Question 3 code

```
## 2 Nonlinear Mapping

import warnings
warnings.filterwarnings("ignore")


import numpy as np
import pandas as pd
import matplotlib.pyplot as plt
from sklearn.linear_model import Perceptron


def linear_decision_function(X, weight, labels):
    """
    Implements the perceptron decision function
    :param X: feature matrix of dimension NxD
    :param weight: weight vector of dimension 1xD
    :param labels: possible class assignments
    :return:
    """
    g_x = np.dot(X, np.transpose(weight))
    pred_label = np.zeros((X.shape[0], 1))
    pred_label[g_x > 0] = labels[0]
    pred_label[g_x < 0] = labels[1]
    return pred_label


def nonlinear_decision_function(X, weight, labels):
    """
    Implements a non linear decision function
    :param X: feature matrix of dimension NxD
    :param weight: weight vector of dimension 1xD
    :param labels: possible class assignments
    :return:
    """
    # TODO: define g_x
    g_x = np.dot(X, np.transpose(weight))
    pred_label = np.zeros((X.shape[0], 1))
    pred_label[g_x > 0] = labels[0]
    pred_label[g_x < 0] = labels[1]
    return pred_label


def plot_perceptron_boundary(training, label_train, weight,
                             decision_function):
    """
    Plot the 2D decision boundaries of a linear classifier
    :param training: training data
    :param label_train: class labels correspond to training data
    :param weight: weights of a trained linear classifier. This
     must be a vector of dimensions (1, D)
```

```
    :param decision_function: a function that takes in a matrix with N
     samples and returns N predicted labels
    """

    if isinstance(training, pd.DataFrame):
        training = training.to_numpy()
    if isinstance(label_train, pd.DataFrame):
        label_train = label_train.to_numpy()

    # 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 = 0.005

    # 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')))

    # Labels for each (x1, x2) pair.
    pred_label = decision_function(grid_2d, weight, classes)

    # 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()
    # show the image, give each coordinate a color according to its class
    # label
    ax.imshow(decision_map, vmin=np.min(classes), vmax=9, cmap='Pastel1',
              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



# ## (a)


## Load data
df = pd.read_csv("./h5w7_data.csv")
pos_data = df[df.label==1].loc[:, ['0', '1']].values
neg_data = df[df.label==2].loc[:, ['0', '1']].values


plt.figure(figsize=(8,6))
_ = plt.scatter(pos_data[:, 0], pos_data[:, 1], label='class=1', marker='o')
_ = plt.scatter(neg_data[:, 0], neg_data[:, 1], label='class=2', marker='v')
_ = plt.legend()
plt.show()


# ## (b)


## train the perceptron
X_train = df.loc[:, ['0', '1']].values
y_train = df.loc[:, 'label'].values

clf = Perceptron(fit_intercept=False)
clf = clf.fit(X_train, y_train)

print("The accuracy score is {:.3f}".format(clf.score(X_train, y_train)))


# ## (c)


## plot the learner decision boundaries
weights = clf.coef_[0]
fig1 = plot_perceptron_boundary(X_train, y_train, weights, linear_decision_function)


# ## (d)
```

```python


## quadratic feature space expansion
X_train_expand = np.zeros((X_train.shape[0], 5))
X_train_expand[:, :2] = X_train
X_train_expand[:, 2] = X_train[:, 0] * X_train[:, 1]
X_train_expand[:, 3:] = X_train ** 2


clf = Perceptron(fit_intercept=False)
clf = clf.fit(X_train_expand, y_train)

print("The accuracy score is {:.3f}".format(clf.score(X_train_expand,
y_train)))


```