SimilarityBasedPredictionAndTextMining

December 27, 2020

[1]:
```
import gzip
import sklearn
from sklearn import linear_model
from collections import defaultdict
import random
import math
import numpy as np
from sklearn.metrics import jaccard_score as jaccard
```

1 Tasks (Play prediction)

1.0.1 Play prediction (both classes)

Predict given a (user,game) pair from 'pairs Played.txt' whether the user would play the game (0 or 1). Accuracy will be measured in terms of the categorization accuracy (fraction of correct predictions). The test set has been constructed such that exactly 50% of the pairs correspond to played games and the other 50% do not.

[2]:
```
def parse(f):
    for l in gzip.open(f):
        yield eval(l)
```

[3]:
```
train_json = list(parse("data/train.json.gz"))
```

[4]:
```
train_json[0]
```

[4]:
```
{'hours': 0.3,
 'gameID': 'b96045472',
 'hours_transformed': 0.37851162325372983,
 'early_access': False,
 'date': '2015-04-08',
 'text': '+1',
 'userID': 'u01561183'}
```

[5]:
```
data = [[d['userID'],d['gameID'],1] for d in train_json]

split = 165000
train = train_json[:split]
```

```
ug_train = [[d['userID'],d['gameID'],1] for d in train]
validation = train_json[split:]
ug_valid = [[d['userID'],d['gameID'],1] for d in validation]
```

[ ]:

Task 1

[ ]:

[6]:
```
def readJSON(path):
  for l in gzip.open(path, 'rt'):
    d = eval(l)
    u = d['userID']
    try:
      g = d['gameID']
    except Exception as e:
      g = None
    yield u,g,d
```

[ ]:

[ ]:
```
# ### Time-played baseline: compute averages for each user, or return the␣
 ↪global average if we've never seen the user before

# allHours = []
# userHours = defaultdict(list)

# for user,game,d in readJSON("data/train.json.gz"):
#   h = d['hours_transformed']
#   allHours.append(h)
#   userHours[user].append(h)

# globalAverage = sum(allHours) / len(allHours)
# userAverage = {}
# for u in userHours:
#   userAverage[u] = sum(userHours[u]) / len(userHours[u])
```

[ ]:
```
# predictions = open("predictions_Hours.txt", 'w')
# for l in open("data/pairs_Hours.txt"):
#   if l.startswith("userID"):
#     #header
#     predictions.write(l)
#     continue
#   u,g = l.strip().split('-')
#   if u in userAverage:
#     predictions.write(u + '-' + g + ',' + str(userAverage[u]) + '\n')
```

```
#   else:
#     predictions.write(u + '-' + g + ',' + str(globalAverage) + '\n')

# predictions.close()
```

[ ]:
```
### Would-play baseline: just rank which games are popular and which are not,␣
 ↪and return '1' if a game is among the top-ranked

# gameCount = defaultdict(int)
# totalPlayed = 0

# for user,game,_ in readJSON("data/train.json.gz"):
#   gameCount[game] += 1
#   totalPlayed += 1

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

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

[ ]:
```
# predictions = open("predictions_Played.txt", 'w')
# for l in open("data/pairs_Played.txt"):
#   if l.startswith("userID"):
#     #header
#     predictions.write(l)
#     continue
#   u,g = l.strip().split('-')
#   if g in return1:
#     predictions.write(u + '-' + g + ",1\n")
#   else:
#     predictions.write(u + '-' + g + ",0\n")

# predictions.close()
```

[ ]:
```
# ### Category prediction baseline: Just consider some of the most common words␣
 ↪from each category

# catDict = {
#   "Action": 0,
#   "Strategy": 1,
#   "RPG": 2,
```

```
#   "Adventure": 3,
#   "Sport": 4
# }

# predictions = open("predictions_Category.txt", 'w')
# predictions.write("userID-reviewID,prediction\n")
# for u,_,d in readJSON("data/test_Category.json.gz"):
#   cat = catDict['Action'] # If there's no evidence, just choose the most␣
 ↪common category in the dataset
#   words = d['text'].lower()
#   if 'strategy' in words:
#     cat = catDict['Strategy']
#   if 'rpg' in words:
#     cat = catDict['RPG']
#   if 'adventure' in words:
#     cat = catDict['Adventure']
#   if 'sport' in words:
#     cat = catDict['Sport']
#   predictions.write(u + '-' + d['reviewID'] + "," + str(cat) + "\n")

# predictions.close()
```

Task 1

[7]:
```
#gets users per game and games per user from entire dataset
usersPerGame = defaultdict(set)
gamesPerUser = defaultdict(set)
uniqueGames = set()

for d in data:
    u, g = d[0], d[1]
    usersPerGame[g].add(u)
    gamesPerUser[u].add(g)
    uniqueGames.add(g)
```

[8]:
```
users_valid = [d[0] for d in ug_valid]
```

[9]:
```
ug_valid_neg= []
for u in users_valid:
    gamesNotPlayed = uniqueGames - gamesPerUser[u]
    randomGame = random.choice(list(gamesNotPlayed))
    ug_valid_neg.append([u, randomGame,0])
```

[10]:
```
ug_valid_neg[:10]
```

[10]:
```
[['u49969792', 'b08286994', 0],
 ['u33147591', 'b41038309', 0],
 ['u00954406', 'b81039440', 0],
 ['u40416473', 'b73515692', 0],
 ['u08125051', 'b73825833', 0],
 ['u16794935', 'b65708637', 0],
 ['u42345370', 'b86627453', 0],
 ['u40134397', 'b99600313', 0],
 ['u38053603', 'b71138223', 0],
 ['u25257262', 'b22856743', 0]]
```

[11]:
```
ug_valid[:10]
```

[11]:
```
[['u49969792', 'b25961467', 1],
 ['u33147591', 'b73229067', 1],
 ['u00954406', 'b37068085', 1],
 ['u40416473', 'b85572219', 1],
 ['u08125051', 'b88846011', 1],
 ['u16794935', 'b19939295', 1],
 ['u42345370', 'b31288876', 1],
 ['u40134397', 'b22332890', 1],
 ['u38053603', 'b44539786', 1],
 ['u25257262', 'b64105731', 1]]
```

[12]:
```
ug_valid_build = ug_valid + ug_valid_neg
```

[13]:
```
ug_valid_build[9995:10005]
```

[13]:
```
[['u90835702', 'b62891570', 1],
 ['u40505592', 'b75563467', 1],
 ['u67709233', 'b14676161', 1],
 ['u79727950', 'b36105300', 1],
 ['u25903175', 'b50879604', 1],
 ['u49969792', 'b08286994', 0],
 ['u33147591', 'b41038309', 0],
 ['u00954406', 'b81039440', 0],
 ['u40416473', 'b73515692', 0],
 ['u08125051', 'b73825833', 0]]
```

[14]:
```
len(ug_valid_build)
```

[14]:
```
20000
```

[15]:
```
Xvalid = [[d[0],d[1]] for d in ug_valid_build]
yvalid = [d[2] for d in ug_valid_build]

Xtrain = [[d[0], d[1]] for d in ug_train]
```

```
ytrain = [d[2] for d in ug_train]
```

[16]:
```
### Would-play baseline: just rank which games are popular and which are not,␣
 ↪and return '1' if a game is among the top-ranked
##task 1 baseline
def baselinePreds(Xvalid, threshold):
    gameCount = defaultdict(int)
    totalPlayed = 0

    for user,game,_ in readJSON("data/train.json.gz"):
      gameCount[game] += 1
      totalPlayed += 1



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

    return1 = set()
    count = 0
    for ic, i in mostPopular:
      count += ic
      return1.add(i)
      if count > totalPlayed/threshold: break


    #task 1 predictions
    predictions = []
    for user, game in Xvalid:
        if game in return1:
            predictions.append(1)
        else:
            predictions.append(0)

    return predictions
```

[17]:
```
def computeAccuracy(preds, true):
    correct = np.array(preds) == np.array(true)
    return sum(correct) / len(correct)
```

[18]:
```
preds = baselinePreds(Xvalid,2)
```

[19]:
```
#accuracy of baseline model on validation set
th2_acc = computeAccuracy(preds, yvalid)
th2_acc
```

[19]:
```
0.68205
```

Task 2

my choice for thresholds was to keep it low, and see if being more selective in terms of popularity percentile would yield a better model, as you can see it performed slightly better in terms of accuracy

[20]:
```
# my choice for thresholds was to keep it low, and see if being more selective
#in terms of popularity percentile would yield a better model

def bestThreshold():
    best_th = 2
    best_acc = th2_acc
    for i in np.arange(1,2.05,.05):
        acc = computeAccuracy(baselinePreds(Xvalid,i), yvalid)
        if acc > best_acc:
            best_th = i
            best_acc = acc
    return best_th, best_acc

bestThreshold()
```

[20]:
```
(1.5000000000000004, 0.70355)
```

Task 3

[21]:
```
import matplotlib.pyplot as plt
```

[22]:
```
def Jaccard(s1, s2):
    numer = len(s1.intersection(s2))
    denom = len(s1.union(s2))
    return numer / denom
```

[ ]:
```

```

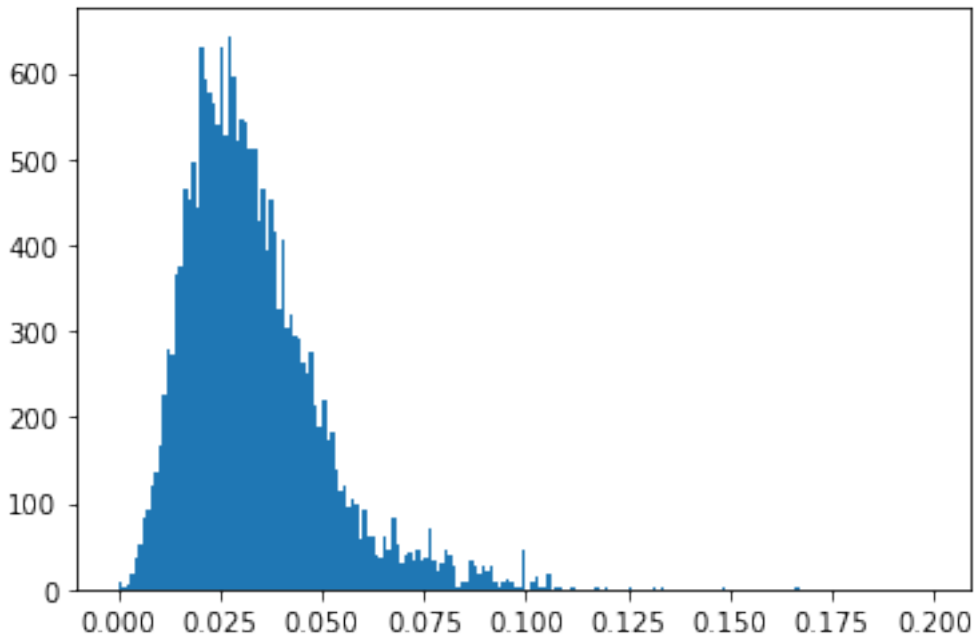
[23]:
```
#all users for a specific game in training data
#all games for a specific user in training data
usersPerGame = defaultdict(set)
gamesPerUser = defaultdict(set)
for u,g in Xtrain:
    usersPerGame[g].add(u)
    gamesPerUser[u].add(g)
```

[24]:
```
def similarityScores(user, game):
    #consider g' in training set that a user has played
    g_primes = gamesPerUser[user]
```

```
    similarities = []

    if len(g_primes) == 0:
        similarities.append(0)
        return similarities

    for g_prime in g_primes:
        if g_prime == g:
            continue
        #users in training data who have played g
        ugTrain = usersPerGame[game]
        #users who have played g'
        ugPrime = usersPerGame[g_prime]
        if len(ugPrime) == 0:
            similarities.append(0)
        else:
            similarities.append(Jaccard(ugTrain,ugPrime))

    return similarities
```

[25]:
```
max_scores = []
for user, game in Xvalid:
    max_scores.append(np.max(similarityScores(user, game)))
```

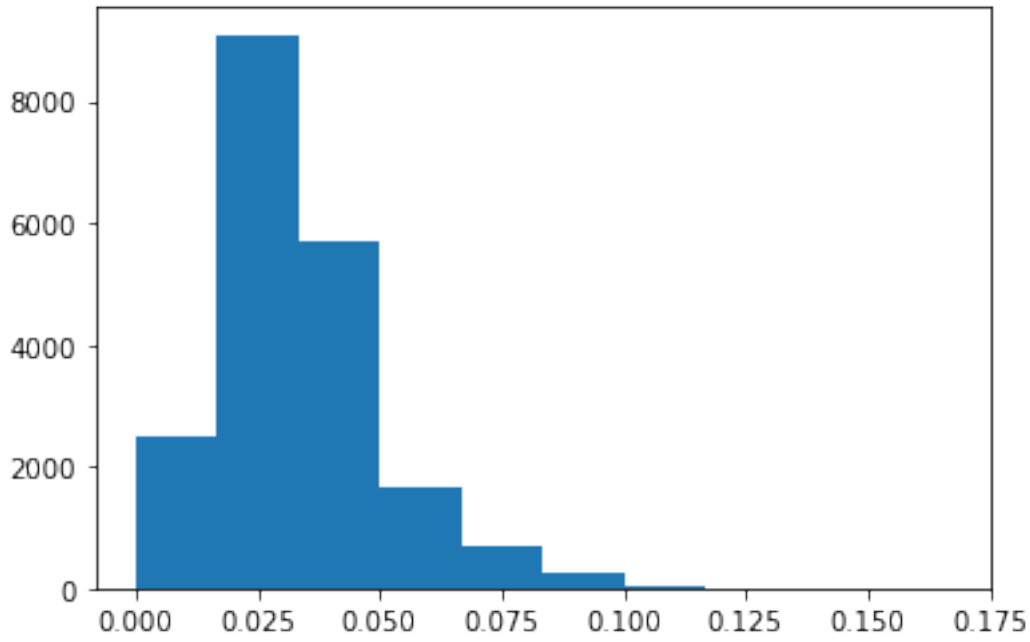
[26]:
```
plt.hist(max_scores, bins = np.arange(0,.2,.001))
plt.show()
```



[ ]:

[27]:
```
plt.hist(max_scores)
```

[27]:
```
(array([2.502e+03, 9.094e+03, 5.712e+03, 1.680e+03, 6.940e+02, 2.630e+02,
        4.800e+01, 4.000e+00, 2.000e+00, 1.000e+00]),
 array([0.        , 0.01666667, 0.03333333, 0.05      , 0.06666667,
        0.08333333, 0.1       , 0.11666667, 0.13333333, 0.15      ,
        0.16666667]),
 <a list of 10 Patch objects>)
```



[28]:
```
def jaccard_preds(max_scores,th):
    predictions = []
    for sim in max_scores:
        if sim > th:
            predictions.append(1)
        else:
            predictions.append(0)
    return predictions
```

[29]:
```
def bestThreshold():
    best_th = 0
    best_acc = 0
    for th in np.arange(0,.1,.001):
```

```
        preds = jaccard_preds(max_scores,th)
        acc = computeAccuracy(preds, yvalid)
        if acc > best_acc:
            best_th = th
            best_acc = acc

    return (best_th, best_acc)
bestThreshold()
```

[29]: (0.03, 0.67515)

Task 4

Incorporating both a Jaccard-based threshold and a popularity threshold seemed to perform better than just a Jaccard-based threshold alone, but not quite as well as a popularity based threshold alone.

[30]:
```
def baselinePreds4(ug_valid_build, j_th,pop_th):
    gameCount = defaultdict(int)
    totalPlayed = 0

    for user,game,_ in readJSON("data/train.json.gz"):
        gameCount[game] += 1
        totalPlayed += 1

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

    return1 = set()
    count = 0
    for ic, i in mostPopular:
        count += ic
        return1.add(i)
        if count > totalPlayed/pop_th: break


#     #task 1 predictions
#     predictions = []
#     for user, game, _ in ug_valid_build:
#         if game in return1:
#             predictions.append(1)
#         else:
#             predictions.append(0)


#     predictions_or = []
```

```
#     for user, game, _ in ug_valid_build:
#         mostSim = similarityScores(user, game)
#         if (game in return1) or (max(mostSim) >= j_th):
#             predictions_or.append(1)
#         else:
#             predictions_or.append(0)

    predictions_and = []
    for user, game, _ in ug_valid_build:
        mostSim = similarityScores(user, game)
        if (game in return1) and (max(mostSim) >= j_th):
            predictions_and.append(1)
        else:
            predictions_and.append(0)
    return predictions_and
```

[32]:
```
preds_and = baselinePreds4(ug_valid_build,.03,1.5)
and_acc = computeAccuracy(preds_and, yvalid)
print(and_acc)
```

```
0.69435
```

redesign model

when passing the best threshold from q2 and q3 into the function above i mistyped one of the thresholds and managed to get an unexpected accuracy, so I ran a script to see if combining different jaccard-based thresholds and popularity thresholds would give better accuracy, or if the optimal thresholds from q2 and q3 were still the best, unfortunately the run time was a couple of hours because I was testing a large range of every possible combination for jaccard/popularity thresholds, i also redid some of the conditional statements for a more lenient predictor

the pseudo code for finding a better threshold combo is having two for loops, one outer and one nested, one loops through a range in jaccard thresholds and the other loops through some popularity threshold, then i run baselinePreds4 passing in those thresholds, compute accuracies and print out the accuracies with the thresholds to see which combo gives the best result in the redesigned baseline predictor

[33]:
```
def baselinePreds4(ug_valid_build, j_th,pop_th):
    gameCount = defaultdict(int)
    totalPlayed = 0

    for user,game,_ in readJSON("data/train.json.gz"):
        gameCount[game] += 1
        totalPlayed += 1

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

```
    mostPopular.reverse()

    return1 = set()
    count = 0
    for ic, i in mostPopular:
        count += ic
        return1.add(i)
        if count > totalPlayed/pop_th: break


#     #task 1 predictions
#     predictions = []
#     for user, game, _ in ug_valid_build:
#         if game in return1:
#             predictions.append(1)
#         else:
#             predictions.append(0)


#     predictions_or = []
#     for user, game, _ in ug_valid_build:
#         mostSim = similarityScores(user, game)
#         if (game in return1) or (max(mostSim) >= j_th):
#             predictions_or.append(1)
#         else:
#             predictions_or.append(0)

    predictions_and = []
    for user, game, _ in ug_valid_build:
        mostSim = similarityScores(user, game)
        if (game in return1):
            predictions_and.append(1)
        else:
            if (game not in return1) and (max(mostSim) > j_th):
                predictions_and.append(1)
            else:
                predictions_and.append(0)
    return predictions_and
```

[34]:
```
preds_and = baselinePreds4(ug_valid_build,.048,1.51)
and_acc = computeAccuracy(preds_and, yvalid)
print(and_acc)
```

```
0.70595
```

Task 5

Kaggle Username: anthonylimon

[36]:
```
j_th = .048
pop_th = 1.51

gameCount = defaultdict(int)
totalPlayed = 0

for user,game,_ in readJSON("data/train.json.gz"):
    gameCount[game] += 1
    totalPlayed += 1

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

return1 = set()
count = 0
for ic, i in mostPopular:
    count += ic
    return1.add(i)
    if count > totalPlayed/pop_th: break

# predictions_and = []
# for user, game, _ in ug_valid_build:
#     mostSim = similarityScores(user, game)
#     if (game in return1):
#         predictions_and.append(1)
#     else:
#         if (game not in return1) and (max(mostSim) >= j_th):
#             predictions_and.append(1)
#         else:
#             predictions_and.append(0)




predictions = open("predictions_Played.txt", 'w')
for l in open("data/pairs_Played.txt"):
    if l.startswith("userID"):
    #header
        predictions.write(l)
        continue
    u,g = l.strip().split('-')
    mostSim = similarityScores(u,g)
    if g in return1:
```

```
        predictions.write(u + '-' + g + ",1\n")
    else:

        if (g not in return1) and (max(mostSim) >= j_th):
            predictions.write(u + '-' + g + ",1\n")
        else:
            predictions.write(u + '-' + g + ",0\n")

predictions.close()
```

2 Tasks (Category prediction)

2.0.1 For these experiments, you may want to select a smaller dictionary size (i.e., fewer words), or a smaller training set size, if the experiments are taking too long to run.

Predict the category of a game from a review. Five categories are used for this task, which can be seen in the baseline program, namely Action, Strategy, RPG, Adventure, and Sport. Performance will be measured in terms of the fraction of correct classifications.

[37]:
```
import string
import nltk
from nltk.stem.porter import *
from scipy.sparse import lil_matrix
```

[38]:
```
train_cat_json = list(parse("data/train_Category.json.gz"))
```

[39]:
```
train_cat_json[0]
```

[39]:
```
{'userID': 'u74382925',
 'genre': 'Adventure',
 'early_access': False,
 'reviewID': 'r75487422',
 'hours': 4.1,
 'text': 'Short Review:\nA good starting chapter for this series, despite the
main character being annoying (for now) and a short length. The story is good
and actually gets more interesting. Worth the try.\nLong Review:\nBlackwell
Legacy is the first on the series of (supposedly) 5 games that talks about the
main protagonist, Rosangela Blackwell, as being a so called Medium, and in this
first chapter we get to know how her story will start and how she will meet her
adventure companion Joey…and really, that\'s really all for for now and
that\'s not a bad thing, because in a way this game wants to show how hard her
new job is, and that she cannot escape her destiny as a medium.\nMy biggest
complain for this chapter, except the short length, it\'s the main protagonist
being a "bit" too annoying to be likeable, and most of her dialogues will always
be about complaining or just be annoyed. Understandable, sure, but lighten\' up
will ya!?\nHowever, considering that in the next installments she will be much
```

```
more likeable and kind of interesting, I\'d say give it a shot and see if you
like it: if you hate this first game, you might like the next, or can always
stop here.\nI recommend it.',
 'genreID': 3,
 'date': '2014-02-07'}
```

[40]:
```
#review data
data = [[d['genreID'],d['text']] for d in train_cat_json]

split = 165000

train = train_cat_json[:split]
Xtrain = [[d['text']] for d in train]
ytrain = [[d['genreID']] for d in train]

validation = train_cat_json[split:]
Xvalid = [[d['text']] for d in validation]
yvalid = [d['genreID'] for d in validation]
```

[ ]:
```

```

[41]:
```
### Ignore capitalization and remove punctuation

wordCount = defaultdict(int)
punctuation = set(string.punctuation)
totalCount = 0
for d in Xtrain:
    r = ''.join([c for c in d[0].lower() if not c in punctuation])
    for w in r.split():
        wordCount[w] += 1
        totalCount +=1

print(len(wordCount))
```

```
154889
```

Task 6

1000 most common words

[42]:
```
#top 1000 most frequent words in training set
sorted(wordCount.items(),key=lambda v: v[1],reverse=True)[:1000]
```

[42]:
```
[('the', 544597),
 ('and', 317620),
 ('a', 305414),
 ('to', 291882),
 ('game', 245359),
```

```
('of', 227234),
('is', 208417),
('you', 200633),
('i', 195953),
('it', 190966),
('this', 158622),
('in', 132348),
('that', 115044),
('for', 105210),
('but', 100985),
('with', 91007),
('its', 83631),
('are', 77355),
('on', 72366),
('as', 69754),
('not', 65475),
('have', 63625),
('if', 58019),
('like', 57252),
('be', 56116),
('can', 50151),
('so', 48201),
('your', 47720),
('was', 46825),
('just', 45696),
('or', 45686),
('all', 45297),
('good', 45152),
('more', 42838),
('one', 42197),
('at', 41525),
('play', 40611),
('get', 39537),
('my', 38847),
('games', 37677),
('there', 37202),
('fun', 36986),
('really', 36441),
('some', 35836),
('an', 35701),
('very', 35477),
('from', 34854),
('time', 32421),
('will', 32065),
('they', 31200),
('me', 30495),
('has', 30323),
```

```
('great', 30314),
('out', 29522),
('up', 29314),
('story', 29273),
('no', 28975),
('even', 27341),
('only', 25972),
('what', 25911),
('dont', 25675),
('do', 25627),
('which', 25568),
('when', 25293),
('about', 24393),
('much', 23984),
('would', 23780),
('by', 22763),
('still', 21979),
('well', 21797),
('than', 21644),
('also', 21088),
('first', 21085),
('them', 19560),
('because', 19217),
('other', 18936),
('played', 18878),
('then', 18826),
('gameplay', 18395),
('into', 18069),
('best', 17463),
('playing', 17196),
('too', 17176),
('make', 17134),
('way', 16600),
('how', 16345),
('most', 16149),
('better', 15721),
('had', 15622),
('pretty', 15607),
('new', 15544),
('2', 15473),
('people', 15043),
('any', 14772),
('lot', 14727),
('now', 14662),
('after', 14539),
('while', 14286),
('where', 14277),
```

```
('want', 14166),
('im', 13751),
('hours', 13709),
('graphics', 13660),
('many', 13608),
('who', 13512),
('go', 13410),
('bad', 13345),
('recommend', 13094),
('over', 13058),
('through', 13043),
('worth', 13035),
('little', 12738),
('buy', 12665),
('love', 12408),
('feel', 12330),
('1', 12150),
('youre', 12097),
('though', 12060),
('every', 11982),
('cant', 11955),
('their', 11948),
('being', 11870),
('few', 11800),
('think', 11689),
('could', 11667),
('been', 11620),
('say', 11619),
('see', 11576),
('ever', 11569),
('level', 11568),
('got', 11488),
('characters', 11297),
('things', 11179),
('3', 11128),
('back', 11074),
('bit', 11049),
('same', 10955),
('ive', 10838),
('nice', 10771),
('around', 10763),
('find', 10757),
('made', 10735),
('different', 10679),
('were', 10566),
('know', 10420),
('again', 10377),
```

```
('never', 10129),
('something', 9820),
('should', 9745),
('off', 9617),
('theres', 9558),
('need', 9466),
('thing', 9336),
('each', 9266),
('hard', 9225),
('experience', 9224),
('amazing', 9160),
('thats', 9141),
('end', 9118),
('actually', 9083),
('doesnt', 9064),
('down', 9048),
('world', 9039),
('enough', 8937),
('take', 8884),
('short', 8868),
('those', 8767),
('before', 8740),
('give', 8708),
('interesting', 8638),
('going', 8623),
('makes', 8612),
('use', 8609),
('does', 8607),
('free', 8589),
('quite', 8579),
('right', 8565),
('character', 8510),
('work', 8461),
('times', 8459),
('long', 8298),
('these', 8288),
('system', 8280),
('levels', 8224),
('player', 8162),
('did', 8122),
('since', 8121),
('enemies', 8094),
('however', 8077),
('far', 8053),
('isnt', 8052),
('try', 7997),
('everything', 7987),
```

```
('point', 7934),
('music', 7872),
('here', 7822),
('old', 7791),
('look', 7763),
('kill', 7715),
('combat', 7687),
('steam', 7610),
('weapons', 7474),
('two', 7467),
('didnt', 7446),
('overall', 7439),
('without', 7407),
('series', 7379),
('why', 7327),
('easy', 7326),
('multiplayer', 7324),
('1010', 7304),
('own', 7247),
('players', 7235),
('start', 7190),
('nothing', 7178),
('money', 7150),
('youll', 6994),
('review', 6979),
('enjoy', 6905),
('awesome', 6869),
('mode', 6833),
('another', 6700),
('once', 6675),
('feels', 6647),
('such', 6629),
('puzzles', 6528),
('original', 6488),
('we', 6457),
('may', 6396),
('price', 6384),
('simple', 6333),
('4', 6315),
('sure', 6265),
('probably', 6229),
('run', 6222),
('real', 6168),
('always', 6156),
('might', 6150),
('getting', 6117),
('style', 6091),
```

```
('part', 6077),
('keep', 6046),
('5', 5984),
('fps', 5956),
('friends', 5875),
('full', 5830),
('sale', 5813),
('looking', 5807),
('anything', 5760),
('yes', 5743),
('10', 5716),
('controls', 5680),
('he', 5628),
('years', 5621),
('am', 5618),
('having', 5538),
('design', 5481),
('life', 5476),
('mechanics', 5472),
('done', 5433),
('cool', 5417),
('almost', 5384),
('his', 5327),
('pc', 5321),
('super', 5281),
('yet', 5276),
('shooter', 5186),
('kind', 5173),
('least', 5152),
('definitely', 5107),
('main', 5096),
('enemy', 5094),
('last', 5087),
('ai', 5074),
('put', 5070),
('version', 5065),
('less', 5064),
('come', 5059),
('id', 5041),
('gets', 5038),
('found', 5026),
('until', 5004),
('action', 4974),
('dead', 4943),
('boring', 4942),
('sometimes', 4908),
('big', 4901),
```

```
('single', 4874),
('trying', 4861),
('both', 4792),
('maybe', 4791),
('making', 4778),
('unique', 4771),
('instead', 4744),
('must', 4742),
('stuff', 4738),
('difficulty', 4706),
('seems', 4702),
('content', 4655),
('rather', 4622),
('art', 4603),
('looks', 4568),
('map', 4564),
('day', 4563),
('whole', 4550),
('especially', 4505),
('fan', 4502),
('used', 4495),
('next', 4493),
('said', 4441),
('either', 4440),
('able', 4440),
('campaign', 4438),
('between', 4423),
('away', 4417),
('puzzle', 4402),
('voice', 4289),
('sound', 4282),
('wont', 4256),
('already', 4251),
('fast', 4240),
('missions', 4238),
('classic', 4213),
('war', 4186),
('complete', 4179),
('adventure', 4170),
('decent', 4152),
('everyone', 4113),
('reason', 4103),
('maps', 4096),
('takes', 4085),
('space', 4068),
('enjoyed', 4068),
('issues', 4034),
```

```
('die', 4006),
('based', 4004),
('team', 3994),
('minutes', 3983),
('itself', 3975),
('problem', 3971),
('anyone', 3966),
('person', 3960),
('difficult', 3959),
('set', 3939),
('wait', 3937),
('online', 3911),
('high', 3858),
('yourself', 3854),
('highly', 3853),
('enjoyable', 3837),
('help', 3821),
('amount', 3817),
('dlc', 3811),
('thought', 3795),
('half', 3778),
('soundtrack', 3777),
('bugs', 3774),
('doing', 3742),
('beautiful', 3740),
('turn', 3739),
('using', 3736),
('small', 3727),
('side', 3726),
('lots', 3721),
('although', 3676),
('change', 3672),
('simply', 3663),
('fact', 3658),
('second', 3633),
('hell', 3626),
('let', 3608),
('place', 3599),
('bought', 3588),
('pay', 3569),
('often', 3536),
('left', 3534),
('early', 3514),
('items', 3511),
('pick', 3486),
('fight', 3469),
('build', 3469),
```

```
('fantastic', 3469),
('strategy', 3460),
('beat', 3430),
('challenging', 3416),
('basically', 3406),
('else', 3399),
('hit', 3364),
('community', 3352),
('save', 3351),
('during', 3349),
('later', 3341),
('add', 3331),
('idea', 3312),
('felt', 3284),
('certain', 3268),
('control', 3265),
('seen', 3256),
('completely', 3252),
('solid', 3248),
('someone', 3245),
('extremely', 3237),
('ending', 3234),
('move', 3226),
('perfect', 3224),
('huge', 3216),
('screen', 3210),
('absolutely', 3208),
('genre', 3192),
('top', 3184),
('title', 3178),
('open', 3176),
('mind', 3174),
('etc', 3172),
('call', 3171),
('mission', 3167),
('guns', 3152),
('seem', 3151),
('weapon', 3146),
('against', 3146),
('final', 3145),
('fighting', 3145),
('ok', 3140),
('sense', 3121),
('boss', 3121),
('cons', 3119),
('havent', 3112),
('expect', 3111),
```

```
('comes', 3110),
('gun', 3105),
('atmosphere', 3087),
('tell', 3064),
('plot', 3059),
('developers', 3047),
('points', 3042),
('annoying', 3031),
('building', 3024),
('shoot', 3023),
('  ing', 3021),
('win', 3015),
('click', 3014),
('myself', 3011),
('dark', 3008),
('acting', 2996),
('arent', 2994),
('rpg', 2993),
('oh', 2977),
('liked', 2974),
('slow', 2966),
('options', 2964),
('pros', 2946),
('wrong', 2942),
('works', 2941),
('excellent', 2932),
('needs', 2925),
('him', 2923),
('ill', 2921),
('took', 2916),
('mostly', 2910),
('others', 2893),
('hope', 2865),
('mods', 2860),
('mean', 2860),
('choose', 2853),
('hour', 2834),
('variety', 2827),
('easily', 2824),
('support', 2823),
('type', 2812),
('wasnt', 2810),
('terrible', 2809),
('challenge', 2808),
('came', 2800),
('three', 2791),
('battle', 2791),
```

```
('problems', 2786),
('year', 2784),
('unless', 2776),
('due', 2770),
('release', 2755),
('loved', 2748),
('lack', 2740),
('shooting', 2737),
('coop', 2737),
('running', 2736),
('choices', 2732),
('ways', 2721),
('opinion', 2708),
('wish', 2701),
('features', 2698),
('entire', 2691),
('funny', 2689),
('favorite', 2687),
('city', 2684),
('wanted', 2669),
('100', 2669),
('recommended', 2665),
('days', 2658),
('previous', 2657),
('understand', 2652),
('stop', 2641),
('youve', 2641),
('reviews', 2640),
('learn', 2632),
('special', 2631),
('become', 2627),
('fine', 2627),
('theyre', 2626),
('her', 2625),
('random', 2590),
('finish', 2587),
('zombies', 2586),
('base', 2584),
('video', 2580),
('goes', 2575),
('similar', 2572),
('buying', 2570),
('means', 2567),
('spend', 2567),
('20', 2563),
('devs', 2559),
('despite', 2555),
```

```
('fix', 2553),
('frustrating', 2553),
('achievements', 2550),
('attack', 2526),
('started', 2525),
('couple', 2524),
('along', 2521),
('feeling', 2516),
('past', 2511),
('order', 2510),
('fans', 2507),
('jump', 2507),
('together', 2504),
('gives', 2502),
('ones', 2493),
('stealth', 2490),
('guess', 2489),
('cannot', 2487),
('it’s', 2486),
('units', 2484),
('course', 2482),
('modern', 2475),
('low', 2448),
('horror', 2431),
('remember', 2428),
('alien', 2424),
('plays', 2422),
('cards', 2414),
('elements', 2413),
('lets', 2412),
('quickly', 2407),
('several', 2396),
('please', 2395),
('honestly', 2393),
('killing', 2392),
('fire', 2392),
('finally', 2380),
('released', 2380),
('job', 2367),
('example', 2361),
('read', 2360),
('tried', 2354),
('given', 2350),
('us', 2350),
('hot', 2350),
('access', 2349),
('hate', 2347),
```

```
('parts', 2347),
('man', 2344),
('worst', 2342),
('death', 2340),
('alot', 2336),
('matter', 2335),
('finished', 2331),
('fallout', 2331),
('wouldnt', 2329),
('spent', 2328),
('servers', 2319),
('doom', 2318),
('mod', 2317),
('  ', 2316),
('kinda', 2306),
('lost', 2299),
('engine', 2286),
('sort', 2280),
('sounds', 2272),
('longer', 2270),
('update', 2268),
('setting', 2268),
('quality', 2254),
('friend', 2249),
('6', 2243),
('actual', 2237),
('controller', 2234),
('shot', 2224),
('possible', 2224),
('810', 2222),
('repetitive', 2222),
('runs', 2196),
('unlock', 2190),
('walking', 2190),
('cheap', 2180),
('true', 2175),
('button', 2170),
('multiple', 2165),
('issue', 2161),
('went', 2156),
('mouse', 2150),
('added', 2146),
('value', 2142),
('skill', 2138),
('choice', 2128),
('except', 2127),
('fairly', 2127),
```

```
('910', 2121),
('care', 2121),
('total', 2121),
('sequel', 2117),
('storyline', 2117),
('star', 2113),
('simulator', 2109),
('rts', 2104),
('modes', 2098),
('room', 2094),
('gaming', 2092),
('progress', 2091),
('max', 2091),
('battles', 2088),
('close', 2079),
('ship', 2075),
('d', 2068),
('damage', 2067),
('areas', 2061),
('basic', 2060),
('seriously', 2044),
('behind', 2043),
('yeah', 2043),
('okay', 2041),
('usually', 2038),
('sucks', 2030),
('port', 2026),
('available', 2019),
('30', 2018),
('couldnt', 2015),
('future', 2001),
('check', 1997),
('number', 1996),
('settings', 1989),
('ability', 1985),
('line', 1979),
('coming', 1977),
('quick', 1974),
('literally', 1969),
('concept', 1965),
('truly', 1960),
('guy', 1958),
('replay', 1955),
('zombie', 1954),
('broken', 1952),
('stuck', 1948),
('effects', 1947),
```

```
('black', 1940),
('poor', 1931),
('large', 1928),
('dialogue', 1927),
('within', 1912),
('stupid', 1910),
('visuals', 1907),
('hand', 1900),
('damn', 1891),
('watch', 1890),
('u', 1890),
('worse', 1890),
('limited', 1887),
('platformer', 1884),
('option', 1878),
('writing', 1878),
('exactly', 1877),
('ago', 1877),
('power', 1874),
('quests', 1871),
('giving', 1868),
('match', 1868),
('otherwise', 1867),
('plenty', 1867),
('incredibly', 1859),
('certainly', 1854),
('plus', 1853),
('normal', 1852),
('survival', 1848),
('god', 1847),
('anyway', 1844),
('weird', 1842),
('health', 1838),
('waste', 1834),
('negative', 1822),
('head', 1811),
('personally', 1809),
('various', 1806),
('deal', 1805),
('7', 1802),
('create', 1801),
('compared', 1799),
('8', 1787),
('guys', 1785),
('taking', 1782),
('third', 1780),
('events', 1778),
```

```
('score', 1778),
('15', 1777),
('speed', 1777),
('somewhat', 1771),
('moments', 1770),
('unfortunately', 1770),
('totally', 1766),
('rest', 1759),
('area', 1759),
('wars', 1757),
('chance', 1753),
('forward', 1751),
('shooters', 1749),
('face', 1742),
('franchise', 1738),
('explore', 1732),
('killed', 1730),
('computer', 1728),
('fixed', 1726),
('throughout', 1723),
('show', 1722),
('extra', 1719),
('abilities', 1719),
('skills', 1718),
('satisfying', 1715),
('sonic', 1710),
('potential', 1710),
('under', 1704),
('believe', 1694),
('soon', 1687),
('alone', 1686),
('important', 1672),
('halflife', 1667),
('episode', 1667),
('cause', 1661),
('called', 1657),
('she', 1657),
('civ', 1654),
('deep', 1654),
('changes', 1652),
('designed', 1651),
('indie', 1650),
('general', 1648),
('age', 1645),
('desu', 1644),
('whats', 1639),
('tutorial', 1634),
```

```
('serious', 1632),
('xcom', 1629),
('moment', 1628),
('upgrade', 1624),
('server', 1622),
('waiting', 1614),
('mechanic', 1611),
('crap', 1603),
('season', 1603),
('piece', 1599),
('animations', 1597),
('cutscenes', 1592),
('purchase', 1590),
('tons', 1588),
('lose', 1584),
('depth', 1584),
('light', 1583),
('working', 1579),
('name', 1573),
('whatever', 1573),
('types', 1571),
('major', 1567),
('710', 1564),
('gone', 1563),
('linear', 1563),
('saying', 1554),
('case', 1549),
('horrible', 1547),
('figure', 1547),
('leave', 1544),
('duty', 1543),
('home', 1542),
('our', 1541),
('ships', 1539),
('cod', 1538),
('menu', 1536),
('movement', 1535),
('clear', 1532),
('beginning', 1528),
('needed', 1526),
('visual', 1523),
('3d', 1522),
('keyboard', 1522),
('hidden', 1521),
('gave', 1520),
('realistic', 1518),
('crashes', 1518),
```

```
('card', 1513),
('force', 1512),
('casual', 1510),
('offer', 1507),
('walk', 1506),
('becomes', 1505),
('constantly', 1504),
('playthrough', 1501),
('bored', 1501),
('cover', 1497),
('red', 1497),
('across', 1493),
('sniper', 1485),
('hold', 1482),
('adds', 1481),
('edit', 1481),
('feature', 1480),
('fantasy', 1480),
('arcade', 1477),
('strong', 1470),
('happy', 1467),
('thanks', 1467),
('thinking', 1466),
('awful', 1462),
('key', 1462),
('rating', 1459),
('updates', 1455),
('moving', 1452),
('impossible', 1450),
('development', 1448),
('note', 1440),
('bring', 1440),
('break', 1440),
('bosses', 1438),
('miss', 1436),
('developer', 1435),
('gold', 1435),
('starts', 1434),
('ha', 1432),
('classes', 1428),
('slightly', 1427),
('addition', 1423),
('higher', 1423),
('paid', 1420),
('unlike', 1418),
('edition', 1418),
('follow', 1417),
```

```
('expected', 1414),
('crazy', 1414),
('environment', 1414),
('missing', 1413),
('upgrades', 1413),
('fair', 1411),
('further', 1409),
('aliens', 1408),
('titles', 1405),
('quest', 1405),
('house', 1404),
('happen', 1399),
('entertaining', 1399),
('addictive', 1396),
('movie', 1395),
('easier', 1392),
('harder', 1392),
('aspect', 1389),
('ammo', 1388),
('attention', 1387),
('focus', 1387),
('school', 1386),
('current', 1385),
('near', 1385),
('keeps', 1385),
('blood', 1384),
('anymore', 1380),
('ingame', 1379),
('core', 1377),
('effect', 1377),
('above', 1377),
('fights', 1374),
('avoid', 1371),
('turns', 1371),
('magic', 1371),
('massive', 1370),
('humor', 1370),
('expansion', 1368),
('epic', 1363),
('class', 1360),
('moves', 1359),
('sadly', 1358),
('seconds', 1358),
('learning', 1352),
('stay', 1343),
('mention', 1343),
('none', 1343),
```

```
('gonna', 1342),
('wow', 1339),
('monsters', 1330),
('grind', 1327),
('physics', 1327),
('telltale', 1326),
('beyond', 1322),
('themselves', 1321),
('improved', 1318),
('sad', 1315),
('including', 1315),
('told', 1315),
('attacks', 1315),
('standard', 1315),
('upon', 1314),
('balance', 1311),
('scenes', 1311),
('fear', 1309),
('interested', 1309),
('pass', 1309),
('bullet', 1309),
('allows', 1308),
('reading', 1307),
('windows', 1304),
('smooth', 1299),
('ii', 1299),
('terms', 1298),
('whether', 1297),
('replayability', 1297),
('minor', 1295),
('talk', 1295),
('complex', 1293),
('straight', 1291),
('supposed', 1289),
('saw', 1286),
('narrative', 1286),
('perhaps', 1286),
('2d', 1286),
('offers', 1285),
('likely', 1285),
('aside', 1284),
('rate', 1274),
('stories', 1274),
('decided', 1274),
('starting', 1270),
('bunch', 1270),
('12', 1270),
```

```
('camera', 1270),
('hitman', 1268),
('car', 1267),
('biggest', 1266),
('nearly', 1266),
('singleplayer', 1265),
('wonderful', 1264),
('today', 1263),
('requires', 1258),
('happens', 1258),
('tactical', 1258),
('evil', 1257),
('four', 1257),
('challenges', 1255),
('chapter', 1252),
('changed', 1252),
('fully', 1250),
('live', 1246),
('seeing', 1246),
('personal', 1246),
('finding', 1246),
('perfectly', 1240),
('paced', 1239),
('playable', 1238),
('considering', 1237),
('cute', 1233),
('lol', 1231),
('stand', 1229),
('resources', 1229),
('party', 1226),
('master', 1225),
('entirely', 1224),
('bug', 1222),
('towards', 1221),
('brilliant', 1220),
('solve', 1219),
('melee', 1218),
('average', 1218),
('50', 1217),
('continue', 1214),
('direction', 1206),
('universe', 1204),
('cut', 1202),
('eventually', 1201),
('wall', 1197),
('reccomend', 1197),
('require', 1197),
```

```
 ('step', 1194),
 ('item', 1192),
 ('objects', 1192),
 ('heard', 1190),
 ('planet', 1188),
 ('stick', 1187),
 ('heavy', 1187),
 ('positive', 1186)]
```

top 10 words and their frequencies

[43]:
```
#top 10 words and frequencies
wordFreq = {k : v /totalCount for k,v in wordCount.items()}
mostCommon1000_wf = sorted(wordFreq.items(),key=lambda v: v[1] /␣
 ↪totalCount,reverse=True)[:1000]


top10 = mostCommon1000_wf[:10]
top10
```

[43]:
```
[('the', 0.047446967093177694),
 ('and', 0.02767203214144606),
 ('a', 0.02660860784726279),
 ('to', 0.025429658351204455),
 ('game', 0.021376431377725155),
 ('of', 0.019797325582864286),
 ('is', 0.018157930617794103),
 ('you', 0.017479764576017718),
 ('i', 0.01707202856939985),
 ('it', 0.016637545777732472)]
```

[44]:
```
mostCommon1000words = [w for w,f in mostCommon1000_wf]
```

[45]:
```
len(mostCommon1000words)
```

[45]:
```
1000
```

Task 7

[46]:
```
wordId = dict(zip(mostCommon1000words, range(len(mostCommon1000words))))
wordSet = set(mostCommon1000words)



def feature(d):
    feat = [0]*len(mostCommon1000words)
    r = ''.join([c for c in d['text'].lower() if not c in punctuation])
    for w in r.split():
```

```
        if w in mostCommon1000words:
            feat[wordId[w]] += 1
    feat.append(1) #offset
    return feat
```

[47]:
```
#(for reference) almost two minutes to run
Xtrain = [feature(d) for d in train]
Xlil = lil_matrix(Xtrain)
ytrain =  [d['genreID'] for d in train]
```

[ ]:
```

```

[48]:
```
from sklearn.linear_model import LogisticRegression
```

[49]:
```
clf = LogisticRegression(max_iter = 2000)
clf.fit(Xlil,ytrain)
```

[49]:
```
LogisticRegression(max_iter=2000)
```

[50]:
```
Xvalid = [feature(d) for d in validation]
```

[51]:
```
pred = clf.predict(Xvalid)
```

[52]:
```
correct = pred == yvalid
print("accuracy: ", sum(correct) / len(correct))
```

```
accuracy:  0.6723
```

Task 8

observation: as dict size increases so does accuracy, the classifier ending up improving by .05% accuracy

unfortunately due to time concerns and nearing the homework deadline I didn't get to test dictionary sizes > 5000

I only tested sizes 500, 800, 1000, 2000, 3000, 4000, and 5000 for c values [10-2, .1,1,10,100], and found that a c value of 1 usually results in the highest accuracy**

i will use a dictionary size of 5000 and C value of 1 in the improved classifier

[53]:
```
mostCommon5000_wf = sorted(wordFreq.items(),key=lambda v: v[1] /␣
 ↪totalCount,reverse=True)[:5000]
mostCommon5000words = [w for w,f in mostCommon5000_wf]
wordId = dict(zip(mostCommon5000words, range(len(mostCommon5000words))))
wordSet = set(mostCommon5000words)
```

[54]:
```
def feature(d):
    feat = [0]*len(mostCommon5000words)
    r = ''.join([c for c in d['text'].lower() if not c in punctuation])
    for w in r.split():
        if w in mostCommon5000words:
            feat[wordId[w]] += 1
    feat.append(1) #offset
    return feat
```

[55]:
```
Xtrain = [feature(d) for d in train]
Xlil = lil_matrix(Xtrain)
ytrain =  [d['genreID'] for d in train]
Xvalid = [feature(d) for d in validation]
```

[56]:
```
clf = LogisticRegression(max_iter = 8000)
clf.fit(Xlil,ytrain)
pred = clf.predict(Xvalid)
```

[57]:
```
correct = pred == yvalid
print("accuracy: ", sum(correct) / len(correct))
```

```
accuracy:  0.7294
```

[ ]:
```
```

[58]:
```
### Category prediction baseline: Just consider some of the most common words␣
 ↪from each category

catDict = {
  "Action": 0,
  "Strategy": 1,
  "RPG": 2,
  "Adventure": 3,
  "Sport": 4
}

predictions = open("predictions_Category.txt", 'w')
predictions.write("userID-reviewID,prediction\n")
for u,_,d in readJSON("data/test_Category.json.gz"):
    cat = clf.predict([feature(d)])
    predictions.write(u + '-' + d['reviewID'] + "," + str(cat[0]) + "\n")

predictions.close()
```

[ ]:
```
# C = [.1,1,10,100]
# ps =[]
# for c in C:
```

```
#     clf = LogisticRegression(max_iter = 8000, C = c)
#     clf.fit(Xlil,ytrain)
#     pred = clf.predict(Xvalid)
#     ps.append(pred)
```

[ ]:
```
# correct = pred == yvalid
# print("accuracy: ", sum(correct) / len(correct))
```

[ ]:
```
# acc_list =[]
# for p in ps:
#     correct = p == yvalid
#     acc_list.append(sum(correct)/len(correct))
```

[ ]:
```
#dict size 2000
#max_iter = 6000
#C = [10**-2,.1,1,10,100]
# acc_list  = [0.6886, 0.7003, 0.7017, 0.7015, 0.7011]
```

[ ]:
```
#dict size 3000
#max_iter = 6000
#C = [.1,1,10,100]
# acc_list = [0.7116, 0.7146, 0.713, 0.7128]
```

[ ]:
```
#dict size 5000
#max_iter = 6000
#C = [.1,1,10,100]
# acc_list = [0.7277, 0.7294, 0.7282, 0.7274]
```

[ ]:
```
```