: plot the different loss functions, and see what happens when we try to maximize the margin for a single input data instance)

**References**

[1] Schölkopf, Bernhard; Herbrich, Ralf; Smola, Alex J. (2001). "A Generalized Representer Theorem". Computational Learning Theory. Lecture Notes in Computer Science 2111: 416–426.