Q1 (a) $k(x,x') = \min(x,x')$ is a PSD kernel, for $\mathcal{X} = \{1,\cdots,n\}$.

Proof.

$$\phi(x) = [\mathbb{1}_{1\le x},\cdots,\mathbb{1}_{n\le x}]^T \tag{1}$$

We prove by showing that the feature map $\phi(x) : \mathcal{X} \to \mathbb{R}^n$ in Equ 1 gives us a feature space, $\mathcal{H} = \text{span}\{\phi(x) : x \in \mathcal{X}\} \subset \mathbb{R}^n$, where $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ defined by $\min(\cdot,\cdot)$ is indeed an inner product in $\mathcal{H}$.

Now, $\forall x, x' \in \{1,\cdots,n\}$,

$$\begin{aligned}\langle\phi(x),\phi(x')\rangle_{\mathcal{H}} &= \phi(x)^T\phi(x')\\ &= \sum_{s=1}^{n}\mathbb{1}_{s\le x}\mathbb{1}_{s\le x'}\\ &= \sum_{s=1}^{n}\mathbb{1}_{s\le\min(x,x')}\\ &= \min(x,x')\end{aligned}$$

The last equality follows from the fact that

$$\text{for } x > 0,\ x = \int_0^x \mathrm{d}s = \int_0^\infty \mathbb{1}_{s\le x}\mathrm{d}s,$$

and in the discrete case, where $x \in \{1,\cdots,n\}$, $x = \sum_{s=1}^n \mathbb{1}_{s\le x}$. Moreover, the symmetry, linearity and scalar multiplication of $\langle\cdot,\cdot\rangle_{\mathcal{H}}$ follows from the vector product and the definition of $\mathcal{H}$.

Finally, $\forall v = \sum_{i=1}^m \alpha_i\phi(x_i) \in \mathcal{H}$, where $\alpha_i \in \mathbb{R}$,

$$\begin{aligned}\langle v,v\rangle_{\mathcal{H}} &= \sum_{i,j=1}^{m}\alpha_i\alpha_j\phi(x_i)^T\phi(x_j)\\ &= \sum_{s=1}^{n}\sum_{i,j=1}^{m}\alpha_i\alpha_j\mathbb{1}_{s\le x_i}\mathbb{1}_{s\le x_j}\\ &= \sum_{s=1}^{n}(\sum_{i=1}^{m}\alpha_i\mathbb{1}_{s\le x_i})^2 \ge 0,\end{aligned}$$

and equality holds iff $v = \sum_{i=1}^m \alpha_i\phi(x_i) \equiv 0$. Since a finite dimensional vector space is complete, and therefore $\mathcal{H}$ is a Hilbert space, by Corollary 13 in the lecture, $k(\cdot,\cdot) = \min(\cdot,\cdot)$ is indeed a PSD kernel. □

(b) $\max(x,x')$ is not a PSD kernel for $\mathcal{X} = \{1,\cdots,n\}$.

Proof. Assume that $k(x,x') = \max(x,x')$ is a PSD kernel, then for data $x_i, x_j \in \{1,\cdots,n\}$, WLOG $x_i > x_j$, then the Gram matrix is

$$K(x_i,x_j) = \begin{bmatrix} x_i & x_i\\ x_i & x_j\end{bmatrix}$$

Since its determinant $|K(x_i,x_j)| = x_ix_j - x_i^2 \le 0$, the product of its eigenvalues is negative, which means there exists an eigenvalue that is negative. Therefore, the Gram matrix is not PSD, by contradiction, $\max(x,x')$ is not a PSD kernel. □

Q2 (a) *Proof.*

$$\begin{aligned}\|\phi(x)-\mu\|_{\mathcal{H}}^2-\|\phi(x)-\nu\|_{\mathcal{H}}^2&=\langle\phi(x)-\mu,\phi(x)-\mu\rangle_{\mathcal{H}}-\langle\phi(x)-\nu,\phi(x)-\nu\rangle_{\mathcal{H}}\\&=-2\langle\phi(x),\mu\rangle_{\mathcal{H}}+2\langle\phi(x),\nu\rangle_{\mathcal{H}}+\langle\mu,\mu\rangle_{\mathcal{H}}-\langle\nu,\nu\rangle_{\mathcal{H}}\\&=\|\mu\|_{\mathcal{H}}^2-\|\nu\|_{\mathcal{H}}^2+2(\langle\phi(x),\nu\rangle_{\mathcal{H}}-\langle\phi(x),\mu\rangle_{\mathcal{H}})\\&=2(\langle\omega,\phi(x)\rangle_{\mathcal{H}}+b)\end{aligned}$$

where $\omega=\nu-\mu$ and $b=\frac{1}{2}(\|\mu\|_{\mathcal{H}}^2-\|\nu\|_{\mathcal{H}}^2)$. Therefore,

$$\begin{aligned}h(x)&=\begin{cases}-1 & 2(\langle\omega,\phi(x)\rangle_{\mathcal{H}}+b)\le 0\\ 1 & \text{otherwise}\end{cases}\\&=\operatorname{sign}(\langle\omega,\phi(x)\rangle_{\mathcal{H}}+b)\end{aligned}$$

□

(b) The decision rule can be further expanded, written in terms of kernel function $k(\cdot,\cdot)$ and the training data $\{x_i\}_{i=1}^{n+m}$, where the empirical mean of the two classes in RKHS are

$$\mu=\frac{1}{m}\sum_{i=1}^{m}\phi(x_i),\ \nu=\frac{1}{n}\sum_{i=m+1}^{m+n}\phi(x_i).$$

are empirical mean of class '-1' and '+1' respectively.

$$\begin{aligned}\langle\omega,\phi(x)\rangle_{\mathcal{H}}+b&=\frac{1}{2}(\|\mu\|_{\mathcal{H}}^2-\|\nu\|_{\mathcal{H}}^2)+\langle\phi(x),\nu\rangle_{\mathcal{H}}-\langle\phi(x),\mu\rangle_{\mathcal{H}}\\&=\frac{1}{2}\left(\frac{1}{m^2}\sum_{i,j=1}^{m}\langle\phi(x_i),\phi(x_j)\rangle_{\mathcal{H}}-\frac{1}{n^2}\sum_{i,j=m+1}^{m+n}\langle\phi(x_i),\phi(x_j)\rangle_{\mathcal{H}}\right)\\&+\frac{1}{n}\sum_{i=m+1}^{m+n}\langle\phi(x_i),\phi(x)\rangle_{\mathcal{H}}-\frac{1}{m}\sum_{i=1}^{m}\langle\phi(x_i),\phi(x)\rangle_{\mathcal{H}}\\&=\frac{1}{2}\left(\frac{1}{m^2}\sum_{i,j=1}^{m}k(x_i,x_j)-\frac{1}{n^2}\sum_{i,j=m+1}^{m+n}k(x_i,x_j)\right)\\&+\frac{1}{n}\sum_{i=m+1}^{m+n}k(x,x_i)-\frac{1}{m}\sum_{i=1}^{m}k(x,x_i)\end{aligned}$$

Therefore,

$$h(x)=\operatorname{sign}\left(\frac{1}{2}\left(\frac{1}{m^2}\sum_{i,j=1}^{m}k(x_i,x_j)-\frac{1}{n^2}\sum_{i,j=m+1}^{m+n}k(x_i,x_j)\right)+\frac{1}{n}\sum_{i=m+1}^{m+n}k(x,x_i)-\frac{1}{m}\sum_{i=1}^{m}k(x,x_i)\right)$$

Q3 The Nearest Mean clustering algorithm (as in Q2) is implemented with `numpy` and `pandas` only for data manipulation. The train and test sets are randomly chosen with a ratio 8:2, and the percentage of spam contained in each set is of roughly 60%.

Notably, to boost efficiency, the pairwise distance of two matrices in the Gaussian kernel matrix is calculated using the expansion of inner product, i.e. for $u, v \in \mathbb{R}^d$

$$\|u - v\| = \sqrt{\|u\|^2 + \|v\|^2 - 2\langle u, v\rangle};$$

for matrix $X \in \mathbb{R}^{n\times d}$ and $Z \in \mathbb{R}^{m\times d}$ with row containing observations $x_i^T \in \mathbb{R}^d$ and $z_i^T \in \mathbb{R}^d$ with d number of features. Then, the pairwise distance $D \in \mathbb{R}^{n\times m}$ between X and Z can be expanded into

$$D = \sqrt{\bar{x}_n \mathbb{1}_m^T + \mathbb{1}_n \bar{z}_m^T - 2XZ^T},$$

where $\bar{x}_n \in \mathbb{R}^n$ is the vector with entries the mean of row vector $x_i, i = 1, \cdots, n$; similarly for $\bar{z}_m$. To evaluate the performance of Nearest Mean algorithm with different kernels, in particular Gaussian kernel and Polynomial kernel, we investigate the misclassification error of the algorithm for a sequence of hyperparameter values, $\sigma > 0$, $c > 0$ for Gaussian kernel $k_G(\cdot,\cdot)$ and Polynomial kernel $k_P(\cdot,\cdot)$ respectively.

$$k_G(x, x') = \exp\{-\frac{\|x - x'\|^2}{\sigma^2}\}$$
$$k_P(x, x') = (x^T x' + c)^2$$

For the implementation, 40 evaluation points are taken for $\sigma \in (0, 20)$ shown in Fig 1 and $c \in (0, 20)$ shown in Fig 2.

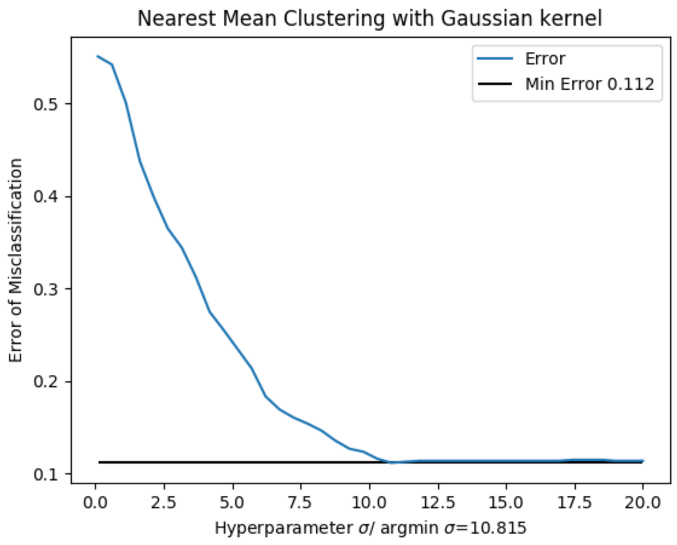

Figure 1: Missclassification error of Nearest Mean algorithm with Gaussian kernel for $\sigma \in (0, 20)$.

As shown in Fig 1, the decrease of error is slow when $\sigma < 1$ and is faster but slowing down as σ increases.

Notably, the rate of decline of error for Gaussian kernel decreases, whereas the rate of decline of error for Polynomial kernel first increases then decreases (an elbow as shown in the plot). This pattern persists when experimenting with different seeds for random selection of train and test sets. And the smallest misclassification error with Polynomial kernel is smaller (only marginally) than that of Gaussian kernel.

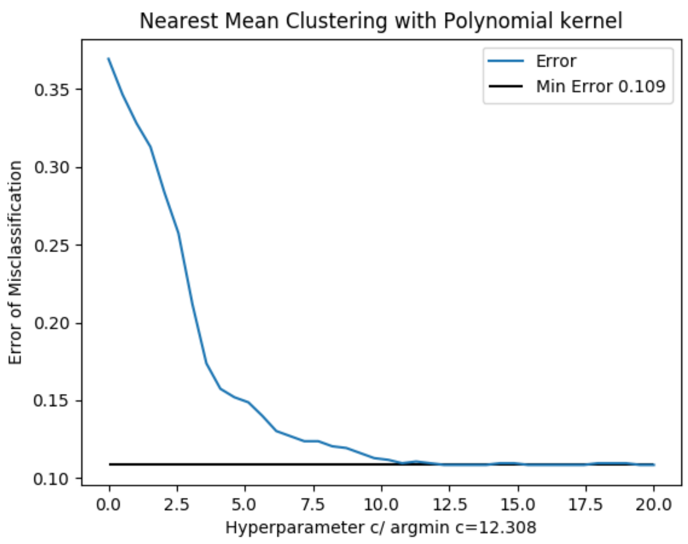

Figure 2: Missclassification error of Nearest Mean algorithm with Polynomial kernel for $c \in (-20, 20)$.

However, we do note that the error somehow converges for large value taken for both σ and c, where the kernel matrix in the former case have entries all close to 1, and in the latter case all dominated by c^2. And it is not immediately clear why this is the case.

Appendices

```
# Pairwise distances between obervation
# A, B two data frame objects
def pdist(A, B):

    n = A.shape[0]
    m = B.shape[0]

     # binomial formula ~ Smola's Blog
    tmp = (A**2).sum(axis=1).values.reshape(n,1) @ np.ones(m).reshape(1,m)
    tmp += np.ones(n).reshape(n,1) @ (B**2).sum(axis=1).values.reshape(1,m)
    tmp -= 2 * A.values @ B.transpose().values

    return tmp

# Gaussian kernel
# A, B: data frame objects storing objects with numeric features
# l: positive real, spatial range
def ker_Gauss(A, B, l):

    dist = pdist(A, B)

    return np.exp(-dist/(l**2))

# Polynomial Kernel
# A, B: data frame objects storing objects with numeric features
# c: positive real, shift
def ker_Poly(A, B, c):

    return (A.values @ B.transpose().values + c)**2

# Nearest Mean Implementation
# hyper: sigma in Gaussian kernel; c in Polynomial kernel.
def NearestMean(kernel, hyper, df_train, df_test, lab_train, lab_test):

     mu_norm = kernel(df_train.loc[lab_train==-1,:], df_train.loc[lab_train==-1,:],
         hyper)
     nu_norm = kernel(df_train.loc[lab_train==1,:], df_train.loc[lab_train==1,:], hyper)

     # predict for training data
     pred_class = np.sign(1/2 * (np.mean(mu_norm) - np.mean(nu_norm))
                   - np.mean(kernel(df_test, df_train.loc[lab_train==-1,:], hyper),
                       axis=1)
                   + np.mean(kernel(df_test, df_train.loc[lab_train==1,:], 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, 'error': misclass}

# Wrapper function to produce the plot
# kernel_dic: dictionary storing kernel name and kernel function pairs
# hyper_dic: dictionary storing hyperparameter name and hyperparameter value pairs
def Wrapper_Plot(kernel_dic, hyper_dic, df_train, df_test, lab_train, lab_test):
    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, df_train, df_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 {}'.format(ker_name))
        plt.xlabel('Hyperparameter{a}/ argmin {a}={b}'.format(a=hyper_name,
            b=round(argmin,3)))
        plt.ylabel('Error of Misclassification')
        plt.legend()
        plt.savefig('{}.png'.format(ker_name))
        plt.show()
        plt.close()

if __name__=='__main__':
    # only using Numpy and Panda for data manipulation
    # plt for plotting
    import pandas as pd
    import numpy as np
    import numpy.random as rm
    from matplotlib import pyplot as plt

    data =
        pd.read_csv('https://web.stanford.edu/~hastie/ElemStatLearn//datasets/spam.data',
        sep='\n', header=None)

    # Data Formatting
        ------------------------------------------------------------------------------------------
    df = data[0].str.split(' ', expand=True)

    # change the feature name to di; class name as 'spam'
    df.columns = np.array(['d{}'.format(i) for i in range(1,58)]+['spam'])

    # change the object type to float; integer for class 'spam'
    df[df.columns[:57]] = df[df.columns[:57]].astype('float')
    df['spam'] = df['spam'].astype('int')

    # change the nominal value in 'spam' to +/-1
    df.spam = np.array(list(map(lambda x: 1 if x==1 else -1, df.spam)))

    # standardise the features to have standard deviation 1 and mean 0
    df_features = df.iloc[:,:-1]
    df_features = (df_features-df_features.mean())/df_features.std()

    labels = df.iloc[:,-1]

    # Split into Test and Train set with random function and seed=1767
        ----------------------------------------------
    rs = rm.RandomState(seed=1767)
```

```
index = rs.rand(df.shape[0])<0.8
# the proportion of spam are roughly 0.6 for train and test

df_train = df_features.loc[index,]
df_test = df_features.loc[~index,]

lab_train = labels[index].values
lab_test = labels[~index].values

# Tuning Hyperparameter
    -----------------------------------------------------------------------------------------
kernel_dic = {'Gaussian kernel': ker_Gauss, 'Polynomial kernel': ker_Poly}
hyper_dic = {'$\sigma$': np.linspace(0.1,20,num=40)
         , 'c': np.linspace(-20,20,num=80)}

Wrapper_Plot(kernel_dic, hyper_dic, df_train, df_test, lab_train, lab_test)
```