def sigmoid(x):
return 1/(1+np.exp(-x))
def f(x, y, theta, lamb):
x = np.vstack((np.ones((1, x.shape[1])), x))
output = sigmoid(np.dot(theta.T,x))
return -sum(y\*np.log(output)+(1-y)\*np.log((1-output))) + lamb\*np.dot(theta.T,theta)
def df(x, y, theta, lamb):
x = np.vstack((np.ones((1, x.shape[1])), x))
output = sigmoid(np.dot(theta.T,x))
return np.dot(x,(output-y).T)+2\*lamb\*theta
def h(X, Y, theta):
X = np.vstack((np.ones((1, X.shape[1])), X))
h = np.dot(theta.T, X)
count = 0.0
for i in range(Y.shape[1]):
if Y[0,i] == 1 and h[0,i] > 0:
count += 1
elif Y[0,i] == 0 and h[0,i] < 0:
count += 1
return count / Y.shape[1]
def divide(dataset, label):
fakeset = []
realset = []
for i in range(len(dataset)):
if label[i]==1:
realset.append(dataset[i])
else:
fakeset.append(dataset[i])
return fakeset, realset
def combineData():
words = []
for i in real\_data:
words.extend(i)
for i in fake\_data:
words.extend(i)
return words
def transform(dataset):
word = combineData()
X = np.zeros((len(words),0))
for i in dataset:
init = np.zeros((len(word),1))
for j in range(len(word)):
if words[j] in i:
init[j][0] = 1
X = np.hstack((X,init))
return X
# part 4
words = combineData()
fTr,rTr = divide(training\_set, training\_label)
fV, rV = divide(validation\_set, validation\_label)
fTe, rTe = divide(test\_set, test\_label)
trainX = np.hstack((transform(fTr),transform(rTr)))
trainY = np.hstack((np.zeros((1,len(fTr))),np.ones((1,len(rTr)))))
validX = np.hstack((transform(fV),transform(rV)))
validY = np.hstack((np.zeros((1,len(fV))),np.ones((1,len(rV)))))
testX = np.hstack((transform(fTe),transform(rTe)))
testY = np.hstack((np.zeros((1,len(fTe))),np.ones((1,len(rTe)))))
def grad\_descent(f, df, x, y, init\_t, alpha, lamb):
epsilon=1e-5
prev\_t = init\_t - 10 \* epsilon
t = init\_t.copy()
iter = 0
max\_iter = 3000
loopNum = []
outTrain = []
outValid = []
outTest = []
while np.linalg.norm(t - prev\_t) > epsilon and iter < max\_iter:
prev\_t = t.copy()
t -= alpha \* df(x, y, t, lamb)
if iter % 30 == 0:
loopNum.append(iter)
outTrain.append(h(trainX,trainY,t))
outValid.append(h(validX,validY,t))
outTest.append(h(testX,testY,t))
iter += 1
out = [outTrain, outValid, outTest]
return t, out, loopNum
def part4():
theta0 = np.zeros((len(words)+1,1))
#tuneP = [3e-4,1e-3,3e-3,1e-2,3e-2,1e-1]
tuneP = [0.003]
alpha = 0.1
minP = -1.0
validAcc = []
for tune in tuneP:
theta, output, lN= grad\_descent(f, df, trainX, trainY, theta0, alpha, tune)
accr = h(validX,validY,theta)
validAcc.append(accr)
if accr > minP:
tunedP = tune
minP = accr
thetaP = theta
fig = plt.figure(10)
plt.semilogx(tuneP, validAcc)
plt.ylabel("Validation Set Accuracy")
plt.xlabel("tuned parameter")
plt.savefig("part4: tune parameter")
print ("optimal parameter"),tunedP, ("optimal accuracy"), minP
print ("training set accuracy"), h(trainX, trainY, thetaP)
print ("validation set accuracy"), h(validX, validY, thetaP)
print ("test set accuracy"), h(testX, testY, thetaP)
fig1 = plt.figure(20)
plt.plot(lN, output[0], label = "Train")
plt.plot(lN, output[1], label = "Validation")
plt.plot(lN, output[2], label = "Test")
plt.ylabel("training accuracy")
plt.xlabel("iteration")
plt.title("Part4: Learning curve")

return thetaP