Problem1:
lambda = 1.0: accuracy for **validation** set is **0.90028199436**
lambda = 1.0: accuracy for **test** set is **0.577788444231**

Problem2:
lambda = 1.0: accuracy for **validation** set is **0.945161096778**
lambda = 1.0: accuracy for **test** set is **0.361792764145**

Problem3:
number of **true positive** on test is **5843**
number of **true negative** on test is **187**
number of **false positive** on test is **10568**
number of **false negative** on test is **69**
Balanced Error Rate for **test: 0.4971419577639251**

Problem4:
For the new classifier, I use the weighted log-likelihood.
Balanced Error Rate for **train: 0.4432375407886931**
Balanced Error Rate for **valid: 0.4199156034378244**
Balanced Error Rate for **test: 0.4445368110876459**

Problem5:
I got the same best performance on lambda = 0,0.01,0.1
lambda = 0: accuracy for **train** set is **0.53726149046**
lambda = 0: accuracy for **validation** set is **0.412971740565**
lambda = 0: accuracy for **test** set is **0.59998800024**

lambda = 0.01: accuracy for train set is 0.53726149046
lambda = 0.01: accuracy for validation set is 0.412971740565
lambda = 0.01: accuracy for test set is 0.59998800024

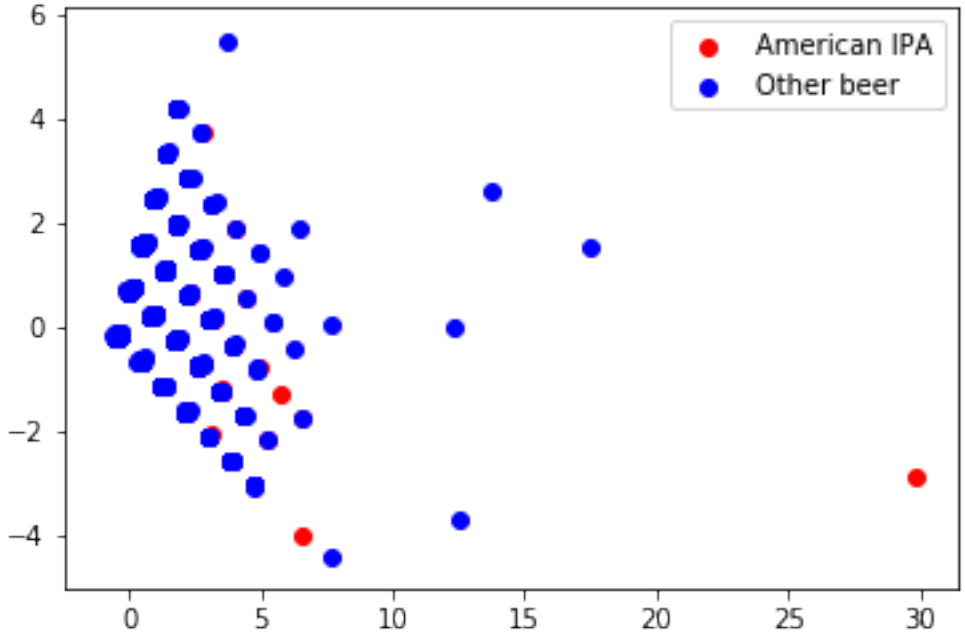
lambda = 0.1: accuracy for train set is 0.53726149046
lambda = 0.1: accuracy for validation set is 0.412971740565
lambda = 0.1: accuracy for test set is 0.59998800024

Problem6
```
[[ -7.61829498e-04   9.67903522e-04  -1.20659582e-02   1.22129184e-02
    4.72976524e-01  -3.22321014e-05   8.78492262e-01   5.93469606e-02
   -1.21867430e-04   2.69483759e-02]
 [  1.55627315e-03   8.69025779e-03   7.62598978e-03  -5.57589095e-03
    8.79624170e-01  -5.58225425e-04  -4.74983274e-01   2.17431280e-02
    8.70381547e-04   3.34474951e-03]]
```

Problem7
I calculate the reconstructed data x_project, subtract x_project from x_training, and sum all the (x_training-x_project)^2, and then divide the number of whole data points(which are 10 dimensional).By using this method,

The reconstruction error is **0.33108369081315765**

Problem8



code:
part I

```
import numpy
from urllib.request import urlopen
import scipy.optimize
import random
from math import exp
from math import log
def parseData(fname):
    for l in urlopen(fname):
        yield eval(l)

print("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.j
son"))
print("done")

def inner(x,y):
    return sum([x[i]*y[i] for i in range(len(x))])

def sigmoid(x):
    return 1.0 / (1 + exp(-x))
def feature(datum):
    feat = [1, datum['review/taste'], datum['review/appearance'], datum['revi
ew/aroma'], datum['review/palate'], datum['review/overall']]
    return feat
def feature(datum):
```

```
    datum_text=datum['review/text'].lower().split()
    feat = [1, datum_text.count("lactic"),datum_text.count("tart"),datum_tex
t.count("sour"),datum_text.count("citric"),datum_text.count("sweet"),datum_te
xt.count("acid"),datum_text.count("hop"),datum_text.count("fruit"),datum_tex
t.count("salt"),datum_text.count("spicy")]
    return feat
X = [feature(d) for d in data]
y = [d['beer/ABV'] >= 6.5 for d in data]
X_train = X[:int(len(X)/3)]
X_valid = X[int(len(X)/3):int(2*len(X)/3)]
X_test = X[int(2*len(X)/3):]
y_train = y[:int(len(y)/3)]
y_valid = y[int(len(y)/3):int(2*len(y)/3)]
y_test = y[int(2*len(y)/3):]
N=len(y_train)
coeff_1=N/(2*sum(y_train))
coeff_0=N/(2*(N-sum(y_train)))
print(coeff_1)
print(coeff_0)
##################################################
# Logistic regression by gradient ascent         #
##################################################

# NEGATIVE Log-likelihood
def f(theta, X, y, lam):
    loglikelihood = 0
    for i in range(len(X)):
        logit = inner(X[i], theta)
        if y[i]:
            loglikelihood -= log(1 + exp(-logit))*coeff_1
        else:
            loglikelihood -= (logit+log(1+exp(-logit)))*coeff_0
    for k in range(len(theta)):
        loglikelihood -= lam * theta[k]*theta[k]
  # for debugging
  # print("ll =" + str(loglikelihood))
    return -loglikelihood

# NEGATIVE Derivative of log-likelihood
def fprime(theta, X, y, lam):
    dl = [0]*len(theta)
    for i in range(len(X)):
        logit = inner(X[i], theta)
        for k in range(len(theta)):
            if y[i]:
                dl[k] += X[i][k] * (1 - sigmoid(logit))*coeff_1
            if not y[i]:
                dl[k] += X[i][k]*(-coeff_0)+X[i][k] * (1 - sigmoid(logit))*co
eff_0
    for k in range(len(theta)):
        dl[k] -= lam*2*theta[k]
    return numpy.array([-x for x in dl])
##################################################
# Train                                          #
##################################################
def train(lam):
```

```
    theta,_,_ = scipy.optimize.fmin_l_bfgs_b(f, [0]*len(X_train[0]), fprime,
pgtol = 10, args = (X_train, y_train, lam))
    return theta
##################################################
# Predict                                        #
##################################################

def performance_valid(theta):
    scores_valid = [inner(theta,x) for x in X_valid]
    predictions_valid = [s > 0 for s in scores_valid]
    correct_valid = [(a==b) for (a,b) in zip(predictions_valid,y_valid)]
    acc_valid = sum(correct_valid) * 1.0 / len(correct_valid)
    return acc_valid
def performance_test(theta):
    scores_test = [inner(theta,x) for x in X_test]
    predictions_test = [s > 0 for s in scores_test]
    correct_test = [(a==b) for (a,b) in zip(predictions_test,y_test)]
    acc_test = sum(correct_test) * 1.0 / len(correct_test)
    return acc_test
def performance_train(theta):
    scores_train = [inner(theta,x) for x in X_train]
    predictions_train = [s > 0 for s in scores_train]
    correct_train = [(a==b) for (a,b) in zip(predictions_train,y_train)]
    acc_train = sum(correct_train) * 1.0 / len(correct_train)
    return acc_train
def evaluate_classifier_test(theta):
    scores_test = [inner(theta,x) for x in X_test]
    predictions_test = [s > 0 for s in scores_test]
    true_positive = [1 if a==1 and b==1 else 0 for (a,b) in zip(predictions_t
est,y_test)]
    true_negative = [1 if a==0 and b==0 else 0 for (a,b) in zip(predictions_t
est,y_test)]
    false_positive = [1 if a==1 and b==0 else 0 for (a,b) in zip(predictions_
test,y_test)]
    false_negative = [1 if a==0 and b==1 else 0 for (a,b) in zip(predictions_
test,y_test)]
    TP=sum(true_positive)
    TN=sum(true_negative)
    FP=sum(false_positive)
    FN=sum(false_negative)
    FPR=FP/(FP+TN)
    FNR=FN/(FN+TP)
    BER=0.5*(FPR+FNR)
    print("number of true positive on test is ",TP)
    print("number of true negative on test is ",TN)
    print("number of false positive on test is ",FP)
    print("number of false negative on test is ",FN)
    print("Balanced Error Rate for test: ",BER)
    # also can calculate the length of ttpp = [ 1 for (a,b) in zip(prediction
s_test,y_test) if a==1 and b==1]
def evaluate_classifier_train(theta):
    scores_train = [inner(theta,x) for x in X_train]
    predictions_train = [s > 0 for s in scores_train]
    true_positive = [1 if a==1 and b==1 else 0 for (a,b) in zip(predictions_t
rain,y_train)]
    true_negative = [1 if a==0 and b==0 else 0 for (a,b) in zip(predictions_t
rain,y_train)]
```

```
    false_positive = [1 if a==1 and b==0 else 0 for (a,b) in zip(predictions_
train,y_train)]
    false_negative = [1 if a==0 and b==1 else 0 for (a,b) in zip(predictions_
train,y_train)]
    TP=sum(true_positive)
    TN=sum(true_negative)
    FP=sum(false_positive)
    FN=sum(false_negative)
    FPR=FP/(FP+TN)
    FNR=FN/(FN+TP)
    BER=0.5*(FPR+FNR)
    #print("number of true positive on test is ",TP)
    #print("number of true negative on test is ",TN)
    #print("number of false positive on test is ",FP)
    #print("number of false negative on test is ",FN)
    print("Balanced Error Rate for train: ",BER)
def evaluate_classifier_valid(theta):
    scores_valid = [inner(theta,x) for x in X_valid]
    predictions_valid = [s > 0 for s in scores_valid]
    true_positive = [1 if a==1 and b==1 else 0 for (a,b) in zip(predictions_v
alid,y_valid)]
    true_negative = [1 if a==0 and b==0 else 0 for (a,b) in zip(predictions_v
alid,y_valid)]
    false_positive = [1 if a==1 and b==0 else 0 for (a,b) in zip(predictions_
valid,y_valid)]
    false_negative = [1 if a==0 and b==1 else 0 for (a,b) in zip(predictions_
valid,y_valid)]
    TP=sum(true_positive)
    TN=sum(true_negative)
    FP=sum(false_positive)
    FN=sum(false_negative)
    FPR=FP/(FP+TN)
    FNR=FN/(FN+TP)
    BER=0.5*(FPR+FNR)
    #print("number of true positive on test is ",TP)
    #print("number of true negative on test is ",TN)
    #print("number of false positive on test is ",FP)
    #print("number of false negative on test is ",FN)
    print("Balanced Error Rate for valid: ",BER)
##################################################
# Validation pipeline                            #
##################################################
lam = 1.0
theta = train(lam)
acc_valid = performance_valid(theta)
acc_test = performance_test(theta)
print("lambda = " + str(lam) + ":\taccuracy for validation set is\t" + str(ac
c_valid))
print("lambda = " + str(lam) + ":\taccuracy for test set is\t" + str(acc_tes
t))

evaluate_classifier_test(theta)
evaluate_classifier_train(theta)
evaluate_classifier_valid(theta)

#(5)
lam = [0,0.01,0.1]
```

```
for i in lam:
    theta = train(i)
    acc_train = performance_train(theta)
    acc_valid = performance_valid(theta)
    acc_test = performance_test(theta)
    print("lambda = " + str(i) + ":\taccuracy for train set is\t" + str(acc_t
rain))
    print("lambda = " + str(i) + ":\taccuracy for validation set is\t" + str
(acc_valid))
    print("lambda = " + str(i) + ":\taccuracy for test set is\t" + str(acc_te
st))
scores_test = [inner(theta,x) for x in X_test]
    predictions_test = [s > 0 for s in scores_test]
    true_positive = [1 if a==1 and b==1 else 0 for (a,b) in zip(predictions_t
est,y_test)]
    true_negative = [1 if a==0 and b==0 else 0 for (a,b) in zip(predictions_t
est,y_test)]
    false_positive = [1 if a==1 and b==0 else 0 for (a,b) in zip(predictions_
test,y_test)]
    false_negative = [1 if a==0 and b==1 else 0 for (a,b) in zip(predictions_
test,y_test)]
    TP=sum(true_positive)
    TN=sum(true_negative)
    FP=sum(false_positive)
    FN=sum(false_negative)
    FPR=FP/(FP+TN)
    FNR=FN/(FN+TP)
    BER=0.5*(FPR+FNR)
    print("number of true positive on test is ",TP)
    print("number of true negative on test is ",TN)
    print("number of false positive on test is ",FP)
    print("number of false positive on test is ",FN)
    print("Balanced Error Rate: ",BER)
    ttpp = [ 1 for (a,b) in zip(predictions_test,y_test) if a==1 and b==1]
    print(len(ttpp))

part 2
import numpy
import urllib.request
import scipy.optimize
import random
from sklearn.decomposition import PCA
from collections import defaultdict
### PCA on beer reviews ###
def parseData(fname):
      for l in urllib.request.urlopen(fname):
            yield eval(l)

print ("Reading data...")
data = list(parseData("http://jmcauley.ucsd.edu/cse190/data/beer/beer_50000.j
son"))
print ("done")

def feature(datum):
    datum_text=datum['review/text'].lower().split()
    feat = [datum_text.count("lactic"),datum_text.count("tart"),datum_text.co
unt("sour"),datum_text.count("citric"),datum_text.count("sweet"),datum_text.c
```

```
ount("acid"),datum_text.count("hop"),datum_text.count("fruit"),datum_text.cou
nt("salt"),datum_text.count("spicy")]
    return feat
X = [feature(d) for d in data]
X_train = X[:int(len(X)/3)]
pca = PCA(n_components=2)
pca.fit(X_train)
#print (pca.components_)
#after using pca.fit_transform, we get loadings for each samples.
#meaning how much of each component you need to describe it best using a line
ar combination of the components_ (the principal axes in feature space).
X_train_pca=pca.fit_transform(X_train)
X_projected = pca.inverse_transform(X_train_pca)
loss = np.mean((X_train - X_projected)** 2)*10

loss
judge=[1 if d['beer/style'] == 'American IPA' else 0 for d in data ]
judge_ipa=judge[:int(len(y)/3)]
import matplotlib.pyplot as plt
import numpy as np

data_IPA=[X_train_pca[i] for i in range(len(X_train)) if judge_ipa[i]==1]
data_else=[X_train_pca[i] for i in range(len(X_train)) if judge_ipa[i]==0]
for i in range(len(data_IPA)):
    data_IPA[i]=data_IPA[i].tolist()
data_IPA=np.array(data_IPA)
for i in range(len(data_else)):
    data_else[i]=data_else[i].tolist()
data_else=np.array(data_else)
x_ipa=data_IPA[:,0]
y_ipa=data_IPA[:,1]
x_else=data_else[:,0]
y_else=data_else[:,1]
ipa=plt.scatter(x_ipa,y_ipa,color = 'red')
other=plt.scatter(x_else,y_else,color = 'blue')
plt.legend((ipa,other),('American IPA','Other beer'))
plt.show()
```