Python codes

```python

import warnings
warnings.filterwarnings("ignore")


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from tqdm import tqdm


# ## **(d) Code a 2-class kernel nearest means classifer**


class KernelNearestMeansClassifier:
    def __init__(self, gamma=0.01, kernel_type='rbf'):
        self.gamma = gamma
        self.kernel_type = kernel_type

    def fit(self, X, y):
        self.X_train = X
        self.y_train = y


    def predict(self, X):
        X1 = self.X_train[self.y_train==0]
        X2 = self.X_train[self.y_train==1]
        if self.kernel_type == 'rbf':
            dist1 = np.zeros((X1.shape[0], X1.shape[0]))
            for i in range(X1.shape[0]):
                dist1[i, :] = np.sum((X1[i] - X1)**2, axis=1)
            self.K1 = np.mean(np.exp(-self.gamma * dist1))

            dist2 = np.zeros((X2.shape[0], X2.shape[0]))
            for i in range(X2.shape[0]):
                dist2[i, :] = np.sum((X2[i] - X2)**2, axis=1)
            self.K2 = np.mean(np.exp(-self.gamma * dist2))

            res = np.zeros((X.shape[0], 2))
            for i in tqdm(range(X.shape[0])):
                res[i, 0] = -self.K1 + 2 * np.mean(np.exp(-self.gamma * np.sum((X[i] - X1)**2, axis=1)))
                res[i, 1] = -self.K2 + 2 * np.mean(np.exp(-self.gamma * np.sum((X[i] - X2)**2, axis=1)))
        elif self.kernel_type == "linear":
            self.K1 = np.mean(X1 @ X1.T)
            self.K2 = np.mean(X2 @ X2.T)

            res = np.zeros((X.shape[0], 2))
            for i in range(X.shape[0]):
                res[i, 0] = -self.K1 + 2 * np.mean(X[i] @ X1.T)
                res[i, 1] = -self.K2 + 2 * np.mean(X[i] @ X2.T)
```

```
        else:
            raise NotImplementedError
        y_pred = np.argmax(res, axis=1)
        return y_pred

    def score(self, X, y):
        y_pred = self.predict(X)
        acc = np.mean(y_pred==y)
        return acc


train_df1 = pd.read_csv("./Pr1_dataset1/train.csv", header=None)
val_df1 = pd.read_csv("./Pr1_dataset1/val.csv", header=None)
test_df1 = pd.read_csv("./Pr1_dataset1/test.csv", header=None)
X_train1, y_train1 = train_df1.iloc[:, :-1].values, train_df1.iloc[:, -1].values
X_val1, y_val1 = val_df1.iloc[:, :-1].values, val_df1.iloc[:, -1].values
X_test1, y_test1 = test_df1.iloc[:, :-1].values, test_df1.iloc[:, -1].values
y_train1[y_train1==2] = 0
y_val1[y_val1==2] = 0
y_test1[y_test1==2] = 0

train_df2 = pd.read_csv("./Pr1_dataset2/train_2.csv", header=None)
val_df2 = pd.read_csv("./Pr1_dataset2/val_2.csv", header=None)
test_df2 = pd.read_csv("./Pr1_dataset2/test_2.csv", header=None)
X_train2, y_train2 = train_df2.iloc[:, :-1].values, train_df2.iloc[:, -1].values
X_val2, y_val2 = val_df2.iloc[:, :-1].values, val_df2.iloc[:, -1].values
X_test2, y_test2 = test_df2.iloc[:, :-1].values, test_df2.iloc[:, -1].values




# ## **(e) plot the errors**



ticks = np.linspace(-2, 2, 5)
gammas = np.array([1 / (10**(-i)) for i in ticks if i<0] + [10**i for i in ticks if i>=0])
rbf_val_errors1 = []
for gamma in gammas:
    clf = KernelNearestMeansClassifier(gamma)
    clf.fit(X_train1, y_train1)
    rbf_val_errors1.append(1 - clf.score(X_val1, y_val1))
plt.figure(figsize=(20,16))
_ = plt.xticks(ticks, list(map(lambda x:round(x,2), gammas)), rotation=45)
_ = plt.plot(ticks, rbf_val_errors1)
_ = plt.xlabel("gamma (log scale)", fontsize=18)
_ = plt.ylabel("Validation error", fontsize=18)
plt.title("Validation dataset 1", fontsize=20)
plt.savefig("./figs/1d1.png", dpi=200)
plt.show()
```

```python
ticks = np.linspace(-2, 2, 5)
gammas = np.array([1 / (10**(-i)) for i in ticks if i<0] + [10**i for i in ticks if i>=0])
rbf_val_errors2 = []
for gamma in gammas:
    clf = KernelNearestMeansClassifier(gamma)
    clf.fit(X_train2, y_train2)
    rbf_val_errors2.append(1 - clf.score(X_val2, y_val2))
plt.figure(figsize=(20,16))
_ = plt.xticks(ticks, list(map(lambda x:round(x,1), gammas)), rotation=45)
_ = plt.plot(ticks, rbf_val_errors2)
_ = plt.xlabel("gamma (log scale)", fontsize=18)
_ = plt.ylabel("Validation error", fontsize=18)
plt.title("Validation dataset 2", fontsize=20)
plt.savefig("./figs/1d2.png", dpi=200)
plt.show()


# ## **(f) compare the test error of two kernel type**


clf = KernelNearestMeansClassifier(kernel_type='linear')
clf.fit(X_train1, y_train1)
print("The test error of linear kernel on the dataset1 is {:.3f}".format(1 - clf.score(X_test1, y_test1)))

clf = KernelNearestMeansClassifier(kernel_type='linear')
clf.fit(X_train2, y_train2)
print("The test error of linear kernel on the dataset2 is {:.3f}".format(1 - clf.score(X_test2, y_test2)))

clf = KernelNearestMeansClassifier(gamma=0.1, kernel_type='rbf')
clf.fit(X_train1, y_train1)
print("The test error of rbf kernel on the dataset1 is {:.3f}".format(1 - clf.score(X_test1, y_test1)))

clf = KernelNearestMeansClassifier(gamma=100, kernel_type='rbf')
clf.fit(X_train2, y_train2)
print("The test error of rbf kernel on the dataset2 is {:.3f}".format(1 - clf.score(X_test2, y_test2)))


# ## **(g, h) plot the training data, decision region and boundary**


def plot_decision_boundary(training, label_train, clf):
    # Total number of classes
    classes = np.unique(label_train)
    nclass = len(classes)

    class_names = []
    for c in classes:
        class_names.append('Class ' + str(int(c)))

    # Set the feature range for plotting
    max_x1 = np.ceil(np.max(training[:, 0])) + 1.0
    min_x1 = np.floor(np.min(training[:, 0])) - 1.0
    max_x2 = np.ceil(np.max(training[:, 1])) + 1.0
```

```
    min_x2 = np.floor(np.min(training[:, 1])) - 1.0

    xrange = (min_x1, max_x1)
    yrange = (min_x2, max_x2)

    # step size for how finely you want to visualize the decision boundary.
    inc = 0.01

    # generate grid coordinates. This will be the basis of the decision boundary visualization.
    (x1, x2) = np.meshgrid(np.arange(xrange[0], xrange[1] + inc / 100, inc),
                           np.arange(yrange[0], yrange[1] + inc / 100, inc))

    # size of the (x1, x2) image, which will also be the size of the
    # decision boundary image that is used as the plot background.
    image_size = x1.shape
    # make (x1, x2) pairs as a bunch of row vectors.
    grid_2d = np.hstack((x1.reshape(x1.shape[0] * x1.shape[1], 1, order='F'),
                         x2.reshape(x2.shape[0] * x2.shape[1], 1, order='F')))


    pred_label = clf.predict(grid_2d)
    # reshape the idx (which contains the class label) into an image.
    decision_map = pred_label.reshape(image_size, order='F')

    # create fig
    fig, ax = plt.subplots()
    ax.imshow(decision_map, vmin=np.min(classes), vmax=9, cmap='Pastel2',
              extent=[xrange[0], xrange[1], yrange[0], yrange[1]],
              origin='lower')

    # plot the class training data.
    data_point_styles = ['rx', 'bo', 'g*']
    for i in range(nclass):
        ax.plot(training[label_train == classes[i], 0],
                training[label_train == classes[i], 1],
                data_point_styles[int(classes[i]) - 1],
                label=class_names[i])
    ax.legend()

    plt.tight_layout()
    plt.show()

    return fig


clf = KernelNearestMeansClassifier(kernel_type='linear')
clf.fit(X_train1, y_train1)
plot_decision_boundary(X_train1, y_train1, clf)


clf = KernelNearestMeansClassifier(kernel_type='linear')
clf.fit(X_train2, y_train2)
plot_decision_boundary(X_train2, y_train2, clf)
```

```python


clf = KernelNearestMeansClassifier(gamma=0.1, kernel_type='rbf')
clf.fit(X_train1, y_train1)
plot_decision_boundary(X_train1, y_train1, clf)


clf = KernelNearestMeansClassifier(gamma=100, kernel_type='rbf')
clf.fit(X_train2, y_train2)
plot_decision_boundary(X_train2, y_train2, clf)


# ## **(i) Repeat (h) using different $\gamma$**


for gamma in [0.01, 0.1, 0.3, 3, 10, 100]:
    clf = KernelNearestMeansClassifier(gamma=gamma*0.1, kernel_type='rbf')
    clf.fit(X_train1, y_train1)
    fig = plot_decision_boundary(X_train1, y_train1, clf)
    fig.savefig('./figs/1i1_r_{}.png'.format(gamma), dpi=200)


for gamma in [0.01, 0.1, 0.3, 3, 10, 100]:
    clf = KernelNearestMeansClassifier(gamma=gamma*100, kernel_type='rbf')
    clf.fit(X_train2, y_train2)
    fig = plot_decision_boundary(X_train2, y_train2, clf)
    fig.savefig('./figs/1i2_r_{}.png'.format(gamma), dpi=200)
```