```
import math
import numpy as np
from sklearn.preprocessing import StandardScaler
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import StratifiedKFold
from sklearn.metrics import accuracy_score
import matplotlib.pyplot as plt

fh_xtr = np.genfromtxt('/Users/huangweiding/Documents/leetcode/EE660/H2W3 data files/Xtrain.csv',
delimiter=',')
fh_ytr = np.genfromtxt('/Users/huangweiding/Documents/leetcode/EE660/H2W3 data files/ytrain.csv',
delimiter=',')
fh_xtest = np.genfromtxt('/Users/huangweiding/Documents/leetcode/EE660/H2W3 data files/Xtest.csv',
delimiter=',')
fh_ytest = np.genfromtxt('/Users/huangweiding/Documents/leetcode/EE660/H2W3 data files/ytest.csv',
delimiter=',')
#fh_xtr = np.asarray(fh_xtr)
#fh_ytr = np.asarray(fh_ytr)

# standardize
scaler = StandardScaler()
scaler.fit(fh_xtr)
xtr_std = scaler.transform(fh_xtr)
st_mean = scaler.mean_
st_std = scaler.var_
xtest_std = np.empty([len(fh_xtest), len(fh_xtest[0])])
for i in range(len(fh_xtest)):
    for j in range(len(fh_xtest[0])):
        xtest_std[i][j] = (fh_xtest[i][j] - st_mean[j]) / st_std[j]



# log(xij+ 0.1)
xtr_log = np.empty([len(fh_xtr), len(fh_xtr[0])])
```

```
for i in range(len(fh_xtr)):
    for j in range(len(fh_xtr[0])):
        xtr_log[i][j] = math.log10(fh_xtr[i][j] + 0.1)
xtest_log = np.empty([len(fh_xtest), len(fh_xtest[0])])
for i in range(len(fh_xtest)):
    for j in range(len(fh_xtest[0])):
        xtest_log[i][j] = math.log10(fh_xtest[i][j] + 0.1)



# binarization
xtr_binary = np.empty([len(fh_xtr), len(fh_xtr[0])])
for i in range(len(fh_xtr)):
    for j in range(len(fh_xtr[0])):
        if fh_xtr[i][j] > 0:
            xtr_binary[i][j] = 1
        else:
            xtr_binary[i][j] = 0
xtest_binary= np.empty([len(fh_xtest), len(fh_xtest[0])])
for i in range(len(fh_xtest)):
    for j in range(len(fh_xtest[0])):
        if fh_xtest[i][j] > 0:
            xtest_binary[i][j] = 1
        else:
            xtest_binary[i][j] = 0

# cross-validation and logistic regression fitment
skf = StratifiedKFold(n_splits=5)
lam = np.linspace(0.01, 1, 50)
ave_error_lst = list()
for l in lam:
    sum_error = 0
    for train_index, test_index in skf.split(fh_xtr, fh_ytr):
        X_train, X_test = xtr_std[train_index], xtr_std[test_index]
```

```
        Y_train, Y_test = fh_ytr[train_index], fh_ytr[test_index]
        max_acc = 0
        clf = LogisticRegression(C=1/l, max_iter=1000)
        clf.fit(X_train, Y_train)
        y_pred = clf.predict(X_test)
        cur_acc = accuracy_score(Y_test, y_pred)
        error = 1 - cur_acc
        sum_error += error
    ave_error = sum_error/5
    ave_error_lst.append(ave_error)
print('std: minimum average error rate is {}, corresponding lambda is {}'. format(min(ave_error_lst),
lam[ave_error_lst.index(min(ave_error_lst))]))

clf = LogisticRegression(C=lam[ave_error_lst.index(min(ave_error_lst))], max_iter=1000)
clf.fit(xtr_std, fh_ytr)
y_pred = clf.predict(xtr_std)
acc = accuracy_score(y_pred, fh_ytr)
print('std: error rate for entire training set: {}'.format(1 - acc))
# standardize the test set using the mu and std from training set
y_test_pred = clf.predict(xtest_std)
acc = accuracy_score(y_test_pred, fh_ytest)
print('std: error rate for test set: {}'.format(1 - acc))

# log transformation logistic regression
skf = StratifiedKFold(n_splits=5)
lam = np.linspace(0.01, 1, 50)
ave_error_lst = list()
for l in lam:
    sum_error = 0
    for train_index, test_index in skf.split(fh_xtr, fh_ytr):
        X_train, X_test = xtr_log[train_index], xtr_log[test_index]
        Y_train, Y_test = fh_ytr[train_index], fh_ytr[test_index]
        max_acc = 0
```

```
        clf = LogisticRegression(C=1/l, max_iter=1000)
        clf.fit(X_train, Y_train)
        y_pred = clf.predict(X_test)
        cur_acc = accuracy_score(Y_test, y_pred)
        error = 1 - cur_acc
        sum_error += error
    ave_error = sum_error/5
    ave_error_lst.append(ave_error)
print('log: minimum average error rate is {}, corresponding lambda is {}'. format(min(ave_error_lst),
lam[ave_error_lst.index(min(ave_error_lst))]))
clf = LogisticRegression(C=lam[ave_error_lst.index(min(ave_error_lst))], max_iter=1000)
clf.fit(xtr_log, fh_ytr)
y_pred = clf.predict(xtr_log)
acc = accuracy_score(y_pred, fh_ytr)
print('log: error rate for entire training set: {}'.format(1 - acc))
y_test_pred = clf.predict(xtest_log)
acc = accuracy_score(y_test_pred, fh_ytest)
print('log: error rate for test set: {}'.format(1 - acc))

# binary logistic regression
skf = StratifiedKFold(n_splits=5)
lam = np.linspace(0.01, 1, 50)
ave_error_lst = list()
for l in lam:
    sum_error = 0
    for train_index, test_index in skf.split(fh_xtr, fh_ytr):
        X_train, X_test = xtr_binary[train_index], xtr_binary[test_index]
        Y_train, Y_test = fh_ytr[train_index], fh_ytr[test_index]
        max_acc = 0
        clf = LogisticRegression(C=1/l, max_iter=1000)
        clf.fit(X_train, Y_train)
        y_pred = clf.predict(X_test)
        cur_acc = accuracy_score(Y_test, y_pred)
```

```
        error = 1 - cur_acc
        sum_error += error
    ave_error = sum_error/5
    ave_error_lst.append(ave_error)
print('binary: minimum average error rate is {}, corresponding lambda is {}'. format(min(ave_error_lst),
lam[ave_error_lst.index(min(ave_error_lst))]))
clf = LogisticRegression(C=lam[ave_error_lst.index(min(ave_error_lst))], max_iter=1000)
clf.fit(xtr_binary, fh_ytr)
y_pred = clf.predict(xtr_binary)
acc = accuracy_score(y_pred, fh_ytr)
print('binary: error rate for entire training set: {}'.format(1 - acc))
y_test_pred = clf.predict(xtest_binary)
acc = accuracy_score(y_test_pred, fh_ytest)
print('binary: error rate for test set: {}'.format(1 - acc))

# additional questions
words = np.empty([len(fh_xtest), 1])
char = np.empty([len(fh_xtest), 1])
for i in range(len(fh_xtest)):
    for j in range(48):
        words[i] += xtest_binary[i][j]
for i in range(len(fh_xtest)):
    for j in range(48, 55):
        char[i] += xtest_binary[i][j]

plt.figure()
for i in range(len(fh_xtest)):
    if fh_ytest[i] == 1:
        plt.scatter(words[i], char[i], s=50, marker='o', color='blue')
    else:
        plt.scatter(words[i], char[i], marker='.', color='red')
plt.show()
```

```
# 3d plot
spamx = np.array([])
spamy = np.array([])
for i in range(len(fh_xtest)):
    if fh_ytest[i] == 1:
        spamx = np.append(spamx, words[i])
        spamy = np.append(spamy, char[i])
#spam_x = np.asarray(spam_x)
#spam_y = np.asarray(spam_y)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
hist, xedges, yedges = np.histogram2d(spamx, spamy)
xpos, ypos = np.meshgrid(xedges[:-1] + 0.25, yedges[:-1] + 0.25, indexing='ij')
xpos = xpos.ravel()
ypos = ypos.ravel()
zpos = 0
dx = dy = 0.5 * np.ones_like(zpos)
dz = hist.ravel()
ax.bar3d(xpos, ypos, zpos, dx, dy, dz, zsort='average')
plt.show()

nonspamx = np.array([])
nonspamy = np.array([])
for i in range(len(fh_xtest)):
    if fh_ytest[i] != 1:
        nonspamx = np.append(nonspamx, words[i])
        nonspamy = np.append(nonspamy, char[i])
#spam_x = np.asarray(spam_x)
#spam_y = np.asarray(spam_y)
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')
hist, xedges, yedges = np.histogram2d(nonspamx, nonspamy)
xpos, ypos = np.meshgrid(xedges[:-1] + 0.25, yedges[:-1] + 0.25, indexing='ij')
```

```
xpos = xpos.ravel()
ypos = ypos.ravel()
zpos = 0
dx = dy = 0.5 * np.ones_like(zpos)
dz = hist.ravel()
ax.bar3d(xpos, ypos, zpos, dx, dy, dz, zsort='average')
plt.show()
```