A10 Submission

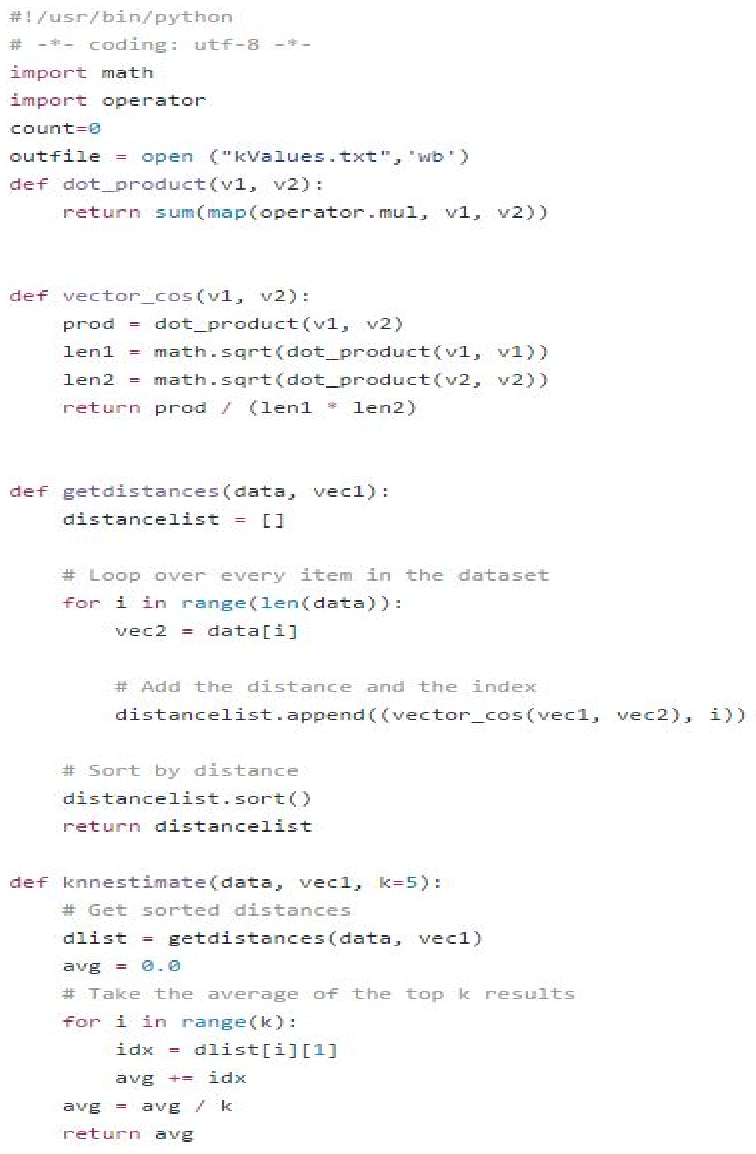
Breon Day

Cs432

Spring 2017

1.Using the data from A8: - Consider each row in the blog-term matrix as a 1000 dimension vector, corresponding to a blog. - From chapter 8, replace numpredict.euclidean() with cosine as the distance metric. In other words, you'll be computing the cosine between vectors of 1000 dimensions. - Use knnestimate() to compute the nearest neighbors for both: http://f-measure.blogspot.com/ http://ws-dl.blogspot.com/ for k={1,2,5,10,20}.
A large portion of this assignment was given to use either through the pc book or slides the cosine function i found at
http://stackoverflow.com/questions/18424228/cosine-similarity-between-2-number-lists

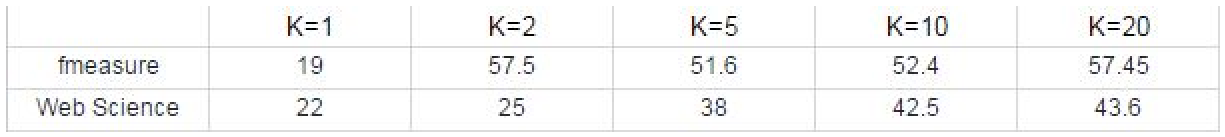
Code Portion:



```
#!/usr/bin/python
# -*- coding: utf-8 -*-
import math
import operator
count=0
outfile = open ("kValues.txt",'wb')
def dot_product(v1, v2):
    return sum(map(operator.mul, v1, v2))


def vector_cos(v1, v2):
    prod = dot_product(v1, v2)
    len1 = math.sqrt(dot_product(v1, v1))
    len2 = math.sqrt(dot_product(v2, v2))
    return prod / (len1 * len2)


def getdistances(data, vec1):
    distancelist = []

    # Loop over every item in the dataset
    for i in range(len(data)):
        vec2 = data[i]

        # Add the distance and the index
        distancelist.append((vector_cos(vec1, vec2), i))

    # Sort by distance
    distancelist.sort()
    return distancelist

def knnestimate(data, vec1, k=5):
    # Get sorted distances
    dlist = getdistances(data, vec1)
    avg = 0.0
    # Take the average of the top k results
    for i in range(k):
        idx = dlist[i][1]
        avg += idx
    avg = avg / k
    return avg
```

Full code:

https://github.com/BreonDay/cs532-s17/blob/master/Submissions/A10/Q1/Q1.py
knnestimates*:



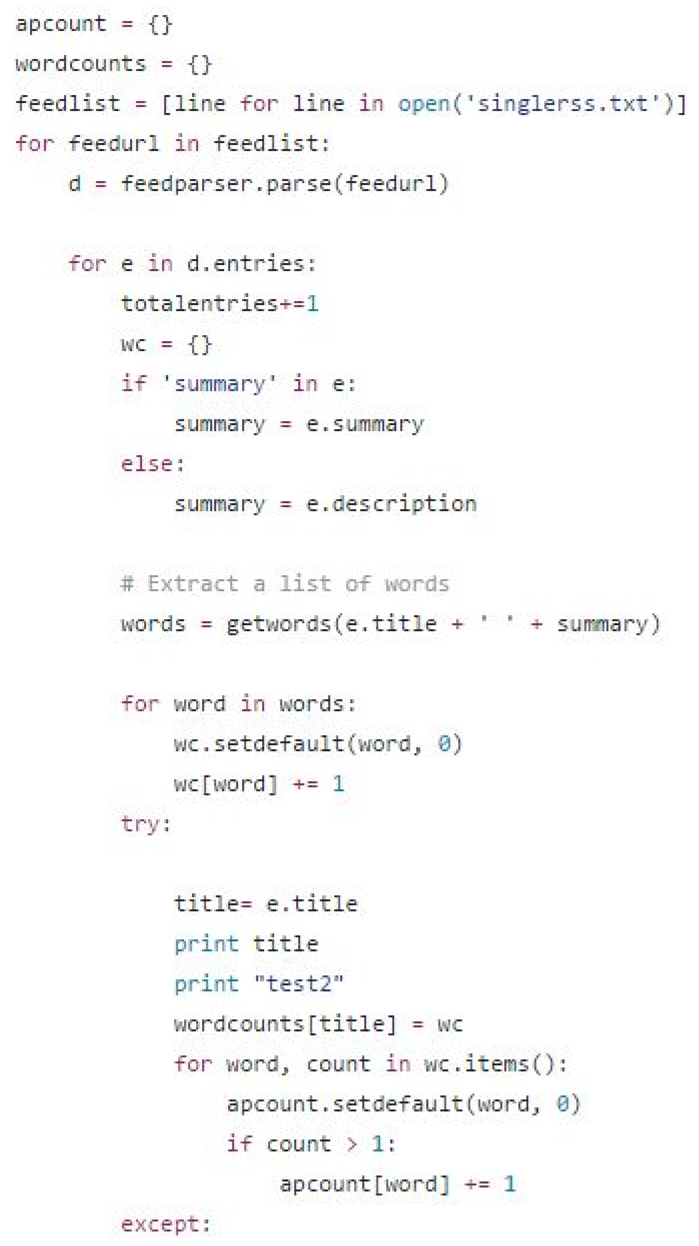
	K=1	K=2	K=5	K=10	K=20
fmeasure	19	57.5	51.6	52.4	57.45
Web Science	22	25	38	42.5	43.6

*note the data was not made in a spreadsheet just placed there for a better view
See:
https://github.com/BreonDay/cs532-s17/blob/master/Submissions/A10/Q1/kValues.txt

rerun A9, Q2 but this time using LIBSVM. If you have n categories, you'll have to run it n times. For example, if you're classifying music and have the categories: metal, electronic, ambient, folk, hip-hop, pop you'll have to classify things as: metal / not-metal electronic / not-electronic ambient / not-ambient etc. Use the 1000 term vectors describing each blog as the features, and your mannally assigned classifications as the true values. Use 10-fold cross-validation (as per slide 46, which shows 4-fold cross-validation) and report the percentage correct for each of your categories.

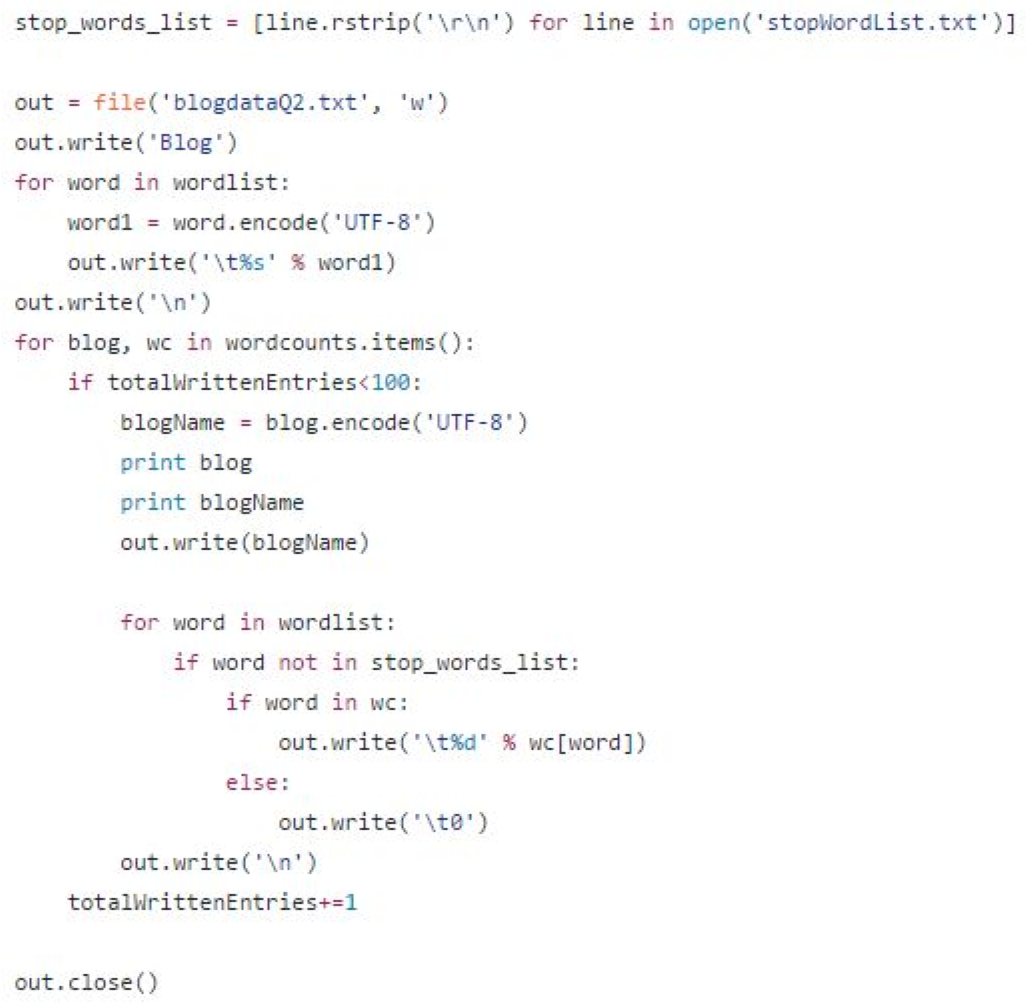
For this question it honestly took my longer to get svm working then the actual problem solution eventually i just gave up downloaded the entire svm project and added my projects into the python subsection of it they ran well from there
https://github.com/BreonDay/cs532-s17/tree/master/Submissions/A10/libsvm-3.22/libsvm-3.22/python

Modifying generate feed vector i was able to get it to run by entries instead of blogs



```python
apcount = {}
wordcounts = {}
feedlist = [line for line in open('singlerss.txt')]
for feedurl in feedlist:
    d = feedparser.parse(feedurl)

    for e in d.entries:
        totalentries+=1
        wc = {}
        if 'summary' in e:
            summary = e.summary
        else:
            summary = e.description

        # Extract a list of words
        words = getwords(e.title + ' ' + summary)

        for word in words:
            wc.setdefault(word, 0)
            wc[word] += 1
        try:

            title= e.title
            print title
            print "test2"
            wordcounts[title] = wc
            for word, count in wc.items():
                apcount.setdefault(word, 0)
                if count > 1:
                    apcount[word] += 1
        except:
```

I then limited it to 100



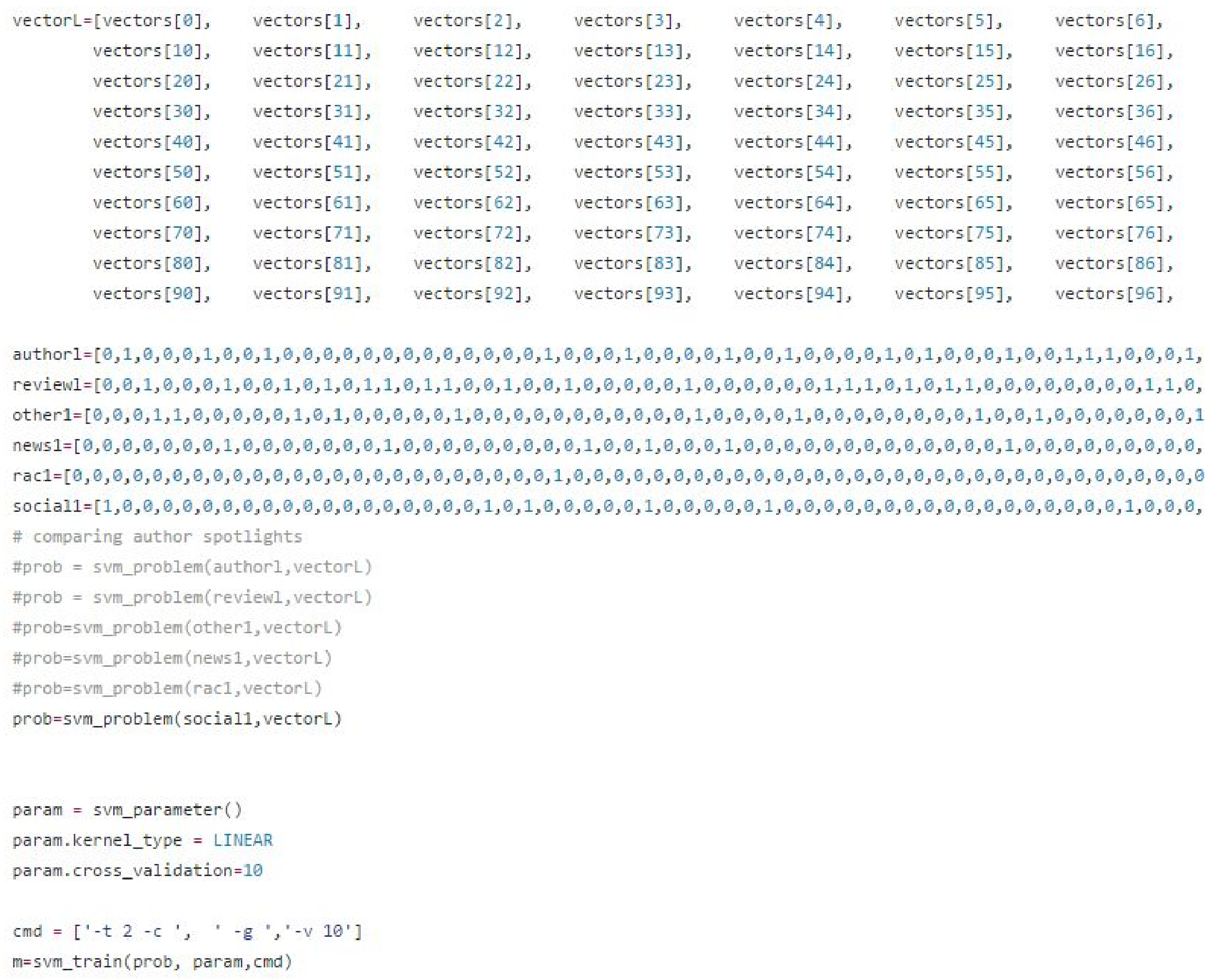
```
stop_words_list = [line.rstrip('\r\n') for line in open('stopWordList.txt')]

out = file('blogdataQ2.txt', 'w')
out.write('Blog')
for word in wordlist:
    word1 = word.encode('UTF-8')
    out.write('\t%s' % word1)
out.write('\n')
for blog, wc in wordcounts.items():
    if totalWrittenEntries<100:
        blogName = blog.encode('UTF-8')
        print blog
        print blogName
        out.write(blogName)

        for word in wordlist:
            if word not in stop_words_list:
                if word in wc:
                    out.write('\t%d' % wc[word])
                else:
                    out.write('\t0')
        out.write('\n')
    totalWrittenEntries+=1

out.close()
```

With the 100 entry 1000 term blog i was now ready to process it
Unfortunately this created a new ground truth as it had been several days since i last ran this
https://github.com/BreonDay/cs532-s17/blob/master/Submissions/A10/Q2/A10%20Ground%20truth%20-%20Sheet1.pdf

Using the ground truth i created a vector of 0,’s and 1’s 100 total and 100 total word counts i created 1 vector of length 100 and 100 vectors of length 1000
Justin was very helpful in figuring bugs in my code in relation to inaccuracy

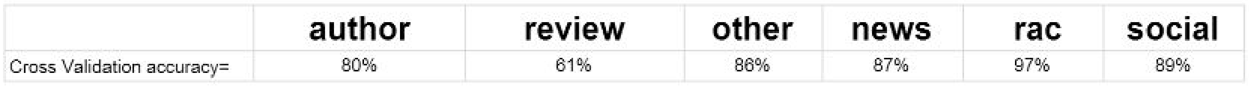
Code

```
# -*- coding: utf-8 -*-
from svm import *
from svmutil import *
from svm import __all__ as svm_all
out=open("blogNames.txt",'wb')



#
TrainingLabel=[]

lines=[]
for line in open("blogdataQ2.txt"):
    lines.append(line)
#seperates the labels data and
blogNames=[]
vectors=[]
words=lines[0].strip().split('\t')[1:]
for line in lines[1:]:
    names=line.strip().split('\t')
    blogNames.append(names[0])
    vectors.append([float(x) for x in names[1:]])
blogNames=blogNames
# print blogNames
# print vectors
# print len(vectors)
# print len([[1,0,1], [-1,0,-1]])
for name in blogNames:
    out.write(name)
    out.write('\n')
# param = svm_parameter()
# svm_model.predict = lambda self, x: svm_predict([0], [x], self)[0][0]
#
# #prob = svm_problem([1,-1], vectors)
# #prob = svm_problem([1,-1], [[1,0,1], [-1,0,-1]])
# prob = svm_problem([1,2,3,4,5], [[1,0,1], [-1,0,-1],[1,0,1], [-1,0,-1],[-1,0,-1]])
# #prob = svm_problem([1,-1],[vectors,vectors])
# param = svm_parameter()
# param.kernel_type = LINEAR
# param.C = 10
# param.cross_validation=10
#
#
```



```
vectorL=[vectors[0],    vectors[1],     vectors[2],     vectors[3],     vectors[4],     vectors[5],     vectors[6],
        vectors[10],    vectors[11],    vectors[12],    vectors[13],    vectors[14],    vectors[15],    vectors[16],
        vectors[20],    vectors[21],    vectors[22],    vectors[23],    vectors[24],    vectors[25],    vectors[26],
        vectors[30],    vectors[31],    vectors[32],    vectors[33],    vectors[34],    vectors[35],    vectors[36],
        vectors[40],    vectors[41],    vectors[42],    vectors[43],    vectors[44],    vectors[45],    vectors[46],
        vectors[50],    vectors[51],    vectors[52],    vectors[53],    vectors[54],    vectors[55],    vectors[56],
        vectors[60],    vectors[61],    vectors[62],    vectors[63],    vectors[64],    vectors[65],    vectors[65],
        vectors[70],    vectors[71],    vectors[72],    vectors[73],    vectors[74],    vectors[75],    vectors[76],
        vectors[80],    vectors[81],    vectors[82],    vectors[83],    vectors[84],    vectors[85],    vectors[86],
        vectors[90],    vectors[91],    vectors[92],    vectors[93],    vectors[94],    vectors[95],    vectors[96],

author1=[0,1,0,0,0,1,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,1,0,0,0,0,1,0,0,1,0,0,0,0,1,0,1,0,0,0,1,0,0,1,1,1,0,0,0,1,
reviewl=[0,0,1,0,0,0,1,0,0,1,0,1,0,1,1,0,1,1,0,0,1,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,1,1,1,0,1,0,1,1,0,0,0,0,0,0,0,0,1,1,0,
other1=[0,0,0,1,1,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,1,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,0,0,0,0,1
news1=[0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,1,0,0,1,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,
rac1=[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0
social1=[1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,1,0,0,0,0,0,1,0,0,0,0,0,1,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,1,0,0,0,
# comparing author spotlights
#prob = svm_problem(authorl,vectorL)
#prob = svm_problem(reviewl,vectorL)
#prob=svm_problem(other1,vectorL)
#prob=svm_problem(news1,vectorL)
#prob=svm_problem(rac1,vectorL)
prob=svm_problem(social1,vectorL)


param = svm_parameter()
param.kernel_type = LINEAR
param.cross_validation=10

cmd = ['-t 2 -c ',  ' -g ','-v 10']
m=svm_train(prob, param,cmd)
```

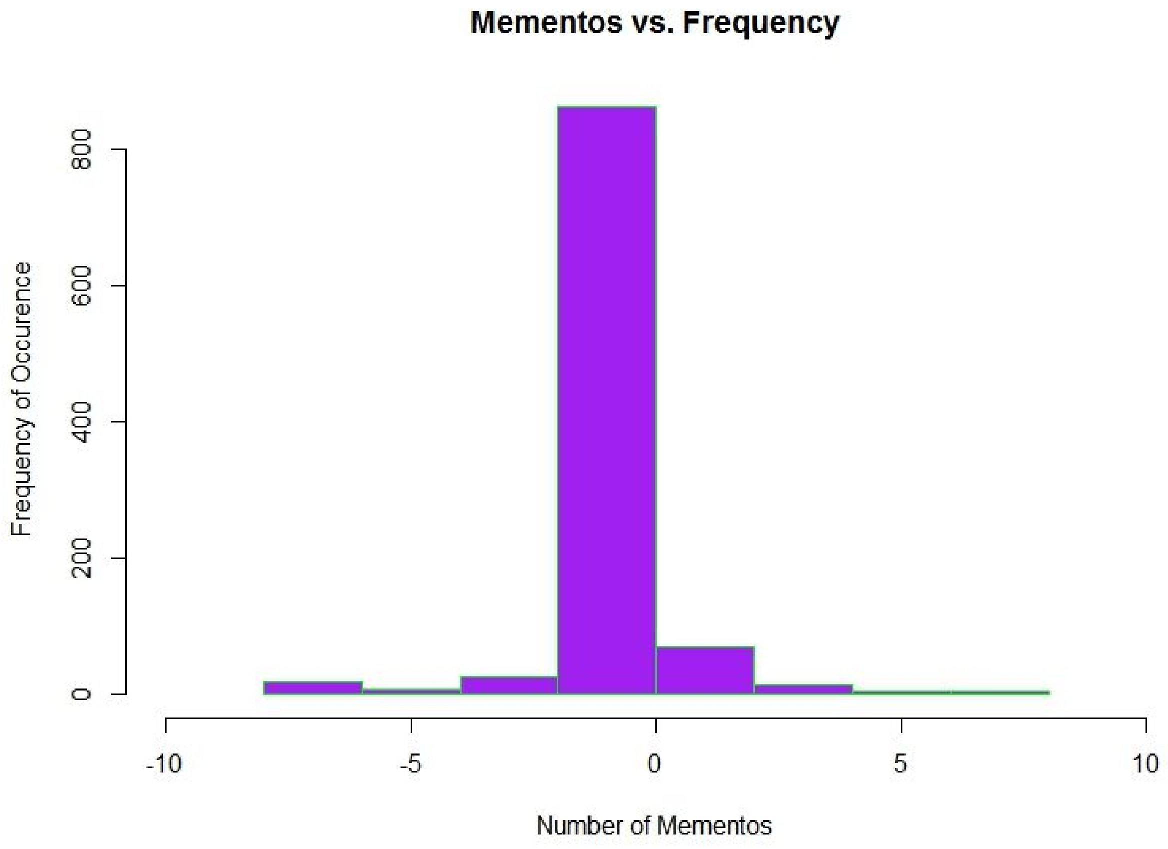
Feeding those into svm param svm problem and setting the cross validation tag i was able to produce the following output



	author	**review**	**other**	**news**	**rac**	**social**
Cross Validation accuracy=	80%	61%	86%	87%	97%	89%

3. Re-download the 1000 TimeMaps from A2, Q2. Create a graph where the x-axis represents the 1000 TimeMaps. If a TimeMap has "shrunk", it will have a negative value below the x-axis corresponding to the size difference between the two TimeMaps. If it has stayed the same, it will have a "0" value. If it has grown, the value will be positive and correspond to the increase in size between the two TimeMaps. As always, upload all the TimeMap data. If the A2 github has the original TimeMaps, then you can just point to where they are in the report.

This section i would not have been able to complete without the help of ross and michelle graph made in r studios



```
histData <- read.table("C:/CS432/A10/Q3/differenceTimeMaps")
hist(histData$V1, xlab='Number of Mementos', xlim = range(c(-10,10)),
     ylab='Frequency of Occurence', main='Mementos vs. Frequency',
     ty =4, col ='pink', border = 'red')
```

https://github.com/rreelachart/cs532-s17/tree/master/submissions/assignment%202