# CSCI 5521: Introduction to Machine Learning (Spring 2021)[1]

## Extra Programming Resources (Optional)

## Due date: Apr 28, 2021 11:59pm, same as HW4

(**3 points, extra credits,** $3 * 14\%$ **toward total score**) In this programming exercise you will implement a Kernel Perceptron using Radial Basis Function (RBF) kernel. Similar to the perceptron algorithm which classifies data into binary classes with $y = sign(w^T x)$, the Kernel Perceptron makes the decision with the function $y = sign(\sum_t \alpha^t r^t K(x^t, x))$, where $x^t$ and $r^t$ are training samples and their labels, $x$ is the sample to be classified, $\alpha^t$ is a counter that records the time $x^t$ is misclassified during training, and $K(x^t, x) = exp\left[-\frac{||x^t - x||^2}{2\sigma}\right]$ is the RBF kernel. During training, the counter $\alpha^t$ is initialized as 0 and will be updated as $\alpha^t = \alpha^t + 1$ when $y^t \neq r^t$ for the current iteration.

1. Test your implementation of the Kernel Perceptron algorithm using the simulation data generated by `generate_data` function (You do not need to modify the function). **Report the accuracy and plot the decision boundary.**
   The decision boundary is a path which separates a region that your model classifies as $+1$ from the region it classifies as $-1$.
   **Hint:** One way to plot decision boundaries (we accept other ways) is using the `meshgrid` and `contourf` functions from matplotlib.
   **Hint 2:** When using `meshgrid`, make sure the number of points in your grid is not extremely large, otherwise it will be very slow to make prediction for each position of the grid. On the other side, using too few points will make the boundary look jagged.
   **Hint 3:** To select a good $\sigma$ value for the RBF kernel, look at the Figure 13.5 in the textbook, where $\sigma = s^2$. Experiment with different $\sigma$ values (not limited to those in the textbook) and choose the one that provide the best accuracy. For simplicity, we do not have a separate validation split, and the model selection is conducted on the test split. **Note that you need to specify your choice of $\sigma$ at the beginning of** `MyKernelPerceptron.py`

2. What happens as you increase $\sigma$? Report the results for different $\sigma$ values and briefly explain.

3. Once you have tested that your Kernel Perceptron works, train and evaluate your implementation using two subsets of the optdigits dataset. The first, containing only the digits 4 and 9, in the files `digits49_train.txt` and `digits49_test.txt`. And the second, containing only the digits 7 and 9, in the files `digits79_train.txt` and

[1]Instructor: Catherine Qi Zhao (qzhao@umn.edu). TA: Shi Chen (chen4595@umn.edu), Yifeng Zhang (zhan6987@umn.edu), Dyah Adila (adila001@umn.edu)

`digits79_test.txt`.
**Report the test accuracy on both datasets.** Note that you may need to specify different $\sigma$ for different datasets.

We have provided the skeleton code `MyKernelPerceptron.py` and `visualization.py` for implementing the algorithm. `MyKernelPerceptron.py` is written in a *scikit-learn* convention, where you have a *fit* function for model training and a *predict* function for generating predictions on given samples. To verify your implementation, call the main function `extra_credit.py`.

## Submission

- **Things to submit:**

  1. `MyKernelPerceptron.py`: a Python source file containing your implementation of the Kernel Perceptron algorithm (`class KernelPerceptron`). Use the skeleton file `MyKernelPerceptron.py` found with the data on the class web site, and fill in the missing parts. The `fit` function should take both training and validation samples as inputs, update the model parameters based on training samples. The *predict* function should take features as inputs and return the predicted class labels.
  2. `visualization.py`: a Python source file containing your code for visualizing the decision boundary of the model. Given the trained model, different samples and their class labels as inputs, the *plot_boundary* function should plot the decision boundary and the given samples. Samples from different classes should be labeled with different colors.

- **Submit**: All material must be submitted electronically via Gradescope (*i.e.,* entry Extra_programming). We will grade the assignment with vanilla Python, and code submitted as iPython notebooks will not be graded.