```
"""
Author      : Jackson Crewe & Matt Guillory
Class       : HMC CS 158
Date        : 2017 Feb 14
Description : Perceptron
"""

# This code was adapted course material by Tommi Jaakola (MIT).

# utilities
from util import *

# scikit-learn libraries
from sklearn.svm import SVC

######################################################################
# functions
######################################################################

def load_simple_dataset(start=0, outlier=False) :
    """Simple dataset of three points."""

    #  dataset
    #     i    x^{(i)}      y^{(i)}
    #     1    (-1, 1)^T    1
    #     2    (0, -1)^T    -1
    #     3    (1.5, 1)^T   1
    #   if outlier is set, x^{(3)} = (12, 1)^T

    # data set
    data = Data()
    data.X = np.array([[ -1, 1],
                       [  0,-1],
                       [1.5, 1]])
    if outlier :
        data.X[2,:] = [12, 1]
    data.y = np.array([1, -1, 1])

    # circularly shift the data points
    data.X = np.roll(data.X, -start, axis=0)
    data.y = np.roll(data.y, -start)

    return data


def plot_perceptron(data, clf, plot_data=True, axes_equal=False, **kwargs) :
    """Plot decision boundary and data."""
    assert isinstance(clf, Perceptron)

    # plot options
    if "linewidths" not in kwargs :
        kwargs["linewidths"] = 2
    if "colors" not in kwargs :
        kwargs["colors"] = 'k'

    # plot data
    if plot_data : data.plot()

    # axes limits and properties
    xmin, xmax = data.X[:, 0].min() - 1, data.X[:, 0].max() + 1
    ymin, ymax = data.X[:, 1].min() - 1, data.X[:, 1].max() + 1
    if axes_equal :
        xmin = ymin = min(xmin, ymin)
        xmax = ymax = max(xmax, ymax)
        plt.xlim(xmin, xmax)
        plt.ylim(ymin, ymax)

    # create a mesh to plot in
    h = .02  # step size in the mesh
    xx, yy = np.meshgrid(np.arange(xmin, xmax, h), np.arange(ymin, ymax, h))

    # determine decision boundary
    Z = clf.predict(np.c_[xx.ravel(), yy.ravel()])
```

```
    # plot decision boundary
    Z = Z.reshape(xx.shape)
    CS = plt.contour(xx, yy, Z, [0], **kwargs)

    # legend
    if "label" in kwargs :
        #plt.clabel(CS, inline=1, fontsize=10)
        CS.collections[0].set_label(kwargs["label"])

    plt.show()


######################################################################
# classes
######################################################################

class Perceptron :

    def __init__(self) :
        """
        Perceptron classifier that keeps track of mistakes made on each data point.

        Attributes
        --------------------
            coef_     -- numpy array of shape (d,), feature weights
            mistakes_ -- numpy array of shape (n,), mistakes per data point
        """
        self.coef_ = None
        self.mistakes_ = None

    def fit(self, X, y, coef_init=None, verbose=False) :
        """
        Fit the perceptron using the input data.

        Parameters
        --------------------
            X         -- numpy array of shape (n,d), features
            y         -- numpy array of shape (n,), targets
            coef_init -- numpy array of shape (n,d), initial feature weights
            verbose   -- boolean, for debugging purposes

        Returns
        --------------------
            self      -- an instance of self
        """
        # get dimensions of data
        n,d = X.shape

        # initialize weight vector to all zeros
        if coef_init is None :
            self.coef_ = np.zeros(d)
        else :
            self.coef_ = coef_init

        # record number of mistakes we make on each data point
        self.mistakes_ = np.zeros(n)

        # debugging
        if verbose :
            print '\ttheta^{(0)} = %s' % str(self.coef_)

        ### ========== TODO : START ========== ###
        # part a: implement perceptron algorithm
        # cycle until all examples are correctly classified
        while self.fitHelper(X,y) == False:
            for i in range(n):
                if np.dot(y[i] * self.coef_, X[i]) <= 0:
                    self.mistakes_[i] += 1
                    self.coef_ = self.coef_ + y[i]*X[i]
                if verbose:
                    print(self.coef_)
        # do NOT shuffle examples on each iteration
```

```
        # on a mistake, be sure to update self.mistakes_
        #               and if verbose, output the updated self.coef_

        ### ========== TODO : END ========== ###

        return self

    def fitHelper(self, X, y):
        """
        Determine if there aren't any mistakes in the classification of the training
 data

        Parameters
        --------------------
            X         -- numpy array of shape (n,d), features
            y         -- numpy array of shape (n,), targets


        Returns
        --------------------
            bool      -- true if there are no mistakes in the training data classifi
cation, false otherwise
        """

         # get dimensions of data
        n,d = X.shape

        for i in range(n):
            if np.dot(y[i]*self.coef_,X[i]) <= 0:
                return False
        return True

    def predict(self, X) :
        """
        Predict labels using perceptron.

        Parameters
        --------------------
            X         -- numpy array of shape (n,d), features

        Returns
        --------------------
            y_pred    -- numpy array of shape (n,), predictions
        """
        return np.sign(np.dot(X, self.coef_))


######################################################################
# main
######################################################################

def main() :

    #========================================
    # test simple data set

    # starting with data point $x^{(1)}$ without outlier
    #   coef = [ 0.  1.], mistakes = 1
    # starting with data point $x^{(2)}$ without outlier
    #   coef = [ 0.5  2. ], mistakes = 2
    # starting with data point $x^{(1)}$ with outlier
    #   coef = [ 0.  1.], mistakes = 1
    # starting with data point $x^{(2)}$ with outlier
    #   coef = [ 6.  7.], mistakes = 7
    clf = Perceptron()
    for outlier in (False, True) :
        for start in (1, 2) :
            text = 'starting with data point $x^{(%d)}$ %s outlier' % \
                (start, 'with' if outlier else 'without')
            print text
            plt.figure()
            data = load_simple_dataset(start, outlier)
            clf.fit(data.X, data.y, verbose=False)
```

```
            plt.title(text)
            print '\tcoef = %s, mistakes = %d' % (str(clf.coef_), sum(clf.mistakes_)
)

    ### ========== TODO : START ========== ###
    train_data = load_data("perceptron_data.csv")
    clf2 = Perceptron()
    clf2.fit(train_data.X, train_data.y)
    print '0 initial: ', '\tcoef = %s, mistakes = %d' % (str(clf2.coef_), sum(clf2.m
istakes_))

    clf3 = Perceptron()
    clf3.fit(train_data.X, train_data.y, np.array([1,0]))
    print 'non-0 initial: ', '\tcoef = %s, mistakes = %d' % (str(clf3.coef_), sum(cl
f3.mistakes_))


    ### ========== TODO : END ========== ###



    #========================================
    # perceptron data set

    train_data = load_data("perceptron_data.csv")

    # you do not have to understand this code -- we will cover it when we discuss SV
Ms
    # compute gamma^2 using hard-margin SVM (SVM with large C)
    clf = SVC(kernel='linear', C=1e10)
    clf.fit(train_data.X, train_data.y)
    gamma = 1./np.linalg.norm(clf.coef_, 2)

    ### ========== TODO : START ========== ###
    # part c: see handout

    # compute R^2
    n,d = np.shape(train_data.X)
    R = 0
    for i in range(0, n):
        option = 0
        for j in range(d):
            option += train_data.X[i][j]**2
        option = option ** 0.5
        if option > R:
            R = option

    # compute perceptron bound (R / gamma)^2
    bound = (float(R) / float(gamma))**0.5
    print 'bound: ', bound
    ### ========== TODO : EEND ========== ###

if __name__ == "__main__" :
    main()
```