1. **lambda** = 1.0: **train accuracy**=0.669946601068
   lambda = 1.0: **validation accuracy**=0.90076198476
   lambda = 1.0: **test accuracy**=0.577163086523

   Code stub:

```
lam = 1.0

theta = train(lam)
acc_train = performance(theta, X_train, y_train)
print("lambda = " + str(lam) + ":\ttrain accuracy=" + str(acc_train))
acc_validation = performance(theta, X_validation, y_validation)
print("lambda = " + str(lam) + ":\tvalidation accuracy=" +
str(acc_validation))
acc_test = performance(theta, X_test, y_test)
print("lambda = " + str(lam) + ":\ttest accuracy=" + str(acc_test))
```

2. **lambda** = 1.0: **train accuracy**=0.562608747825
   lambda = 1.0: **validation accuracy**=0.946301073979
   lambda = 1.0: **test accuracy**=0.36229449178

   Code stub:

```
def feature1(datum):
rev = datum['review/text'].lower()
rev = re.sub('['+string.punctuation+']', '', rev)
w = rev.split()
c = Counter(w)
feat = [1, c['lactic'], c['tart'], c['sour'], c['citric'], c['sweet'],
c['acid'], c['hop'], c['fruit'], c['salt'], c['spicy']]
return feat
```

3. **Balanced Error Rate**: 0.497105515479
   True +ve: 5832
   True -ve: 206
   False +ve: 10549
   False -ve: 79

   Code stub:

```
def performance_detailed(theta, X, Y):
        scores = [inner(theta,x) for x in X]
        predictions = [s > 0 for s in scores]
        TP = sum([(a==b) and a==True for (a,b) in zip(predictions,Y)])
        TN = sum([(a==b) and a==False for (a,b) in zip(predictions, Y)])
        FP = sum([(a!=b) and a==True for (a,b) in zip(predictions, Y)])
        FN = sum([(a!=b) and a==False for (a,b) in zip(predictions, Y)])
        TPR = TP/(TP+FN)
        TNR = TN/(TN+FP)
        BER = 1 - 0.5*(TPR+TNR)
        print("Balanced Error Rate: " + str(BER))
        print("True +ve: " + str(TP))
        print("True -ve: " + str(TN))
        print("False +ve: " + str(FP))
        print("False -ve: " + str(FN))
```

4. **Training: Balanced Error Rate**: 0.43426256748
   Validation: Balanced Error Rate: 0.410820006797
   Test: Balanced Error Rate: 0.443418510478

   Code stub:

```
# NEGATIVE Log-likelihood for class imbalance
def fc(theta, X, y, lam):
  total = len(y)
  y1count = sum(y)
  y0count = total - sum(y)
  y1balance = total/(2*y1count)
  y0balance = total/(2*y0count)
  loglikelihood = 0
  for i in range(len(X)):
    logit = inner(X[i], theta)
    if(y[i]):
        loglikelihood -= (y1balance*log(1+exp(-logit)))
    elif not y[i]:
        loglikelihood -= (y0balance*(logit + log(1+exp(-logit))))
  for k in range(len(theta)):
    loglikelihood -= lam * theta[k]*theta[k]
  return -loglikelihood

# NEGATIVE Derivative of log-likelihood for class imbalance
def fcprime(theta, X, y, lam):
  total = len(y)
  y1count = sum(y)
  y0count = total - sum(y)
  y1balance = total/(2*y1count)
  y0balance = total/(2*y0count)
  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] += (y1balance*(X[i][k] * (1-sigmoid(logit))))
      elif not y[i]:
        dl[k] -= (y0balance*(X[i][k]*sigmoid(logit)))
  for k in range(len(theta)):
    dl[k] -= lam*2*theta[k]
  return numpy.array([-x for x in dl])


def trainc(lam):
  theta,_,_ = scipy.optimize.fmin_l_bfgs_b(fc, [0]*len(X[0]), fcprime, pgtol = 10,
args = (X_train, y_train, lam))
  return theta
```

5.

Lambda	Train Accuracy	Validation Accuracy
0	0.554088918222	0.470630587388
0.01	0.554088918222	0.470630587388
0.1	0.554148917022	0.471170576588
1	0.554148917022	0.471170576588
100	0.554208915822	0.471590568189

Best value of lambda: 100
Train accuracy: 0. 554208915822 **Train BER:** 0.434268457917
Validation accuracy: 0.471590568189 **Validation BER:** 0.410600005539
Test accuracy: 0.471590568189 **Test BER:** 0.443490176342

6. **n_components = 5;**
   Transform matrix:

```
[[ -5.99354946e-04   3.95164941e-03  -9.30338083e-03   9.76942328e-03
   7.99767032e-01  -1.16030365e-04   5.94903989e-01   7.26399102e-02
   1.73572637e-04   3.14224279e-02]
 [ -1.57672227e-03  -8.65225859e-03  -1.41693175e-02   1.36630501e-02
  -5.96926193e-01   2.48730319e-04   8.01927918e-01  -3.23901167e-03
  -1.24896291e-03   1.06688940e-02]
 [  4.00346584e-03   4.43677073e-02   9.06390127e-02   3.63629924e-03
  -6.09998226e-02  -2.42367581e-04  -3.98349064e-02   9.91604464e-01
   3.81694276e-04   3.46324860e-02]
 [ -4.26042897e-04   2.28664104e-02  -1.25209784e-02   1.95017088e-02
  -1.68324282e-02  -1.59490471e-04  -2.62633047e-02  -3.69062502e-02
   2.69024737e-03   9.98297264e-01]
 [  2.60663010e-02   2.24900691e-01   9.68807416e-01   3.42586587e-03
   3.00551477e-03   9.49385575e-03   2.13252265e-02  -9.78297807e-02
   7.56473452e-04   3.93835606e-03]]
```

   Code stub:

```
def feature1(datum):
rev = datum['review/text'].lower()
rev = re.sub('['+string.punctuation+']', '', rev)
w = rev.split()
c = Counter(w)
feat = [c['lactic'], c['tart'], c['sour'], c['citric'], c['sweet'], c['acid'],
c['hop'], c['fruit'], c['salt'], c['spicy']]
return feat

pca = PCA(n_components=5)
pca.fit(X_train)
print(pca.components_)
```

7. **n_components=2**
   Mean squared reconstruction error: 0.474986515177
   Transform Matrix:

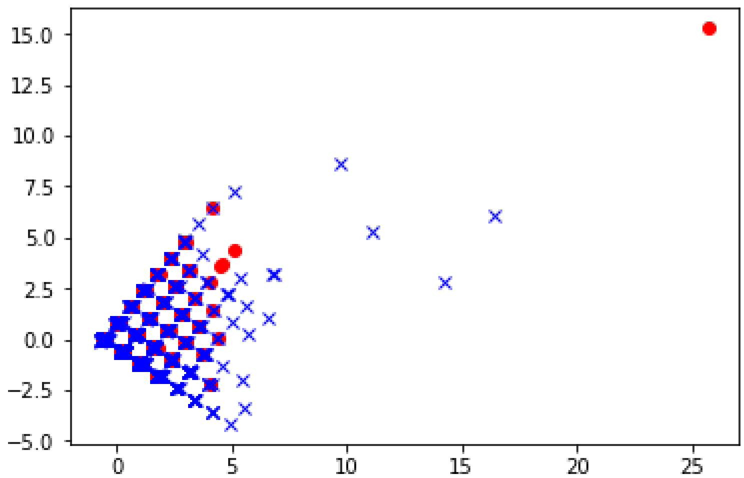
```
[[ -5.99354946e-04   3.95164941e-03  -9.30338083e-03   9.76942328e-03
   7.99767032e-01  -1.16030365e-04   5.94903989e-01   7.26399102e-02
   1.73572637e-04   3.14224279e-02]
 [ -1.57672227e-03  -8.65225859e-03  -1.41693175e-02   1.36630501e-02
```

-5.96926193e-01 2.48730319e-04 8.01927918e-01 -3.23901167e-03
-1.24896291e-03 1.06688940e-02]]

Code stub:

```
pca = PCA(n_components=2)
pca.fit(X_train)
print(pca.components_)
X_lowdim = pca.transform(X_train)
X_reconst = pca.inverse_transform(X_lowdim)
loss = ((X_train - X_reconst)**2).mean()*10
```

8.



Blue cross: Others, Red dot: American IPA

Code stub:

```
i = 0;
Am_X = []
Am_Y = []
Non_X = []
Non_Y = []
for d in X_lowdim:
    if(data[i]['beer/style'] == 'American IPA'):
        Am_X.append(d[0])
        Am_Y.append(d[1])
    else:
        Non_X.append(d[0])
        Non_Y.append(d[1])
    i = i+1

plt.plot(Am_X, Am_Y, 'ro', Non_X, Non_Y, 'bx')
plt.show()
```