AIML - CS 335

Lab 8: Kernels Kernels Kernels

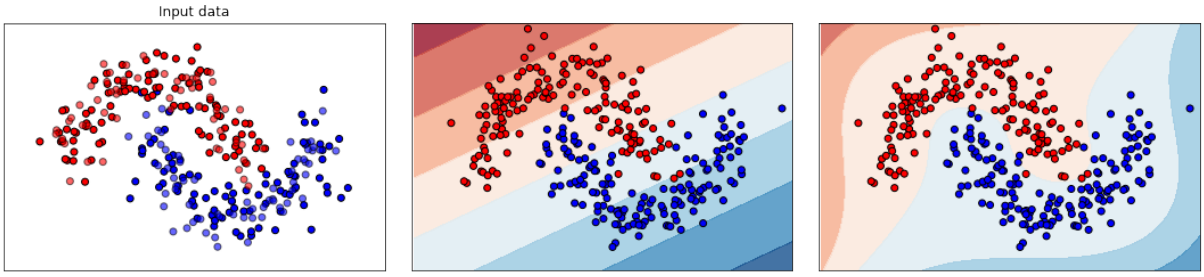
29 September 2022

Lecturer: Abir De

Important: Please read the instructions mentioned in the questions carefully. We have provided boilerplate code for each question. Please ensure that you make changes in the areas marked with TODO. Please read the comments in the code carefully for detailed information regarding input and output format.

1 Trick or Treat

The two moons dataset is a classical set of 2D points that serves as a benchmark for a good classifier. Your task for the first question is to expand the feature space of the input data using polynomial transformations to achieve an accurate decision boundary. This can be using binomial expansion of $(x_1 + x_2)^n$, where x_1 and x_2 are the original features and n is a tunable degree. Alternatively, polynomial features for any value of n can be made by designing an appropriate polynomial kernel. Begin by designing additional features in the function `poly_features` and then design a suitable kernel by completing the function `my_kernel`.



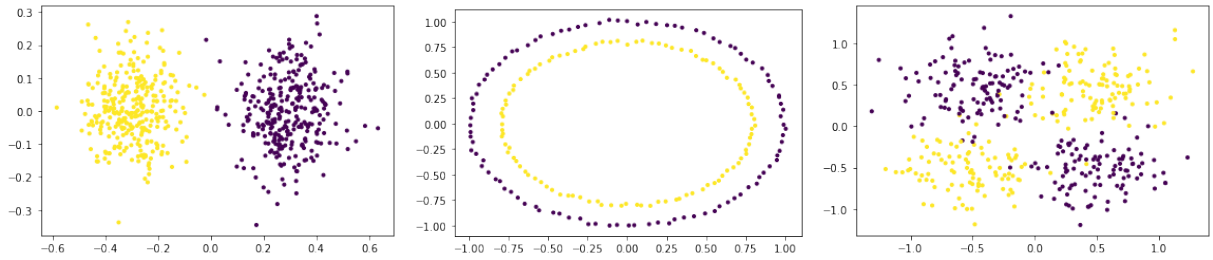
(a) Two moons dataset (b) Linear Kernel (c) Polynomial kernel

2 Not your typical Kernel

This section aims to help you learn how to apply kernel functions to non linear data so that it can be classified using linear classifiers in an efficient way. More accurately, kernel functions provide a way to manipulate data as though it were projected into a higher dimensional space, by operating on it in its original space. In real life applications data is very segregated and full of complexities like bias, variance, correlation, etc.

So to understand how to best use this data for a task and how it can be used for classification

even if the data seems non linearly separable at first glance, we need to understand which kernel function is best suited for a task. For example: In case of 2-D features which are linearly separable, the hyperplane separating them is a flat affine 1D subspace (line). So looking at the below 2 dimensional data find the kernel function which is more likely suited for a support vector classifier.

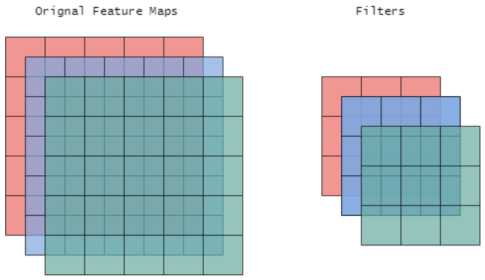


(a) Sample Data for Kernel 1 (b) Sample Data for Kernel 2 (c) Sample Data for Kernel 3

Try not to overfit to the data and find a kernel function which will generalize well on new data. Fill up the functions **Q2_kernel_1**, **Q2_kernel_2**, **Q2_kernel_3** such that when they are passed with input feature a shape=(n,d), b shape=(n,d) they return the relation between every point in the higher dimensional space. This can be thought of as getting the relation between input features in higher dimension without converting them into higher dimensional subspace.

3 It's Convoluted

This section first briefly touches on the idea of convolution for images, used as a backbone in convolution neural networks.



(a) RGB multichannel input with corresponding kernel filter

So the image dimensions can be written as follows: Height (in pixels) * Width (in pixels) * Depth (no. of color channels)

When Depth $>$ 1, then its called a multi input channel.

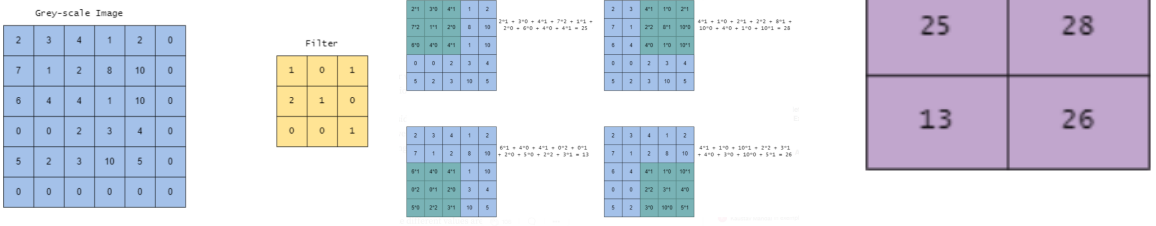
3.1 Kernel Convolution

This is the process where we simply pass a kernel/filter (matrix of numbers) over the image pixel matrix and transform it based on the kernel values. We discuss the following notations here.

Padding

As we move our filter over our image, it impacts the pixels located on the boundaries much less as compared to the image center. Hence this can be treated as a lossy compression. We thus use *padding*, usually consisting of zeroes. If we try this with our grey-scale image from before, then it would look like so as in Fig. 4a

Stride It controls how much we move our filter in each step. In this example we take a stride of value = 2, resulting in a final feature map of dimensions 2x2. Fig. 4b, Fig 4c



(a) Padding Instance (b) Convolution using stride=2 (c) Feature Map obtained

By considering the below notations,

- n: image size
- f: filter size
- n_c — number of color channels in the image
- p: padding size
- s: stride size
- n_f: number of filters

we can compute the new dimensions can be calculated by using the following formula:

$$n_{out} = \lfloor \frac{n_{in} + 2p - f}{s} + 1 \rfloor$$

.

If we then have multiple color channels, i.e. a depth greater than 1, then we can use the following formula to calculate the dimensions:

$$n_{out} = \lfloor \frac{n_{in} + 2p - f}{s} + 1, \frac{n_{in} + 2p - f}{s} + 1, n_c \rfloor$$

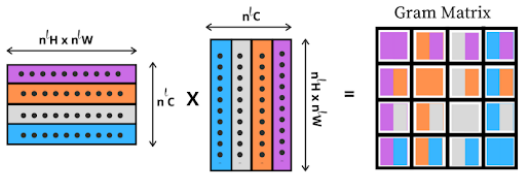
3.2 Gram Matrix Computation

Given the matrix X, the Gram Matrix is simply given by $X^T.X$.

Upon performing the above convolution, we take n_c feature vectors(flattened) from a convolutional feature map of depth n_c and compute the dot product with every one of them(including with a feature vector itself). The result is the Gram Matrix. You can read more in detail here -

TODO: Perform the convolution of a given input image pixel array by implementing a naive convolution function.

Then use the feature output obtained above to compute the gram matrix.



(a) Gram Matrix computation post convolution

4 On the Edge

This section is intended to introduce you to convolution operation by applying it to detect edges in an image. This part requires minimal coding!

A colour image is an array of dimension N x M x 3 where N is the height (number of rows), M is the width (number of columns) and 3 is related to the colors red, green, blue composing the image. A grayscale image image is an array of dimension N x M.

The Sobel operator is used in image processing and computer vision, particularly within edge detection algorithms where it creates an image emphasising edges. Technically, it is a discrete differentiation operator, computing an approximation of the gradient of the image intensity function. At each point in the image, the result of the Sobel–Feldman operator is either the corresponding gradient vector or the norm of this vector. The Sobel–Feldman operator is based on convolving

the image with a small, separable, and integer-valued filter in the horizontal and vertical directions and is therefore relatively inexpensive in terms of computations. Please refer to Sobel operator's (or filter's) Wikipedia page for more information (recommended).

TO DO: Complete the edgeDetection function in the sections marked. Running the code will save an image called imageWithEdges.png. You are expected to submit that as a part of the submission folder too.

Submission instructions

Complete the functions in **assignment.py**. Keep the file in a folder named **<ROLL_NUMBER>_L8** and compress it to a tar file named **<ROLL_NUMBER>_L8.tar.gz** using the command

tar -zcvf <ROLL_NUMBER>_L8.tar.gz <ROLL_NUMBER>_L8

Submit the tar file on Moodle.

The directory structure should be -

<ROLL_NUMBER>_L8
| - - - - assignment.py
| - - - - Q1.py
| - - - - Q2.py
| - - - - Q3.py
| - - - - Q4.py
| - - - - utils.py
| - - - - input/* *Do Not Modify the input directory*
| - - - - output/
| - - - - - - - - - imageWithEdges.png
| - - - - - - - - - q1_linear.png
| - - - - - - - - - q1_poly.png

Replace ROLL_NUMBER with your own roll number. If your Roll number has alphabets, they should be in "small" letters.