```
# Questions 7, 9
import numpy as np

target_digit = 5
lam = 1

test = np.loadtxt("features.test.txt")
train = np.loadtxt("features.train.txt")

digit = []
digit2 = []
other = []
other2 = []
for row in train:
    if row[0] == target_digit:
        digit.append([row[0], row[1], row[2]])
    elif row[0] != target_digit:
        other.append([row[0], row[1], row[2]])
for row in test:
    if row[0] == target_digit:
        digit2.append([row[0], row[1], row[2]])
    elif row[0] != target_digit:
        other2.append([row[0], row[1], row[2]])

other = np.asarray(other)
other2 = np.asarray(other2)
digit = np.c_[np.ones(len(digit)), digit]
other = other[:, [1,2]]
other = np.c_[(np.ones(len(other)) * -1), other]
other2 = other2[:, [1,2]]
other2 = np.c_[(np.ones(len(other2)) * -1), other2]
digit2 = np.c_[np.ones(len(digit2)), digit2]

digit = digit[:, [0, 2, 3]]
new_digits = np.concatenate((digit, other), axis=0)

digit2 = digit2[:, [0,2,3]]
new_digits2 = np.concatenate((digit2, other2), axis=0)

ys = new_digits[:,0]
z = new_digits[:, [1,2]]

z = np.c_[np.ones(len(z)), z]

zt = np.transpose(z)

ztz = np.dot(zt,z)
w_raw = np.linalg.inv(ztz + (lam * np.eye(len(ztz))))
weight = np.dot(w_raw, zt)
weight = np.dot(weight, ys)

newys = np.sign(np.dot(weight, np.transpose(new_digits)))
newys2 = np.sign(np.dot(weight, np.transpose(new_digits2)))
#ein = 1 - np.sum(newys) / len(newys)
ein = float(len(np.where(newys < 0)[0])) / len(newys)
eout = float(len(np.where(newys2 < 0)[0])) / len(newys2)
print "Target digit is " + str(target_digit)
print "E in " + str(ein)
print "E out " + str(eout)
# Question 8
# this version of the code uses a feature transform
import numpy as np

target_digit = 5
lam = 1
```

```
test = np.loadtxt("features.test.txt")
train = np.loadtxt("features.train.txt")

digit = []
digit2 = []
other = []
other2 = []
for row in train:
    if row[0] == target_digit:
        digit.append([row[0], row[1], row[2]])
    elif row[0] != target_digit:
        other.append([row[0], row[1], row[2]])
for row in test:
    if row[0] == target_digit:
        digit2.append([row[0], row[1], row[2]])
    elif row[0] != target_digit:
        other2.append([row[0], row[1], row[2]])

other = np.asarray(other)
other2 = np.asarray(other2)
digit = np.c_[np.ones(len(digit)), digit]
digit2 = np.c_[np.ones(len(digit2)), digit2]
other = other[:, [1,2]]
new_other = np.zeros([len(other), 6])
for i in range(len(other)):
    row = other[i]
    new_other[i] = [1, row[0], row[1], row[0]*row[1], row[0] ** 2, row[1] ** 2]
other = new_other
other = np.c_[(np.ones(len(other)) * -1), other]

new_other2 = np.zeros([len(other2), 6])
for i in range(len(other2)):
    row = other2[i]
    new_other2[i] = [1, row[0], row[1], row[0]*row[1], row[0] ** 2, row[1] ** 2]
other2 = new_other2
other2 = np.c_[(np.ones(len(other2)) * -1), other2]

digit = digit[:, [0, 2, 3]]
new_digit = np.zeros([len(digit), 6])
for i in range(len(digit)):
    row = digit[i]
    new_digit[i] = [1, row[1], row[2], row[1]*row[2], row[1] ** 2, row[2] ** 2]
digit = new_digit
digit = np.c_[np.ones(len(digit)), digit]

digit2 = digit2[:, [0, 2, 3]]
new_digit2 = np.zeros([len(digit2), 6])
for i in range(len(digit2)):
    row = digit2[i]
    new_digit2[i] = [1, row[1], row[2], row[1]*row[2], row[1] ** 2, row[2] ** 2]
digit2 = new_digit2
digit2 = np.c_[np.ones(len(digit2)), digit2]

new_digits = np.concatenate((digit, other), axis=0)
new_digits2 = np.concatenate((digit2, other2), axis=0)

ys = new_digits[:,0]
ys2 = new_digits2[:0]
z = new_digits[:, [1,2,3,4,5,6]]
z2 = new_digits2[:, [1,2,3,4,5,6]]

zt = np.transpose(z)

ztz = np.dot(zt,z)
w_raw = np.linalg.inv(ztz + (lam * np.eye(len(ztz))))
weight = np.dot(w_raw, zt)
```

```
weight = np.dot(weight, ys)

newys = np.sign(np.dot(weight, np.transpose(z)))
newys2 = np.sign(np.dot(weight, np.transpose(z2)))
ein = float(len(np.where(newys == 1)[0])) / len(newys)
eout = float(len(np.where(newys2 == 1)[0])) / len(newys2)
print "Target digit is " + str(target_digit)
print "E in " + str(ein)
print "E out " + str(eout)




# Question 10
# this version of the code uses a feature transform
import numpy as np

target_digit = 1
vs_digit = 5
lam = 1

test = np.loadtxt("features.test.txt")
train = np.loadtxt("features.train.txt")

digit = []
digit2 = []
other = []
other2 = []
for row in train:
    if row[0] == target_digit:
        digit.append([row[0], row[1], row[2]])
    elif row[0] == vs_digit:
        other.append([row[0], row[1], row[2]])
for row in test:
    if row[0] == target_digit:
        digit2.append([row[0], row[1], row[2]])
    elif row[0] == vs_digit:
        other2.append([row[0], row[1], row[2]])

other = np.asarray(other)
other2 = np.asarray(other2)
digit = np.c_[np.ones(len(digit)), digit]
digit2 = np.c_[np.ones(len(digit2)), digit2]
other = other[:, [1,2]]
new_other = np.zeros([len(other), 6])
for i in range(len(other)):
    row = other[i]
    new_other[i] = [1, row[0], row[1], row[0]*row[1], row[0] ** 2, row[1] ** 2]
other = new_other
other = np.c_[(np.ones(len(other)) * -1), other]

new_other2 = np.zeros([len(other2), 6])
for i in range(len(other2)):
    row = other2[i]
    new_other2[i] = [1, row[0], row[1], row[0]*row[1], row[0] ** 2, row[1] ** 2]
other2 = new_other2
other2 = np.c_[(np.ones(len(other2)) * -1), other2]

digit = digit[:, [0, 2, 3]]
new_digit = np.zeros([len(digit), 6])
for i in range(len(digit)):
    row = digit[i]
    new_digit[i] = [1, row[1], row[2], row[1]*row[2], row[1] ** 2, row[2] ** 2]
digit = new_digit
digit = np.c_[np.ones(len(digit)), digit]
```

```
digit2 = digit2[:, [0, 2, 3]]
new_digit2 = np.zeros([len(digit2), 6])
for i in range(len(digit2)):
    row = digit2[i]
    new_digit2[i] = [1, row[1], row[2], row[1]*row[2], row[1] ** 2, row[2] ** 2]
digit2 = new_digit2
digit2 = np.c_[np.ones(len(digit2)), digit2]

new_digits = np.concatenate((digit, other), axis=0)
new_digits2 = np.concatenate((digit2, other2), axis=0)

ys = new_digits[:,0]
ys2 = new_digits2[:0]
z = new_digits[:, [1,2,3,4,5,6]]
z2 = new_digits2[:, [1,2,3,4,5,6]]

zt = np.transpose(z)

ztz = np.dot(zt,z)
w_raw = np.linalg.inv(ztz + (lam * np.eye(len(ztz))))
weight = np.dot(w_raw, zt)
weight = np.dot(weight, ys)

newys = np.sign(np.dot(weight, np.transpose(z)))
newys2 = np.sign(np.dot(weight, np.transpose(z2)))
ein = float(len(np.where(newys == 1)[0])) / len(newys)
eout = float(len(np.where(newys2 == 1)[0])) / len(newys2)
print "Target digit is " + str(target_digit)
print "E in " + str(ein)
print "E out " + str(eout)
```

```
# question 12
from sklearn import svm
import numpy as np

points = np.zeros((7,3))
points[0] = [-1, 1, 0]
points[1] = [-1, 0, 1]
points[2] = [-1, 0, -1]
points[3] = [1, -1, 0]
points[4] = [1, 0, 2]
points[5] = [1, 0, -2]
points[6] = [1, -2, 0]

classifications = points[:, [0]]
data = points[:, [1,2]]

clf = svm.SVC(kernel = 'poly', degree = 2, coef0 = 1.0, gamma = 1)
clf.fit(data, classifications)

print len(clf.support_vectors_)
```

```
# Question 13
from sklearn import svm
import numpy as np
```

```
points = 100
runs = 100
counter = 0.0
gam = 1.5

for j in range(runs):
    xs = np.random.uniform(-1,1, (points, 2))
    classifications = np.zeros(points)
    for i in range(points):
        row = xs[i]
        classifications[i] = int(np.sign(row[1] - row[0] +
                                         0.25*np.sin(np.pi * row[0])))
    clf = svm.SVC(kernel = 'rbf', gamma = gam)

    clf.fit(xs, classifications)
    y_pred = clf.predict(xs)
    comparisons = np.equal(y_pred, classifications)
    ein = 1 - float(np.sum(comparisons)) / len(comparisons)
    if ein == 0:
        counter += 1
print counter / runs
```

```
# Question 14-18
from sklearn import svm
import numpy as np
from scipy.spatial import distance

svm_wins = 0
lloyd_wins = 0
points = 100
runs = 100
counter_svm = 0.0
gam = 1.5
k = 9
def same(points1, points2):
    return np.array_equal(points1, points2)

def find_closest_center_index(target_point, center_points):
    best_dist = 10000
    best_center = 0
    center_points = center_points[:, [1,2]]
    for i in range(len(center_points)):
        dist = distance.euclidean(target_point, center_points[i])
        if dist < best_dist:
            best_dist = dist
            best_center = i
    return best_center

def adjust_centers(training_points, center_points):
    for i in range(len(training_points)):
        new_center =find_closest_center_index(training_points[i], center_points)
        training_points[i][0] = new_center
    return training_points

for j in range(runs):
    centers = np.random.uniform(-1,1, (k, 2))
    xs = np.random.uniform(-1,1, (points, 2))
    xs2 = xs
    xs2 = np.c_[np.zeros(len(xs2)), xs2]
```

```
    classifications = np.zeros(points)
    for i in range(points):
        row = xs[i]
        classifications[i] = int(np.sign(row[1] - row[0] +
                                         0.25*np.sin(np.pi * row[0])))
    clf = svm.SVC(kernel = 'rbf', gamma = gam)

    clf.fit(xs, classifications)
    y_pred = clf.predict(xs)
    comparisons = np.equal(y_pred, classifications)
    ein_svm = 1 - float(np.sum(comparisons)) / len(comparisons)
    if ein_svm == 0:
        counter += 1

    xs2_new = adjust_centers(xs2, centers)
    while (not same(xs2, xs2_new)):
        xs2 = xs2_new
        xs2_new = adjust_centers(xs2_new, centers)
    sum = 0.0
    I = np.zeros((points, k))
    for i in range(points):
        for j in range(k):
            I[i][j] = np.exp(-gam * np.power(np.norm(xs2[k*i + j]) -
                                             xs2[k*i + j][0]), 2)
    I = np.linalg.pinv(I)
    weight = np.dot(I, comparisons)
    signs_for_lloyd = 0.0
    for i in range(9):
        for j in range(points):
            signs_for_lloyd += np.sign(weight[i] * np.exp(-gam *
                               np.power(np.norm(xs2[j, [1,2]] - xs2[j][0]), 2)))
        weight[i] = signs_for_lloyd
        signs_for_lloyd = 0
    test_points = np.random.uniform(-1,1, (points, 2))
    test_out_classes = clf.predict(test_points)
    test_out_classes2 = np.sign(np.dot(weight, test_points))

    svm_out = 1 - float(np.sum(test_out_classes)) / len(test_out_classes)
    lloyd_out = 1 - float(np.sum(test_out_classes2)) / len(test_out_classes2)
    if svm_out < lloyd_out:
        svm_wins += 1
    else:
        lloyd_wins += 1
print "SVM won: " + str(svm_wins > lloyd_wins)
```