Q1 (a) We note that, the missclassification rate highly depends on the choice of image classes and the ratio of the class size. If one class is significantly smaller, then the classification result may be extremely bad. Therefore, we choose to work with two classes of similar sizes.

We pick two classes of size 121 and 144 respectively for the analysis in the first part, and 3 by 3 for the size of the square patches. The kernel between images are found by averaging the pairwise Gaussian/Vovk Polynomial kernel for two patches that are of 50% overlaps.

The images are first normalised and processed into 3×3 dense patches, where each patch is divided by the L_2 vector norm. Next, the two classes of images are split into train and test set with ratio 75 to 25. Then the nearest mean clustering algorithm is run for varying hyperparameters, $\sigma > 0$ in Gaussian kernel and $p \in \mathbb{N}$ for Vovk Polynomial kernel.

Notably, the Vovk Polynomial kernel for two vectors $x, y \in \mathbb{R}^d$ with hyperparameter p is

$$k(x,y) = \sum_{k=0}^{p-1} (x \cdot y)^k = \frac{1-(x\cdot y)^p}{1-(x \cdot y)},\ p \in \mathbb{N}$$

since the the second equality does not hold if the inner product between two vectors are of value +/-1, in the implementation, it is handled with the first formula, i.e. the evaluation is p.

The misclassification rate for the algorithm with the Gaussian kernel is as shown in Fig 1, and Vovk Polynomial kernel as shown in Fig 2. And since for $p = 1$, the evaluation for any pairs of vectors is always 1, so we only consider $p \geq 2$.

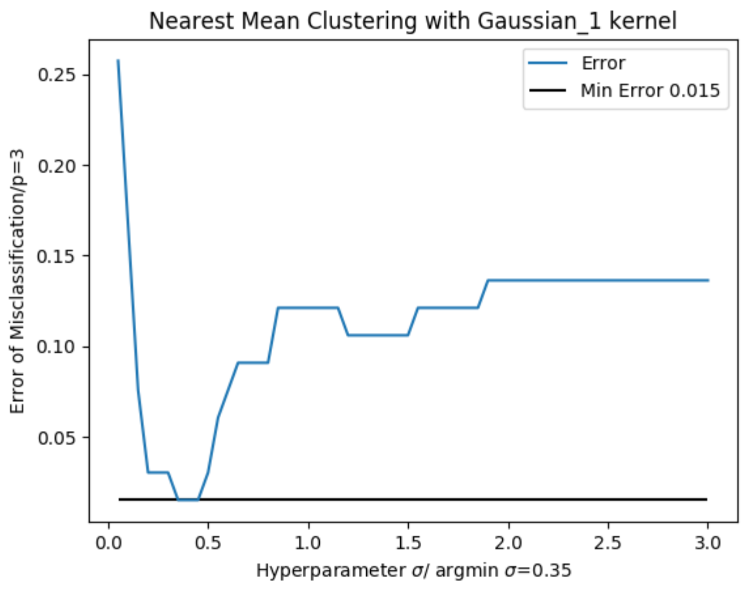

Figure 1: Missclassification rate of two classes of images with Gaussian kernel for varying hyperparameter σ.

We see that in Fig 1, the misclassification rate has a sharp drop when σ is around 0.2, and reaches minimum at $\sigma = 0.35$ before going back up to above 0.1. The rate then plateaus around 0.13 after $\sigma = 2$.

As for the Vovk Polynomial kernel, we see that in Fig 2, the misclassification rate is monotonically decreasing in a staircase shape as p increases. And no minimum is found within range.

From the above analysis the classification result is very much dependant on the choice of hyperparameters. And since we are only considering the local structure of the image, i.e. the

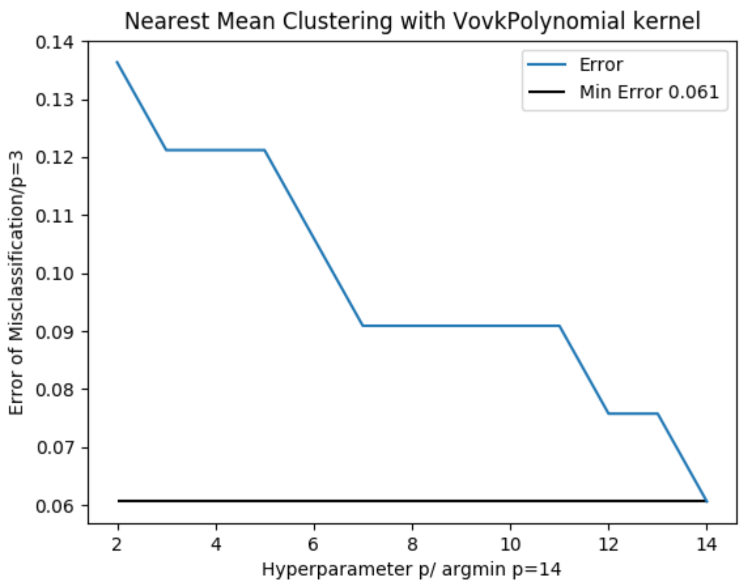

Figure 2: Missclassification rate of two classes of images with Vovk Polynomial kernel for varying hyperparameter p.

average of kernel evaluation on pairs with 50% overlapped patches, the increase in σ may bring no difference in the case of Gaussian kernel where σ indicates the range of Gaussian, hence the constant misclassification rate after $\sigma = 2$.

And the plateaus may be resulted from the fact that the rank of distance of each image to empirical class mean stays roughly unchanged as hyperparameter changes, for which we investigate in the following part.

(b) The image class with 121 images is analysed then visualised. For $\{x_i\}_{i=1}^n, x_i \in \mathcal{X}\ \forall i$, and RKHS $\mathcal{H}$ with PSD kernel k and feature map $\phi(\cdot)$, the distance from any sample x in the feature space to the class empirical mean is as follows,

$$\begin{aligned}\|\phi(x) - \hat{\mu}_X\|^2_{\mathcal{H}} &= \langle \phi(x), \phi(x)\rangle_{\mathcal{H}} + \langle \hat{\mu}_X, \hat{\mu}_X\rangle_{\mathcal{H}} - 2\langle \phi(x), \hat{\mu}_X\rangle_{\mathcal{H}} \\ &= k(x,x) + \frac{1}{n^2}\sum_{i,j=1}^{n} k(x_i, x_j) - \frac{2}{n}\sum_{i=1}^{n} k(x, x_i) \end{aligned} \tag{1}$$

We then investigate the change of distance to empirical class mean as in Equ 1 with varying hyperparameter for Gaussian and Vovk Polynomial kernel. The furthest to the empirical class mean is ranked highest, i.e. in a decreasing order. Then the result is visualised in Fig 3 and Fig 4. 1-step change of rank is also plotted for both kernels in Fig 5 and Fig 6.

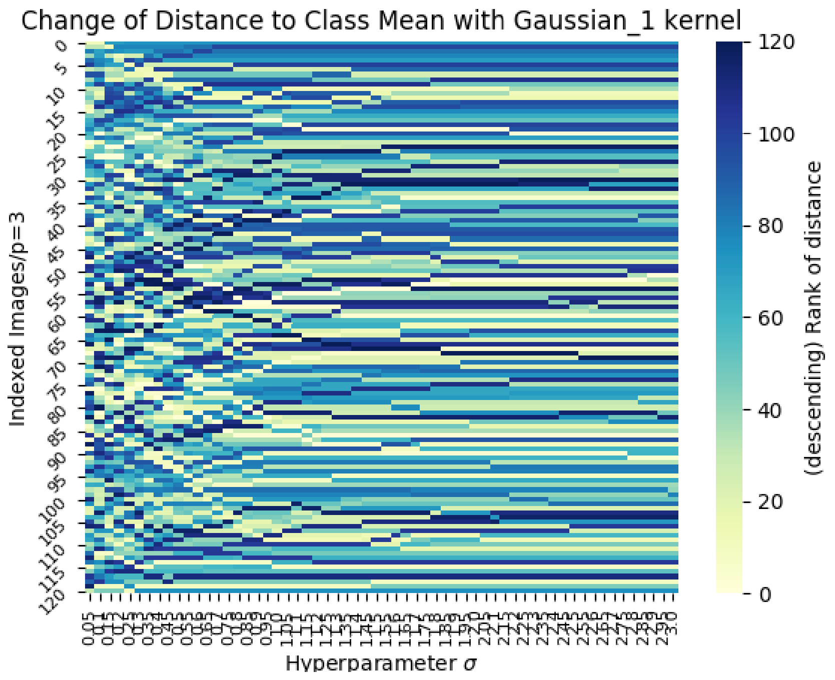

Figure 3: Shift of rank of distance (decreasing) to empirical class mean as σ increases.

In both cases, the rank shuffles frequently for small value of σ and p. Then the changes slows down as the σ reach value 1, and p reaches 8. And the reshuffle diminishes when σ is above 2, and p is above 11. So we observe increasingly long stripes of colours in the rank shift plot, and growing number of 0 values in the 1-step change of rank visualisation.

However, note that there are still images that have a drastic change of rank for large hyper-parameter values.

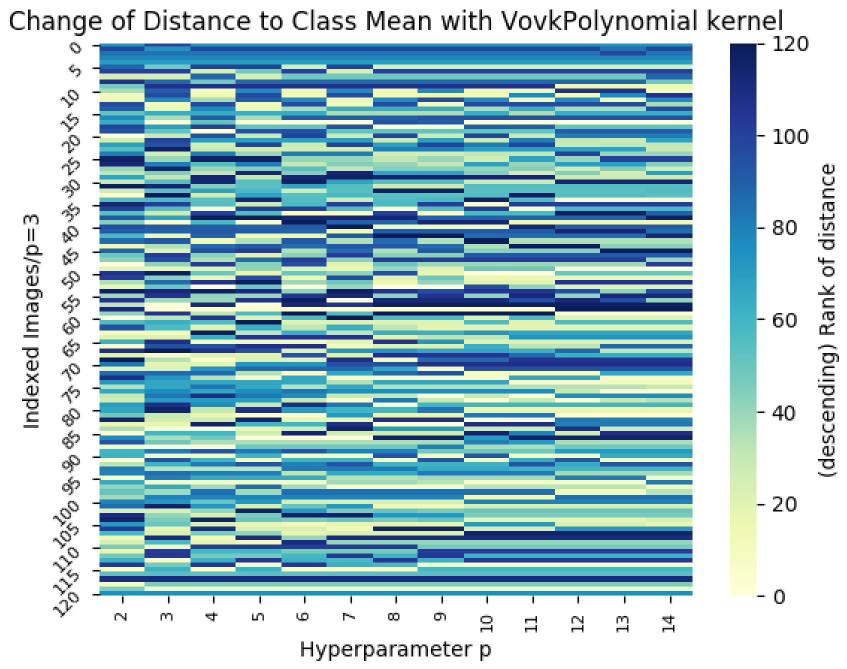

Figure 4: Shift of rank of distance (decreasing) to empirical class mean as p increases.

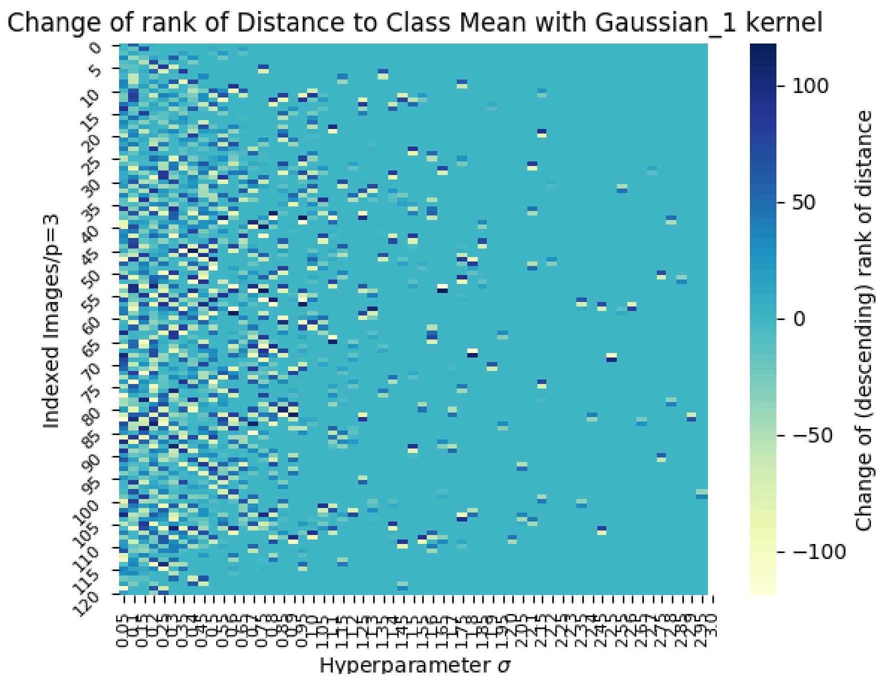

Figure 5: 1-step change of rank of distance (decreasing) to empirical class mean as σ increases.

Q2 For RKHS $\mathcal{F}$ on $\mathcal{X}$ and $\mathcal{G}$ on $\mathcal{Y}$ with PSD kernels $k(\cdot,\cdot)$ and $h(\cdot,\cdot)$ respectively, there is orthonormal basis $\{f_i\}_{i\in I}$ and $\{g_j\}_{j\in J}$ respectively, where I, J are finite or countable index sets. Let $\mathcal{HS} = \mathcal{HS}(\mathcal{G},\mathcal{F}) = \{L \text{ compact} : L : \mathcal{G} \to \mathcal{F}\}$ be the Hilbert space with all the compact operator from $\mathcal{G}$ to $\mathcal{F}$, equipped with Hilbert Schmidt inner product/norm.

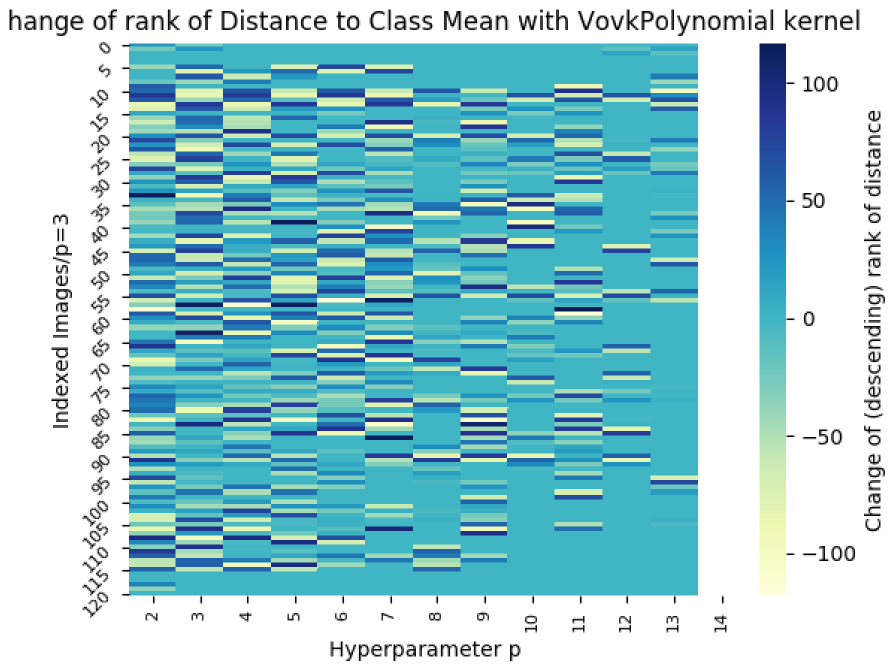

Figure 6: 1-step change of rank of distance (decreasing) to empirical class mean as p increases.

Then the cross-covariance operator $\mathcal{C}_{XY} \in \mathcal{HS}$.

For i.i.d samples $\{X_i\}_{i=1}^n \in \mathcal{X}$ and $\{Y_j\}_{j=1}^n \in \mathcal{Y}$, the empirical cross-covariance operator is defined by

$$\hat{\mathcal{C}}_{XY} = \frac{1}{n-1}\sum_{i=1}^n (k(X_i,\cdot) - \hat{\mu}_X) \otimes (h(Y_i,\cdot) - \hat{\mu}_Y),$$

where $\hat{\mu}_X = \frac{1}{n}\sum_{i=1}^n k(X_i,\cdot)$ and $\hat{\mu}_Y$ similarly are empirical means in RKHS.

We prove $\hat{\mathcal{C}}_{XY} \xrightarrow{p} \mathcal{C}_{XY}$ as $n \to \infty$ by proving $\mathbb{P}(\|\hat{\mathcal{C}}_{XY} - \mathcal{C}_{XY}\|_{\mathcal{HS}} > \epsilon) \to 0$.

Proof. Let $Z_i = \frac{1}{n-1}I_i - \frac{1}{n}\mathcal{C}_{XY}$ be an operator mapping from $\mathcal{G}$ to $\mathcal{F}$, then

$$\hat{\mathcal{C}}_{XY} - \mathcal{C}_{XY} = \sum_{i=1}^n Z_i.$$

By triangle inequality,

$$\|Z_i\|_{\mathcal{HS}}^2 \le \left[\frac{1}{n-1}\|I_i\|_{\mathcal{HS}} + \frac{1}{n}\|\mathcal{C}_{XY}\|_{\mathcal{HS}}\right]^2.$$

Let $I_i = (k(X_i,\cdot) - \hat{\mu}_X) \otimes (h(Y_i,\cdot) - \hat{\mu}_Y)$, then

$$\|I_i\|_{\mathcal{HS}}^2 = \|k(X_i,\cdot) - \hat{\mu}_X\|_{\mathcal{F}}^2 \|h(Y_i,\cdot) - \hat{\mu}_Y\|_{\mathcal{G}}^2.$$

Consider $\|k(X_i,\cdot) - \hat{\mu}_X\|_{\mathcal{F}}^2$. Since $\|k(X_i,\cdot)\|_{\mathcal{F}}^2 = k(X_i,X_i) \le k_{\max}$ and

$$\|\hat{\mu}_X\|_{\mathcal{F}}^2 = \frac{1}{n^2}\|\sum_{i=1}^n k(X_i,\cdot)\|_{\mathcal{F}}^2 \le \frac{1}{n^2}\left[\sum_{i=1}^n \|k(X_i,\cdot)\|_{\mathcal{F}}\right]^2 \le k_{\max}.$$

It gives,

$$\begin{aligned}\|k(X_i,\cdot) - \hat{\mu}_X\|_{\mathcal{F}}^2 &\le (\|k(X_i,\cdot)\|_{\mathcal{F}} + \|\hat{\mu}_X\|_{\mathcal{F}})^2 \\ &\le 4k_{\max};\end{aligned}$$

and similarly $\|h(Y_i,\cdot)-\hat{\mu}_Y\|_{\mathcal{G}}^2 \le 4h_{\max}$. Thus

$$\|I_i\|_{\mathcal{HS}}^2 \le 16k_{\max}h_{\max}\ \forall i.$$

Moreover, by Jensen's inequality

$$\begin{aligned}\|\mathcal{C}_{XY}\|_{\mathcal{HS}}^2 &\le \mathbb{E}_{XY}\{(\|k(X,\cdot)-\mu_X)\otimes(h(Y,\cdot)-\mu_Y)\|_{\mathcal{HS}}^2\}\\ &= \mathbb{E}_{XY}\{\|k(X,\cdot)-\mu_X\|_{\mathcal{F}}^2\|(h(Y,\cdot)-\mu_Y\|_{\mathcal{G}}^2\} \le 16k_{\max}h_{\max},\end{aligned}$$

where the last inequality follows from that

$$\|k(X,\cdot)-\mu_X\|_{\mathcal{F}}^2 \le [\|k(X,\cdot)\|_{\mathcal{F}}+\|\mu_X\|_{\mathcal{F}}]^2 \le 4k_{\max},$$

where $\|k(X,\cdot)\|_{\mathcal{F}}^2 \le k_{\max}$ and $\|\mu_X\|_{\mathcal{F}}^2 = \mathbb{E}_{X'}\mathbb{E}_X k(X',X) \le k_{\max}$. Since for any $X_i, X_j \in \mathcal{X}$, by Cauchy Schwartz

$$k(X_i,X_j) \le \|k(X_i,\cdot)\|_{\mathcal{F}}\|k(X_j,\cdot)\|_{\mathcal{F}} = \sqrt{k(X_i,X_i)k(X_j,X_j)} \le k_{\max}.$$

Then

$$\|Z_i\|_{\mathcal{HS}}^2 \le \left[\frac{4}{n-1}\sqrt{k_{\max}h_{\max}}+\frac{4}{n}\sqrt{k_{\max}h_{\max}}\right]^2 \le \left(\frac{4(2n-1)}{n(n-1)}\right)^2 m^2 < \infty,$$

where we denote $m = k_{\max}\vee h_{\max}$. Since Z_i finite $\forall i$, we conclude that $Z_i \in HS(\mathcal{G},\mathcal{F})$. For $p\ge 2$,

$$\sum_{i=1}^n \mathbb{E}\|Z_i\|_{\mathcal{HS}}^p = \sum_{i=1}^n \mathbb{E}(\|Z_i\|_{\mathcal{HS}}^2)^{p/2} \le n\cdot\frac{4^p(2n-1)^p m^p}{n^p(n-1)^p} \le \frac{p!}{2}\left(\frac{4m(2n-1)}{\sqrt{n}(n-1)}\right)^2\left(\frac{4m(2n-1)}{n(n-1)}\right)^{p-2}$$

which implies that the condition in Pinelis-Sakhanenko's inequality holds for $Z_i,\ \forall i=1,\dots,n$ with

$$\sigma^2 = \left(\frac{4m(2n-1)}{\sqrt{n}(n-1)}\right)^2,\ \ C=\frac{4m(2n-1)}{n(n-1)}.$$

Therefore, $\forall\epsilon>0$,

$$\mathbb{P}(\|\hat{\mathcal{C}}_{XY}-\mathcal{C}_{XY}\|_{\mathcal{HS}}>\epsilon) = \mathbb{P}(\|\sum_{i=1}^n Z_i\|_{\mathcal{HS}}>\epsilon) \le 2\exp\left\{-\frac{\epsilon^2}{2(\sigma^2+C\epsilon)}\right\},$$

which goes to zero as n goes to infinity. Therefore $\hat{\mathcal{C}}_{XY} \xrightarrow{p} \mathcal{C}_{XY}$ as $n\to\infty$. □

Appendices

```
# Q1 a) Nearest Mean clustering ---------------------------------------------------------

# Image pre-processing
def Wrapper_Dat(cla, dat):
    """Wrapper function to process image data from lfw into patched format.
    If teo classes are inputed, relabel the data into class +/-1
    -----------
    Input
    cla: list; containing a pair or one label of class in lfw
    dat: 3d array; containing images in row
    p: integer; dimension of the patch
    -----------
    Output
    processed images a/na relabeled classes
    """
    target = dat.target

    cla_index = np.concatenate([np.where(target == i)[0] for i in cla])
    arr_dat = dat.images[cla_index]

    # scale the original image
    arr_dat = ((arr_dat - np.mean(arr_dat, axis=(-1,-2), keepdims=True))
                  /np.std(arr_dat, axis=(-1,-2), keepdims=True))
    proc_imag = Wrapper_extra_patches3(arr_dat)

    # change the label to +/-1; if input class is 2-dim
    if len(cla) is 2:

        labels = target[cla_index]
        labels = [-1 if i==cla[0] else 1 for i in labels]

        return proc_imag, np.array(labels)

    elif len(cla) is 1:

        return proc_imag

def extra_patches3(imag):
    """Extracts 3 by 3 patches of any nd array in place using strides,
    with stepsize 1
    -----------
    Input
    imag: np array storing images
    -----------
    Returns
     patches : strided 4d array
     4d array indexing patches in row and column order in the first two axes
     and containing patches on the last 2 axes.
    """
    import numpy as np
    from numpy.lib.stride_tricks import as_strided

    # pad the image with np.nan, width=1
    imag = np.pad(imag, 1, mode='constant', constant_values=np.nan)

    # dimension of image and patches
    i_h, i_w = imag.shape
    p = 3
    n_patches = (i_h-p+1) * (i_w-p+1)

    imag_ndim = imag.ndim
    patch_shape = tuple([p] * imag_ndim)
    patch_strides = imag.strides

    # the shape of produced array of patches
    patch_indices_shape = np.array(imag.shape) - np.array(patch_shape) + 1
    # output of strided array of patches
    shape_out = tuple(list(patch_indices_shape) + list(patch_shape))
    strides_out = tuple(list(patch_strides) * 2)
    patches = as_strided(imag, shape=shape_out, strides=strides_out)

    # renormalise by deviding the norm
    return patches/np.sqrt(np.sum(patches**2, axis=(-1,-2), keepdims=True)) # 4d array


def Wrapper_extra_patches3(arr_imag):
    """Wrapper function of extra_patches3 for each image in the data set
    """
    from itertools import starmap, product

    return np.array(list(starmap(extra_patches3, product(arr_imag))))

# kernel function
def moving_window3(X, Y, kernel, hyper):
    """Compute the distance between patches in one image with the other only if
    the overlap between two patches are above 50%
    -----------
    Input
    X, Y: 4d array; storing patched image processed by 'extra_patches3'
    kernel: fuction; kernel function for patches
    hyper: scalar; hyperparameter of the kernel function
    -----------
    Return:
    scalar; mean of kernel evaluation between patches
    """
    # X, Y 4d array of same dimension, n, m, p, p
    n, m = X.shape[:2]
    X_true = X[1:(n-1), 1:(m-1), :, :]

    # distance^2 between >100%, <50% overlapped patch
    ker = [kernel(X_true, Y[1:(n-1), 1:(m-1), :, :], hyper)]
    for i in range(-1, 2, 2):
        ker += [kernel(X_true, Y[(1+i):(n-1+i), 1:(m-1), :, :], hyper)]
        ker += [kernel(X_true, Y[1:(n-1), (1+i):(m-1+i), :, :], hyper)]

    return np.nanmean(np.array(ker))

def ker_Gauss3(u, v, l):
    """Gaussian kernel for vectors
    ----------
    Input
    u, v: 4d array; each position of first two axis stores patches as from 'extra_patches3'
    l: positive scalar; hyperparameter sigma
    ----------
    Output
    kernel evaluation between u and v
    """
    # u, v 3 by 3 array
    return np.exp(-np.sum((u - v)**2, axis=(-1,-2))/l**2)

def Wrapper_ker_Gauss(X_arr, Y_arr, l):
    """Wrapper function for evaluating the between image kernel value with 'moving_window3'
    for Guassian kernel
    ----------
    Input
    X_arr, Y_arr: 5d array; first axis stores patched images in class X and Y respectively
    l: positive scalar; hyperparameter of Gaussian kernel
    -----------
    Output
    2d array; storing between image kernel evaluations from two classes X and Y.
    """
    from itertools import starmap, product

    ker = np.array(list(starmap(moving_window3, product(X_arr, Y_arr, [ker_Gauss3],
        np.array([l])))))

    return ker.reshape(X_arr.shape[0], Y_arr.shape[0])

def ker_Poly3(u, v, p):
    """Vovk Polynomial kernel for vectors
    ----------
    Input
    u, v: 4d array; each position of first two axis patches as from 'extra_patches3'
    p: non-negative integer; hyperparameter p
    ----------
    Output
    kernel evaluation between u and v
    """

    dotprod = np.sum(u * v, axis=(-1,-2))

    # runtime error arises when doing operations on nan
    if np.all(abs(abs(dotprod) - 1)<10**(-6)):
        dotprod[:] = p
        return dotprod
    else:
        return (1 - dotprod**p)/(1 - dotprod)

def Wrapper_ker_Poly(X_arr, Y_arr, p):
    """Wrapper function for evaluating the between image kernel value with 'moving_window3'
    for Vovk polynomial kernel
    ----------
    Input
    X_arr, Y_arr: 5d array; first axis stores patched images in class X and Y respectively
    p: non negative integer; hyperparameter of Vovk Polynomial kernel
    -----------
    Output
    2d array; storing between image kernel evaluations from two classes X and Y.
    """
    from itertools import starmap, product

    ker = np.array(list(starmap(moving_window3, product(X_arr, Y_arr, [ker_Poly3],
        np.array([p])))))

    return ker.reshape(X_arr.shape[0], Y_arr.shape[0])

# nearest mean algorithm
def train_test_split(dat, lab, test_ratio=0.25, seed=17671):
    """Split dataset into train and test set (stratified)
    ----------------
    Input:
    dat: n darray; X
    lab: 1 darray; y
    test_ratio: [0,1]; test to train ratio
    ----------------
    Output:
    dat_train, dat_test, lab_train, lab_test
    """
    import numpy.random as rm
    rs = rm.RandomState(seed=seed)
    # split into -/+1 classes
    ind = lab<0
    neg_dat = dat[ind]; neg_lab = lab[ind]
    pos_dat = dat[~ind]; pos_lab = lab[~ind]

    # select test set in two classes respectively
    ind_neg = np.arange(neg_dat.shape[0])
    ind_pos = np.arange(pos_dat.shape[0])

    ind_test_neg = rs.choice(ind_neg, round(neg_dat.shape[0]*test_ratio), replace=False)
    ind_train_neg = np.delete(ind_neg, ind_test_neg)

    ind_test_pos = rs.choice(ind_pos, round(pos_dat.shape[0]*test_ratio), replace=False)
    ind_train_pos = np.delete(ind_pos, ind_test_pos)

    # split into test/train classes
    dat_test = np.concatenate((neg_dat[ind_test_neg], pos_dat[ind_test_pos]))
    dat_train = np.concatenate((neg_dat[ind_train_neg], pos_dat[ind_train_pos]))

    lab_test = np.concatenate((neg_lab[ind_test_neg], pos_lab[ind_test_pos]))
    lab_train = np.concatenate((neg_lab[ind_train_neg], pos_lab[ind_train_pos]))

    return dat_train, dat_test, lab_train, lab_test

def NearestMean(kernel, hyper, arr_train, arr_test, lab_train, lab_test):
    """Find the kernel population mean for two classes, and make classification
    on the test set with 0-1 loss
    -----------
    Input:
    kernel: function to compute the kernel martix
    hyper: float; e.g. sigma in Gaussian kernel
    -----------
    Output:
    pred_class: predicted label for the test set;
    conf_mat: confusion matrix of the true label and the predicted;
    error: missclassification error
    """
    X_arr_train = arr_train[np.where(lab_train==1)[0]]
    Y_arr_train = arr_train[np.where(lab_train==-1)[0]]

    mu_norm = kernel(Y_arr_train, Y_arr_train, hyper)
    nu_norm = kernel(X_arr_train, X_arr_train, hyper)

    # predict for training data
    pred_class = np.sign(1/2 * (np.mean(mu_norm) - np.mean(nu_norm))
                 - np.mean(kernel(arr_test, Y_arr_train, hyper), axis=1)
                 + np.mean(kernel(arr_test, X_arr_train, hyper), axis=1))

    # confusion matrix
    conf_mat = pd.crosstab(pd.Series(lab_test, name="actual"),
                              pd.Series(pred_class.astype('int'), name="predict"))

    misclass = (conf_mat.iloc[0,1]+conf_mat.iloc[1,0])/conf_mat.values.sum()

    return {'pred_class': pred_class, 'conf_mat': conf_mat, 'error': misclass}

# visualisations
def Wrapper_Plot(kernel_dic, hyper_dic, arr_train, arr_test, lab_train, lab_test, p=3):
    """Wrapper function to plot the missclassification rate with varying hyper
    parameters.
    -------------
    Input:
    kernel_dic: dictionary; storing kernel name and corresponding function
    hyper_dic: dictionary; storing hyper parameter name and corresponding list
    of values
    -------------
    Output:
    Plot of misclassification rate against hyper parameter values
    """
    for j in range(len(kernel_dic)):
        ker_name, kernel = list(kernel_dic.items())[j]
        hyper_name, hyper_lis = list(hyper_dic.items())[j]
        error_lis = []

        for i in hyper_lis:
            pred_class = NearestMean(kernel, i, arr_train, arr_test, lab_train, lab_test)
            error_lis += [pred_class['error']]

        min_error = min(error_lis)
        argmin = hyper_lis[error_lis.index(min_error)]

        plt.figure()
        plt.plot(hyper_lis, error_lis, label='Error')
        plt.hlines(min_error, xmin=hyper_lis.min(), xmax=hyper_lis.max(),
                label='Min Error {}'.format(round(min_error,3)))
        plt.title('Nearest Mean Clustering with {} kernel'.format(ker_name))
        plt.xlabel('Hyperparameter {a}/ argmin {a}={b}'.format(a=hyper_name,
            b=round(argmin,3)))
        plt.ylabel('Error of Misclassification/p={}'.format(p))
        plt.legend()
        plt.savefig('{a}_{b}.png'.format(a=ker_name, b=p))
        plt.show()
        plt.close()

# Q1 b) Distance to class population mean
    ----------------------------------------------------------
def dist_to_mean(dat, kernel, hyper):
    """Find the distance between an image and empirical mean embedding in the feature space,
    which is then ranked in decreasing order.
    """
    ker_eval = kernel(dat, dat, hyper)
    ker_dist = (np.diag(ker_eval) - 2 * np.mean(ker_eval, axis=1)
                           + np.mean(ker_eval))

    return np.sqrt(ker_dist)

def Wrapper_HeatMap(kernel_dic, hyper_dic, dat, p=3):
    """Wrapper function: visualise the change of order of rank with varying hyperparameters
    """
    import seaborn as sns
    for j in range(len(kernel_dic)):
        ker_name, kernel = list(kernel_dic.items())[j]
        hyper_name, hyper_lis = list(hyper_dic.items())[j]
        dist_lis = []

        for i in hyper_lis:
            dist_lis += [(-dist_to_mean(dat, kernel, i)).argsort()]

        dist_arr = np.array(dist_lis).T
        dist_diff = np.diff(dist_arr)

        # rank shift
        plt.figure()
        ax = sns.heatmap(dist_arr, cbar_kws={'label': '(descending) Rank of distance'}
                         , cmap='YlGnBu'
                         , xticklabels=np.round(hyper_lis,2))
        ax.set_xticklabels(ax.get_xticklabels(), rotation=90, fontsize=8)
        ax.set_yticklabels(ax.get_yticklabels(), rotation=45, fontsize=8)
        ax.set_ylabel('Indexed Images/p={}'.format(p))
        ax.set_xlabel('Hyperparameter {}'.format(hyper_name))
        ax.set_title('Change of Distance to Class Mean with {} kernel'.format(ker_name))
        plt.savefig('{a}_{b}_dist.png'.format(a=ker_name, b=p))
        plt.show()
        plt.close()

        # 1 step chanage of rank
        plt.figure()
        ax = sns.heatmap(dist_diff, cbar_kws={'label': 'Change of (descending) rank of
            distance'}
                         , cmap='YlGnBu'
                         , xticklabels=np.round(hyper_lis,2))
        ax.set_xticklabels(ax.get_xticklabels(), rotation=90, fontsize=8)
        ax.set_yticklabels(ax.get_yticklabels(), rotation=45, fontsize=8)
        ax.set_ylabel('Indexed Images/p={}'.format(p))
        ax.set_xlabel('Hyperparameter {}'.format(hyper_name))
        ax.set_title('Change of rank of Distance to Class Mean with {}
            kernel'.format(ker_name))
        plt.savefig('{a}_{b}_diff.png'.format(a=ker_name, b=p))
        plt.show()
        plt.close()



if __name__=='__main__':
    import matplotlib.pyplot as plt
    import numpy as np
    import pandas as pd
    import time

    # can we use this function, but what should be the `resize` argument? By default?
    from sklearn.datasets import fetch_lfw_people
    # Download the data, if not already on disk and load it as numpy arrays
    lfw_people = fetch_lfw_people(min_faces_per_person=40, resize=0.3)
    target = lfw_people.target
    print('target has classes', np.unique(target, return_counts=True))

    # preprocessing
        ------------------------------------------------------------------------------
    # 121, 144
    cla=[3,17]; test_ratio = 0.25
```

```
proc_imag, labels = Wrapper_Dat(cla, lfw_people)
print("dim of classes", np.unique(labels, return_counts=True))
dat_train, dat_test, lab_train, lab_test = train_test_split(proc_imag,
                                        labels, test_ratio, seed=21203)
print("dim of train ", dat_train.shape)
print("train to test ratio ", np.unique(lab_train, return_counts=True),
                           np.unique(lab_test, return_counts=True))

# test the kernel function
    --------------------------------------------------------------------
# t0 = time.time()
# ker = NearestMean(Wrapper_ker_Gauss, 0.3, dat_train, dat_test, lab_train, lab_test) #
    5d array
# t1 = time.time()
# print("time taken to run, nearest mean", t1-t0)
# print("Mis classification rate", ker)

# Varying hyperparams
    --------------------------------------------------------------------------

kernel_dic = {
     'Gaussian_1': Wrapper_ker_Gauss,
     'VovkPolynomial': Wrapper_ker_Poly
     }
hyper_dic = {
     '$\sigma$': np.linspace(0.05,3,num=60),
     'p': np.arange(2,15)
     }

# a) Nearest Mean algorithm
Wrapper_Plot(kernel_dic, hyper_dic, dat_train, dat_test, lab_train, lab_test)

# b) change of distance to mean
cla_ = [3] # 121 images

proc_dat_ = Wrapper_Dat(cla_, lfw_people)
Wrapper_HeatMap(kernel_dic, hyper_dic, proc_dat_)
```