

Ex1 MC:

1. What is the difference between boosting and Random Forests?
 1. Random forests never train on bootstrapped data
 2. Boosting uses weighted averaging of predictions
 3. ?
 4. ?
2. What is the VC dimension
 1. Minimum dimension to shatter a number of points
3. What is true for backpropagation?
 1. Computes each weight independently
 2. Backpropagates the error from the output to the input
 3. A Neural net can be trained without parameters
 4. Can only be used for fully connected Neural Nets
4. ?

Ex2

- a.) Given the Dataset $D = \{1, 2, 4\}$ produced by the Density function $\theta(1 - \theta)^{x_i-1}$ with $x_i - 1 \geq 0$ derive a closed form solution for the Data Likelihood
- b.) Compute Maximum likelihood solution $\hat{\theta}$
- c.) Compute $P(x_4 = 1|\hat{\theta})$
- d.) Bayesian View: Assuming a prior of $P(\theta) = 2(1 - \theta)$ show that can be written as

$$P(\theta|D) = \frac{\theta^a (1 - \theta)^b}{\int_0^1 \theta^a (1 - \theta)^b d\theta}$$

state the values of a and b

- e.) Using your Bayesian Classifier: compute $P(x_4 = 1|\theta)$

Ex3

PCA

- a.) Derive the necessary condition for the optimization problem using the Lagrangian:

$$\max_u u^T C u$$

s.t. $\|u\|_2 = 1$

- b.) show that the solution is obtained by the Eigenvector corresponding to the largest Eigenvalue of C.

- c.) Let $\{u_1, \dots, u_d\}$ be an Eigenbasis of C. Assuming C is centered i.e. $\mathbb{E}[xx^T] = 0$

Show that $\mathbb{E}[\|x - (\hat{x})\|^2] = \sum_{i=2}^d \lambda_i$ with $\hat{x} = u_1^T x u_1^T$

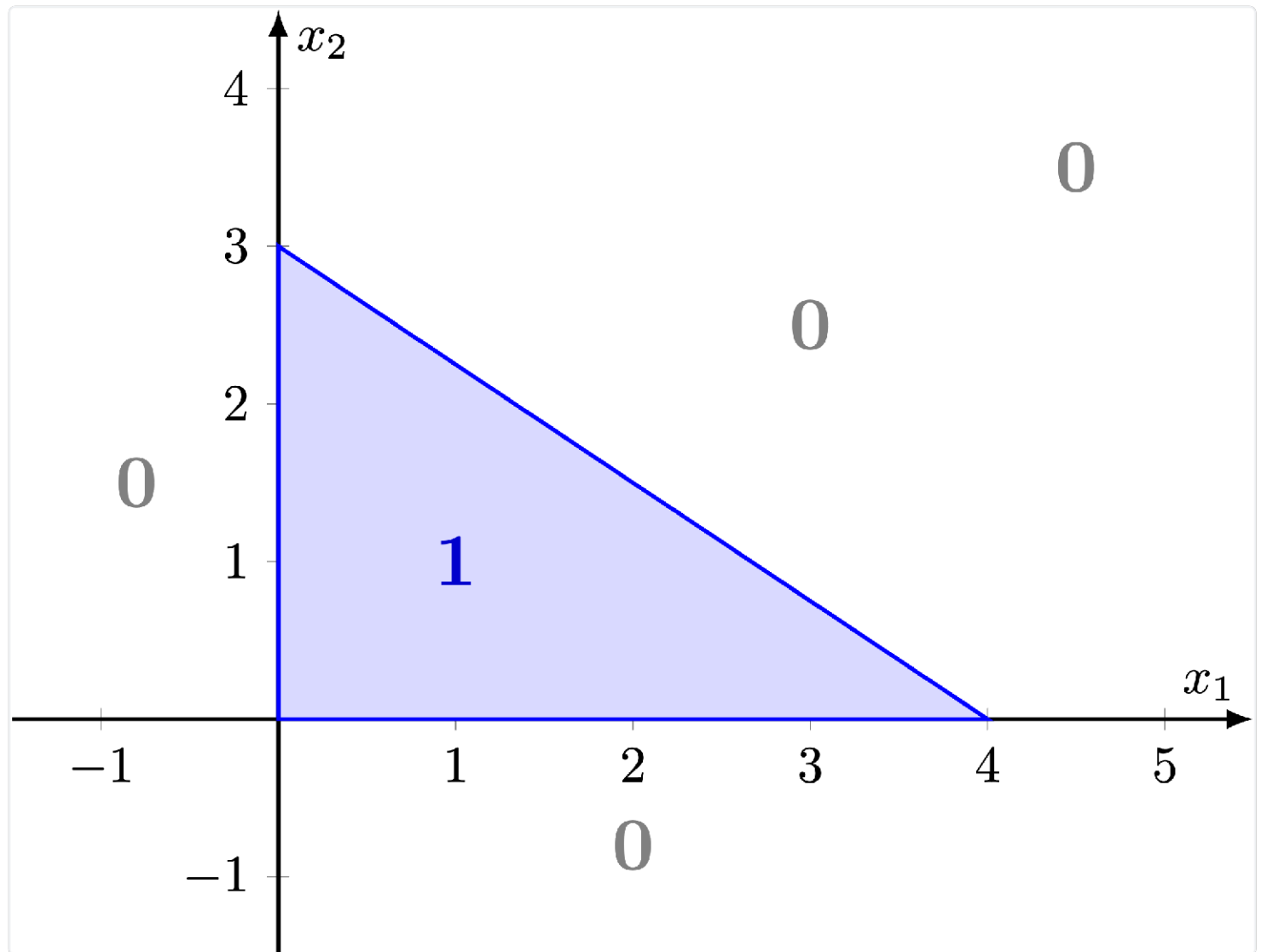
(hint) Write x in the Eigenbasis.

Ex 4.

a) Given the Neural network activation function:

$$a_j = \mathbf{1}_{\sum_i w_{ij} x_i + b_j}$$

Construct a Neural network which performs the following classification. Draw it. Annotate your drawings with the weights, biases etc.



b.) State all activations for the point $x_i = (3, 1)$ and provide the final classification

EX5

a.)

a few numpy functions given (only their names, not the actual docstring or similar)

Compute the Kernel function:

$$k(x, z) = (\mathbf{x}^T \mathbf{z} + c)^p$$

```
def kernel(X, Z=None, p):  
    # X nxd  
    # Z nxd
```

```
return K
```

If Z is None the Kernel Matrix $k(x_i, x_j)$ should be returned

b.)

Compute the alpha for KRR. Your function should be efficient (no for loops) and numerically stable

```
def compute_alpha(X, y, c):  
    # X nxd  
    # y n,  
    # c = small value larger than zero  
  
    return alpha
```