In [1]:
```
import pandas as pd
import numpy as np
import gzip
import string

from collections import defaultdict
from sklearn.linear_model import LogisticRegression
```

In [2]:
```
def readJSON(path):
    for l in gzip.open(path, 'r+'):
        d = eval(l)
        u = d['userID']
        try:
            g = d['gameID']
        except Exception as e:
            g = None
        yield u,g,d

def parseData(path):
    for l in gzip.open(path, 'r+'):
        yield eval(l)
```

Problem 1

In [3]:
```
parsedT = list(parseData('train.json.gz'))
```

In [4]:
```
#Splits dataset into train and validation sets
train = []
val = []
for x in range(165000):
    train.append(parsedT[x])
for x in range(165000,175000):
    val.append(parsedT[x])
```

In [5]:
```
#Creates List For USER ITEM pair and a list for all unique games
UIV = [(d['userID'],d['gameID'],1) for d in val]
allGames = list(set([d['gameID'] for d in parsedT]))
uniqV = {}

#Loops through user item pair and creates dictionaries for each user an
d all games they have played
for x in UIV:
    if x[0] not in uniqV.keys():
        uniqV[x[0]] = [x[1]]
    else:
        uniqV[x[0]].append(x[1])

#Loops through rest of train data and adds other games users have playe
d
for x in train:
    if x['userID'] in uniqV.keys():
        uniqV[x['userID']].append(x['gameID'])

#Loops through each user in user item list and randomly samples game th
ey have never played and appends to list
for x in range(len(UIV)):
    #Finds the games from all games that the user has never played
    no_play = list(set(allGames) ^ set(uniqV[UIV[x][0]]))

    #Randomly samples a new game
    new_game = np.random.choice(no_play,1)[0]

    #Checks to see if this game has been sampled before
    if (UIV[x][0], new_game, 0) not in UIV:
        #Appends new sample game as a game the user has not played yet
        UIV.append((UIV[x][0], new_game, 0))

        #Appends sampled game to dictionary so as to say this game has
now been selected do not select again
        uniqV[UIV[x][0]].append(new_game)
```

In [6]:
```
gameCount = defaultdict(int)
totalPlayed = 0

for x in train:
    gameCount[x['gameID']] += 1
    totalPlayed += 1

mostPopular = [(gameCount[x], x) for x in gameCount]
mostPopular.sort()
mostPopular.reverse()
```

In [7]:
```
return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > totalPlayed/2: break
```

In [8]:
```
#The true values for the validation set
truePlay = [d[2] for d in UIV]

#The predicted value based on a 50th percentile threshold
playPred = []
for x in UIV:
    if x[1] in return1:
        playPred.append(1)
    else:
        playPred.append(0)
```

In [9]:
```
#Finds the True Positive, False Positive, True Negative, and False Nega
tive array values
TP_ = np.logical_and(playPred, truePlay)
FP_ = np.logical_and(playPred, np.logical_not(truePlay))
TN_ = np.logical_and(np.logical_not(playPred), np.logical_not(truePla
y))
FN_ = np.logical_and(np.logical_not(playPred), truePlay)

#Finds the number of True Positive, False Positive, True Negative, and
False Negative
TP = sum(TP_)
FP = sum(FP_)
TN = sum(TN_)
FN = sum(FN_)

# accuracy
accuracy = (TP + TN) / (TP + FP + TN + FN)
{'Accuracy':accuracy}
```

Out[9]:
```
{'Accuracy': 0.6806}
```

Problem 2

In [31]:
```
#Randomly choose values incrementing from the baseline of 2
per = [1,2,3,4,5,6,7,8,9,10]
acc = []
for thresh in per:
    return2 = set()
    count2 = 0
    for ic, i in mostPopular:
        count2 += ic
        return2.add(i)
        if count2 > totalPlayed/thresh: break

    #The predicted value based on the xth percentile threshold
    predImp = []
    for x in UIV:
        if x[1] in return2:
            predImp.append(1)
        else:
            predImp.append(0)

    #Finds the True Positive, False Positive, True Negative, and False
Negative array values
    TP_2 = np.logical_and(predImp, truePlay)
    FP_2 = np.logical_and(predImp, np.logical_not(truePlay))
    TN_2 = np.logical_and(np.logical_not(predImp), np.logical_not(trueP
lay))
    FN_2 = np.logical_and(np.logical_not(predImp), truePlay)

    #Finds the number of True Positive, False Positive, True Negative,
and False Negative
    TP2 = sum(TP_2)
    FP2 = sum(FP_2)
    TN2 = sum(TN_2)
    FN2 = sum(FN_2)

    # accuracy
    acc.append((TP2 + TN2) / (TP2 + FP2 + TN2 + FN2))
acc
```

Out[31]:
```
[0.5, 0.6806, 0.636, 0.6069, 0.586, 0.57195, 0.5639, 0.5555, 0.5502,
0.54495]
```

In [32]:
```
#Saw that dividing by 3 gave the highest value so narrowed results betw
een 2 and 3
per = [2.1,2.2,2.3,2.4,2.5,2.6,2.7,2.8,2.9]
acc = []
for thresh in per:
    return2 = set()
    count2 = 0
    for ic, i in mostPopular:
        count2 += ic
        return2.add(i)
        if count2 > totalPlayed/thresh: break

    #The predicted value based on the xth percentile threshold
    predImp = []
    for x in UIV:
        if x[1] in return2:
            predImp.append(1)
        else:
            predImp.append(0)

    #Finds the True Positive, False Positive, True Negative, and False
Negative array values
    TP_2 = np.logical_and(predImp, truePlay)
    FP_2 = np.logical_and(predImp, np.logical_not(truePlay))
    TN_2 = np.logical_and(np.logical_not(predImp), np.logical_not(trueP
lay))
    FN_2 = np.logical_and(np.logical_not(predImp), truePlay)

    #Finds the number of True Positive, False Positive, True Negative,
and False Negative
    TP2 = sum(TP_2)
    FP2 = sum(FP_2)
    TN2 = sum(TN_2)
    FN2 = sum(FN_2)

    # accuracy
    acc.append((TP2 + TN2) / (TP2 + FP2 + TN2 + FN2))
acc
```

Out[32]:
```
[0.6757, 0.6701, 0.6653, 0.6611, 0.65645, 0.65135, 0.64745, 0.6433,
0.6397]
```

In [45]:
```
#Tried multiplying be percentages instead
per = [0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9]
acc = []
for thresh in per:
    return2 = set()
    count2 = 0
    for ic, i in mostPopular:
        count2 += ic
        return2.add(i)
        if count2 > totalPlayed*thresh: break

    #The predicted value based on the xth percentile threshold
    predImp = []
    for x in UIV:
        if x[1] in return2:
            predImp.append(1)
        else:
            predImp.append(0)

    #Finds the True Positive, False Positive, True Negative, and False
Negative array values
    TP_2 = np.logical_and(predImp, truePlay)
    FP_2 = np.logical_and(predImp, np.logical_not(truePlay))
    TN_2 = np.logical_and(np.logical_not(predImp), np.logical_not(trueP
lay))
    FN_2 = np.logical_and(np.logical_not(predImp), truePlay)

    #Finds the number of True Positive, False Positive, True Negative,
and False Negative
    TP2 = sum(TP_2)
    FP2 = sum(FP_2)
    TN2 = sum(TN_2)
    FN2 = sum(FN_2)

    # accuracy
    acc.append((TP2 + TN2) / (TP2 + FP2 + TN2 + FN2))
acc
```

Out[45]:
```
[0.54495, 0.586, 0.6249, 0.65645, 0.6806, 0.6975, 0.70115, 0.6818, 0.
6264]
```

In [47]:
```
#Tightened the range
per = [0.61,0.62,0.63,0.64,0.65,0.66,0.67,0.68,0.69]
acc = []
for thresh in per:
    return2 = set()
    count2 = 0
    for ic, i in mostPopular:
        count2 += ic
        return2.add(i)
        if count2 > totalPlayed*thresh: break

    #The predicted value based on the xth percentile threshold
    predImp = []
    for x in UIV:
        if x[1] in return2:
            predImp.append(1)
        else:
            predImp.append(0)

    #Finds the True Positive, False Positive, True Negative, and False
Negative array values
    TP_2 = np.logical_and(predImp, truePlay)
    FP_2 = np.logical_and(predImp, np.logical_not(truePlay))
    TN_2 = np.logical_and(np.logical_not(predImp), np.logical_not(trueP
lay))
    FN_2 = np.logical_and(np.logical_not(predImp), truePlay)

    #Finds the number of True Positive, False Positive, True Negative,
and False Negative
    TP2 = sum(TP_2)
    FP2 = sum(FP_2)
    TN2 = sum(TN_2)
    FN2 = sum(FN_2)

    # accuracy
    acc.append((TP2 + TN2) / (TP2 + FP2 + TN2 + FN2))
acc
```

Out[47]:
```
[0.6987, 0.6989, 0.7006, 0.7002, 0.70115, 0.7021, 0.703, 0.70245, 0.7
0235]
```

In [48]:
```
{'Best threshold is totalPlayed*0.7': 0.70115}
```

Out[48]:
```
{'Best threshold is totalPlayed*0.7': 0.70115}
```

Problem 3

In [12]:
```
def Jaccard(user1,user2):
    user1 = set(user1)
    user2 = set(user2)
    numer = len(user1.intersection(user2))
    denom = len(user1.union(user2))
    if denom > 0:
        return numer/denom
    else:
        return 0

def findUsers(id):
    return [d['userID'] for d in train if d['gameID'] == id]
```

In [13]:
```
#Creates list unique userID and two dictionaries store validation users
and the games they played and all games
#and users that have played them
valID = set([d['userID'] for d in val])
valTrainPlay = {}
userGames = {}
for x in valID:
    valTrainPlay[x] = []
for x in train:
    if x['userID'] in valTrainPlay.keys():
        valTrainPlay[x['userID']].append(x['gameID'])
for x in allGames:
    userGames[x] = findUsers(x)
```

In [33]:
```
#Tried 0.1 and realized should choose values between 0 and 0.1
base = [0.01,0.02,0.03,0.04,0.05,0.06,0.07,0.08,0.09]
simil = []
for thresh in base:
    jPred = []
    for x in UIV:
        similarities = []
        #Finds all users that have played g
        g = userGames[x[1]]
        for y in valTrainPlay[x[0]]:
            #Finds all users that have played g' and computes Jaccard s
imilarity
            gPrime = userGames[y]
            similarities.append(Jaccard(g,gPrime))
        #Make prediction based on whether Jaccard similarity is greater
than threshold
        if max(similarities) >= thresh:
            jPred.append(1)
        else:
            jPred.append(0)
    #Finds the True Positive, False Positive, True Negative, and False
Negative array values
    TP_J = np.logical_and(jPred, truePlay)
    FP_J = np.logical_and(jPred, np.logical_not(truePlay))
    TN_J = np.logical_and(np.logical_not(jPred), np.logical_not(truePla
y))
    FN_J = np.logical_and(np.logical_not(jPred), truePlay)

    #Finds the number of True Positive, False Positive, True Negative,
and False Negative
    TPJ = sum(TP_J)
    FPJ = sum(FP_J)
    TNJ = sum(TN_J)
    FNJ = sum(FN_J)

    simil.append((TPJ + TNJ) / (TPJ + FPJ + TNJ + FNJ))
simil
```

Out[33]:
```
[0.5191, 0.60805, 0.67445, 0.6458, 0.58875, 0.55205, 0.53385, 0.5181,
0.50835]
```

In [34]:
```
#Saw the best accuracy around 0.03 and 0.04
base = [0.031,0.032,0.033,0.034,0.035,0.036,0.037,0.038,0.039]
simil = []
for thresh in base:
    jPred = []
    for x in UIV:
        similarities = []
        #Finds all users that have played g
        g = userGames[x[1]]
        for y in valTrainPlay[x[0]]:
            #Finds all users that have played g' and computes Jaccard s
imilarity
            gPrime = userGames[y]
            similarities.append(Jaccard(g,gPrime))
        #Make prediction based on whether Jaccard similarity is greater
than threshold
        if max(similarities) >= thresh:
            jPred.append(1)
        else:
            jPred.append(0)
    #Finds the True Positive, False Positive, True Negative, and False
Negative array values
    TP_J = np.logical_and(jPred, truePlay)
    FP_J = np.logical_and(jPred, np.logical_not(truePlay))
    TN_J = np.logical_and(np.logical_not(jPred), np.logical_not(truePla
y))
    FN_J = np.logical_and(np.logical_not(jPred), truePlay)

    #Finds the number of True Positive, False Positive, True Negative,
and False Negative
    TPJ = sum(TP_J)
    FPJ = sum(FP_J)
    TNJ = sum(TN_J)
    FNJ = sum(FN_J)

    simil.append((TPJ + TNJ) / (TPJ + FPJ + TNJ + FNJ))
simil
```

Out[34]:
```
[0.6747, 0.67405, 0.6745, 0.6719, 0.67, 0.66625, 0.6613, 0.65665, 0.6
5125]
```

In [52]:
```
{'Best threshold is >= 0.031': 0.6747}
```

Out[52]:
```
{'Best threshold is >= 0.031': 0.6747}
```

Problem 4

In [92]:
```
cPred = []
return3 = set()
count3 = 0
for ic, i in mostPopular:
    count3 += ic
    return3.add(i)
    if count3 > totalPlayed*0.7: break
for x in UIV:
    similarities = []
    #Finds all users that have played g
    gC = userGames[x[1]]
    for y in valTrainPlay[x[0]]:
        #Finds all users that have played g' and computes Jaccard simil
arity
        gCPrime = userGames[y]
        similarities.append(Jaccard(gC,gCPrime))
    #Make prediction based on Jaccard threshold and popularity threshol
d
    if (max(similarities) >= 0.031) and (x[1] in return3):
        cPred.append(1)
    elif (max(similarities) < 0.031) and (x[1] in return3):
        cPred.append(1)
    else:
        cPred.append(0)

#Finds the True Positive, False Positive, True Negative, and False Nega
tive array values
TP_C = np.logical_and(cPred, truePlay)
FP_C = np.logical_and(cPred, np.logical_not(truePlay))
TN_C = np.logical_and(np.logical_not(cPred), np.logical_not(truePlay))
FN_C = np.logical_and(np.logical_not(cPred), truePlay)

#Finds the number of True Positive, False Positive, True Negative, and
False Negative
TPC = sum(TP_C)
FPC = sum(FP_C)
TNC = sum(TN_C)
FNC = sum(FN_C)

{'Accuracy':(TPC + TNC) / (TPC + FPC + TNC + FNC)}
```

Out[92]:
```
{'Accuracy': 0.70115}
```

Problem 5

Kaggle Username: Anthony Fong

In [93]:
```
predictions = open("predictions_Played.txt", 'w')
for l in open("pairs_Played.txt"):
    if l.startswith("userID"):
        #header
        predictions.write(l)
        continue
    u,g = l.strip().split('-')
    similarities = []
    #Finds all users that have played g
    if (g in userGames.keys()) and (u in valTrainPlay.keys()):
        gC = userGames[g]
        for y in valTrainPlay[u]:
            #Finds all users that have played g' and computes Jaccard s
imilarity
            gCPrime = userGames[y]
            similarities.append(Jaccard(gC,gCPrime))
    if len(similarities) == 0:
        greatBig = 0
    if len(similarities) != 0:
        greatBig = max(similarities)
    if greatBig >= 0.031 and g in return3:
        predictions.write(u + '-' + g + ",1\n")
    elif greatBig < 0.031 and g in return3:
        predictions.write(u + '-' + g + ",1\n")
    else:
        predictions.write(u + '-' + g + ",0\n")

predictions.close()
```

Problem 6

In [16]:
```
parsedC = list(parseData('train_Category.json.gz'))
```

In [17]:
```
#Splits dataset into train and validation splits
trainC = []
valC = []
for x in range(165000):
    trainC.append(parsedC[x])
for x in range(165000,175000):
    valC.append(parsedC[x])
```

In [18]:
```
### Just take the most popular words...

wordCount = defaultdict(int)
punctuation = set(string.punctuation)
for d in trainC:
    r = ''.join([c for c in d['text'].lower() if not c in punctuation])
    for w in r.split():
        wordCount[w] += 1

counts = [(wordCount[w], w) for w in wordCount]
counts.sort()
counts.reverse()

words = [x[1] for x in counts[:1000]]
counts[:10]
```

Out[18]:
```
[(544597, 'the'),
 (317620, 'and'),
 (305414, 'a'),
 (291882, 'to'),
 (245359, 'game'),
 (227234, 'of'),
 (208417, 'is'),
 (200633, 'you'),
 (195953, 'i'),
 (190966, 'it')]
```

Problem 7

Complete

In [19]:
```
wordId = dict(zip(words, range(len(words))))
wordSet = set(words)

def feature(datum):
    feat = [0]*len(words)
    r = ''.join([c for c in datum['text'].lower() if not c in punctuati
on])
    for w in r.split():
        if w in words:
            feat[wordId[w]] += 1
    return feat

#Creates feature vector using feature function and creates list of genr
eID values
bow = [feature(d) for d in trainC]
genreID = [d['genreID'] for d in trainC]
```

In [20]:
```
#Creates validation x and y sets for prediction
x_val = [feature(d) for d in valC]
y_val = [d['genreID'] for d in valC]
```

In [21]:
```
#Creates Logistic Regressor and trains on train feature vector and genr
eID to predict on validation set
clf = LogisticRegression(max_iter=5000).fit(bow,genreID)
bowPred = clf.predict(x_val)
```

In [24]:
```
#Finds the True Positive, False Positive, True Negative, and False Nega
tive array values
TP_Bag = np.logical_and(bowPred, y_val)
FP_Bag = np.logical_and(bowPred, np.logical_not(y_val))
TN_Bag = np.logical_and(np.logical_not(bowPred), np.logical_not(y_val))
FN_Bag = np.logical_and(np.logical_not(bowPred), y_val)

#Finds the number of True Positive, False Positive, True Negative, and
False Negative
TPBag = sum(TP_Bag)
FPBag = sum(FP_Bag)
TNBag = sum(TN_Bag)
FNBag = sum(FN_Bag)

{'Accuracy': (TPBag + TNBag) / (TPBag + FPBag + TNBag + FNBag)}
```

Out[24]:
```
{'Accuracy': 0.6922}
```

Problem 8

Kaggle Username: Anthony Fong

In [36]:
```
def featureMod(datum,much):
    feat = [0]*len(words[:much])
    r = ''.join([c for c in datum['text'].lower() if not c in punctuati
on])
    for w in r.split():
        if w in words[:much]:
            feat[wordId2[w]] += 1
    return feat
```

In [37]:
```
#Tries different C and size values and stores it in dictionary
c = [0.000001,0.00001,0.0001,0.001]
size = [250,500,750]
grid = {}
for x in c:
    curr = []
    for y in size:
        wordId2 = dict(zip(words[:y], range(len(words[:y]))))
        wordSet2 = set(words[:y])
        bow2 = [featureMod(d,y) for d in trainC]
        x_val2 = [featureMod(d,y) for d in valC]
        clf2 = LogisticRegression(C=x,max_iter=1000).fit(bow2,genreID)
        bowPred2 = clf2.predict(x_val2)

        #Finds the True Positive, False Positive, True Negative, and Fa
lse Negative array values
        TP_Bag2 = np.logical_and(bowPred2, y_val)
        FP_Bag2 = np.logical_and(bowPred2, np.logical_not(y_val))
        TN_Bag2 = np.logical_and(np.logical_not(bowPred2), np.logical_n
ot(y_val))
        FN_Bag2 = np.logical_and(np.logical_not(bowPred2), y_val)

        #Finds the number of True Positive, False Positive, True Negati
ve, and False Negative
        TPBag2 = sum(TP_Bag2)
        FPBag2 = sum(FP_Bag2)
        TNBag2 = sum(TN_Bag2)
        FNBag2 = sum(FN_Bag2)

        curr.append((TPBag2 + TNBag2) / (TPBag2 + FPBag2 + TNBag2 + FNB
ag2))
    grid[x] = curr
grid
```

Out[37]:
```
{1e-06: [0.5533, 0.5533, 0.5533],
 1e-05: [0.5556, 0.5564, 0.557],
 0.0001: [0.5695, 0.5792, 0.5822],
 0.001: [0.595, 0.6248, 0.637]}
```

In [43]:
```
#Creates Logistic Regressor and trains on train feature vector and genr
eID to predict on validation set
words2 = [x[1] for x in counts[:1500]]
wordId2 = dict(zip(words2, range(len(words2))))
wordSet2 = set(words2)
def feature1500(datum):
    feat = [0]*len(words2)
    r = ''.join([c for c in datum['text'].lower() if not c in punctuati
on])
    for w in r.split():
        if w in words2:
            feat[wordId2[w]] += 1
    return feat
bow2 = [feature1500(d) for d in trainC]
x_val2 = [feature1500(d) for d in valC]
clfBest = LogisticRegression(max_iter=5000).fit(bow2,genreID)
bowPredBest = clfBest.predict(x_val2)

#Finds the True Positive, False Positive, True Negative, and False Nega
tive array values
TP_Bag2 = np.logical_and(bowPredBest, y_val)
FP_Bag2 = np.logical_and(bowPredBest, np.logical_not(y_val))
TN_Bag2 = np.logical_and(np.logical_not(bowPredBest), np.logical_not(y_
val))
FN_Bag2 = np.logical_and(np.logical_not(bowPredBest), y_val)

#Finds the number of True Positive, False Positive, True Negative, and
False Negative
TPBag2 = sum(TP_Bag2)
FPBag2 = sum(FP_Bag2)
TNBag2 = sum(TN_Bag2)
FNBag2 = sum(FN_Bag2)

{'Accuracy': (TPBag2 + TNBag2) / (TPBag2 + FPBag2 + TNBag2 + FNBag2)}
```

Out[43]:
```
{'Accuracy': 0.7154}
```

In [71]:
```
parsedCat = list(parseData('test_Category.json.gz'))
```

In [75]:
```
testSet = [feature1500(d) for d in parsedCat]
```

In [77]:
```
cat = clfBest.predict(testSet)
```

In [87]:
```
### Category prediction baseline: Just consider some of the most common
words from each category
hold = 0
predictions = open("predictions_Category.txt", 'w')
for l in open("pairs_Category.txt"):
    if l.startswith("userID"):
        #header
        predictions.write(l)
        continue
    u,r = l.strip().split('-')
    predictions.write(u + '-' + r + "," + str(cat[hold]) + "\n")
    hold += 1
predictions.close()
```

In [ ]:
```

```