Assignment 10

CS532-s16: Web Sciences

Spring 2016

John Berlin

Generated on April 30, 2016

Index

List of Tables

Listings

Question 1

1. Using the data from A8:

- Consider each row in the blog-term matrix as a 500 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 500 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}.

Answer

As I did in A8, both the stemmed and non-stemmed words for the blogs were used. The cosign distance used was provided by the SciPy library. Also from A8 I used the `processData.py` file listing 2 with the relavant methods from `clusters.py` required for this assignment. The code used to generate the results can be seen in listing 1, as usual please consult the comments in the listing for further details.

The k nearest neighbors for F-Measure can be seen in table 1 for non-stemmed and table 2 for stemmed. The results for WSDL can be seen in table 3 for non-stemmed and table 4 for stemmed. Please not that for each k value the previous kth nearest neighbors were included, so I removed them and only include the most unique values for each k value. This means that k=20 only has ten entries, all values for k=10 were apart of the output and can be seen in the k=10 entry.

Just like in A8 the WSDL first nearest neighbor was Dr. Nelsons blog F-Measure but unlike A8 F-Measure had more music blogs in its nearest neighbors.

k	nearest neighbors
1	FOLK IS NOT HAPPY
2	Eli Jace — The Mind Is A Terrible Thing To Paste
5	DaveCromwell Writes, The Kids Are Coming Up From Behind, Blog Name Pending
10	The World's First Internet Baby, Did Not Chart, mouxlaloulouda, I Hate The 90s, turnitup!
20	., I/LOVE/TOTAL/DESTRUCTION, Tremble Under Boom Lights, The Night Mail, Encore, Flatbasset, The Power of Independent Trucking, Steel City Rust, from a voice plantation, Cherry Area

Table 1: F-Measure knearest nonstemmed

k	nearest neighbors
1	FOLK IS NOT HAPPY
2	Floorshime Zipper Boots
5	mouxlaloulouda, DaveCromwell Writes, MTJR RANTS & RAVES ON MUSIC
10	Encore, Samtastic! Review, I/LOVE/TOTAL/DESTRUCTION, I Hate The 90s, The Kids Are Coming Up From Behind
20	The World's First Internet Baby, Mile In Mine, Blog Name Pending, from a voice plantation, The Campus Buzz on WSOU, The Night Mail, Did Not Chart, Tremble Under Boom Lights, Interstellar Radio Shower, .

Table 2: F-Measure knearest stemmed

k	nearest neighbors
1	F-Measure
2	The Power of Independent Trucking
5	I/LOVE/TOTAL/DESTRUCTION, ., MAGGOT CAVIAR
10	Swinging Singles Club, mouxlaloulouda, Steel City Rust, The World's First Internet Baby, Cherry Area
20	Encore, Flatbasset, DaveCromwell Writes, KiDCHAIR, MTJR RANTS & RAVES ON MUSIC, Blog Name Pending, I Hate The 90s, The Girl at the Rock Show, Eli Jace — The Mind Is A Terrible Thing To Paste, Tremble Under Boom Lights

Table 3: WSDL knearest nonstemmed

k	nearest neighbors
1	F-Measure
2	Flatbasset
5	Tremble Under Boom Lights, I Hate The 90s, Riley Haas' blog
10	., mouxlaloulouda, MTJR RANTS & RAVES ON MUSIC, The Listening Ear, Cherry Area
20	What Am I Doing?, Karl Drinkwater, My Name Is Blue Canary, The World's First Internet Baby, But She's Not Stupid, Samtastic! Review, Pithy Title Here, I/LOVE/TOTAL/DESTRUCTION, Did Not Chart, The Jeopardy of Contentment

Table 4: WSDL knearest stemmed

Question 2

```
2.  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 500 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.
```

Answer

As before in A9 the percentage correct is low. Using genre as a label only results in a maximum of a 58% for Pop/Electronic/Hip-Hop accuracy. But when using Dr. Nelsons labeling of his own blog we get a maximum of 66% accuracy for lp review. I have included the metrics requested in A9 along with the accuracy and they had only a marginal increase. I still hold to my conclusion from A9 that using the words alone is not enough to classify these blog posts.

Genre 10 fold accuracy non stemmed can be seen in table 5 and stemmed in table 6. Dr. Nelson label accuracy non stemmed can be seen in table 7 and stemmed in table 8. Genre preivous metrics can be seen in table 9 and table 10 for non stemmed and stemmed. Dr. Nelson labels preivous metrics can be seen in table 11 and table 12 for stemmed and non stemmed.

label	mean accuracy
Rock, not Rock	49.24%
R&B/Jazz/Mowtown/Country/Other, not R&B/Jazz/Mowtown/Country/Other	56.95%
Pop/Electronic/Hip-Hop, not Pop/Electronic/Hip-Hop	58.39%
Metal/Punk/Hardcore, not Metal/Punk/Hardcore	58.42%
Indie Rock/Alternative, not Indie Rock/Alternative	52.87%

Table 5: Genre Ten Fold Non Stemmed Accuracy

label	mean accuracy
Rock, not Rock	46.29%
R&B/Jazz/Mowtown/Country/Other, not R&B/Jazz/Mowtown/Country/Other	54.84%
Pop/Electronic/Hip-Hop, not Pop/Electronic/Hip-Hop	54.34%
Metal/Punk/Hardcore, not Metal/Punk/Hardcore	55.32%
Indie Rock/Alternative, not Indie Rock/Alternative	50.92%

Table 6: Genre Ten Fold Stemmed Accuracy

label	mean accuracy
the song remains the same, not the song remains the same	56.42%
concert/spotlight, not concert/spotlight	54.37%
lp review, not lp review	66.05%
forgotten song, not forgotten song	46.16%

Table 7: Dr. Nelson Ten Fold Non Stemmed

label	mean accuracy
the song remains the same, not the song remains the same	53.32%
concert/spotlight, not concert/spotlight	49.26%
lp review, not lp review	64.05%
forgotten song, not forgotten song	44.16%

Table 8: Dr. NelsonTen Fold Stemmed

label	precision	recall	f1
Rock	0.50297	0.479219	0.454893
R&B/Jazz/Mowtown/Country/Other	0.508877	0.518228	0.449765
Pop/Electronic/Hip-Hop	0.494767	0.481555	0.446287
Metal/Punk/Hardcore	0.493387	0.492323	0.431816
Indie Rock/Alternative	0.494432	0.48864	0.472867

Table 9: Genre Ten Fold Non Stemmed

label	precision	recall	f1
Rock	0.459143	0.444075	0.43609
R&B/Jazz/Mowtown/Country/Other	0.524861	0.553098	0.458025
Pop/Electronic/Hip-Hop	0.500927	0.508585	0.440113
Metal/Punk/Hardcore	0.515069	0.534176	0.443302
Indie Rock/Alternative	0.497443	0.480556	0.47425

Table 10: Genre Ten Fold Stemmed

label	precision	recall	f1
the song remains the same	0.506409	0.459346	0.434141
concert/spotlight	0.455613	0.431168	0.399722
lp review	0.629662	0.636321	0.61821
forgotten song	0.408316	0.394083	0.382929

Table 11: Dr. Nelson Ten Fold Non Stemmed

label	precision	recall	f1
the song remains the same	0.506409	0.459346	0.434141
concert/spotlight	0.455613	0.431168	0.399722
lp review	0.629662	0.636321	0.61821
forgotten song	0.408316	0.394083	0.382929

Table 12: Dr. NelsonTen Fold Stemmed

```
import json
import re
import statistics
from collections import defaultdict

from sklearn.cross_validation import KFold
from sklearn.metrics import f1_score, precision_score, recall_score
from tabulate import tabulate

from numpredict import knnestimate
from processData import consume_all
from svmutil import svm_problem, svm_predict, svm_parameter,
    svm_train
from processData import generate_blogfile, generate_blogfile_stem

# regex to capture the self labeled topic of the blog post
findClass = re.compile("^.+\\((.+)\\)$")

# extract the artist portion. Capture everything until our negative
     look ahead says we have a space - space "
artistsExtractor = re.compile("^(?!\s\-\s\")([a-zA-Z0-9.&\-']+\s
    (?:[a-zA-Z0-9.&\-']+\s)*)")

# regex to remove / from my labels for nice file names
rem = re.compile("[^a-z]")

# load the A9 artist to genre file
with open("datafiles/artistsToGenre.json", "r") as agi:
    aTg = json.load(agi)


# read file gotten from clusters.py in A8
def readfile(filename):
    with open(filename, "r") as o:
        lines = [line.rstrip("\n") for line in o]
    colnames = lines[0].strip().split('\t')[1:]
    rownames = []
    data = []
    for line in lines[1:]:
        p = line.strip().split('\t')
        rownames.append(p[0])
        data.append({'input': [float(x) for x in p[1:]], 'result':
    p[0]})
    return rownames, colnames, data

# using the blogtop500.txt file from A8 do knn on it
def nonstemmeda8():
    # load the data
    blognames, words, data = readfile("datafiles/blogtop500.txt")
    # find out where F-Measure is
    fmeasure_idk = blognames.index('F-Measure')
    # find out where wsdl is
    wsdl_idk = blognames.index('Web Science and Digital Libraries
    Research Group')
    # remove fmeasure from the data as we do not want to include it
    takeout = data.pop(fmeasure_idk)
    print("http://f-measure.blogspot.com/")
```

```
    # hold the results of knn for reporting
    mout = []
    klast = None
    # do knn
    for k in [1, 2, 5, 10, 20]:
        # for reporting I am shortening the output to only include the new data
        # successive knn runs return the same data for previous k values
        # ie k=2 [a,b] then k=5 would be [a,b,c,d,e] I only want k=2[a,b] and k=5[c,d,e]
        if k != 1:
            val = knnestimate(data, data[fmeasure_idk]['input'], k)[klast:]
        else:
            val = knnestimate(data, data[fmeasure_idk]['input'], k)
        klast = k
        mout.append(["%d" % k, ", ".join(val)])
        print("The %d nearest neighbors are:" % k, knnestimate(data, data[fmeasure_idk]['input'], k))
    # write out a table in latex
    headers = ['k', 'nearest neighbors']
    with open("tables/fmeasure-knearest-nonstemmed.text", "w+") as out:
        out.write(tabulate(mout, headers=headers, tablefmt="latex"))
    # do the same for wsdl
    data.insert(fmeasure_idk, takeout)
    print("http://ws-dl.blogspot.com/")
    data.pop(wsdl_idk)
    mout.clear()
    for k in [1, 2, 5, 10, 20]:
        if k != 1:
            val = knnestimate(data, data[fmeasure_idk]['input'], k)[klast:]
        else:
            val = knnestimate(data, data[fmeasure_idk]['input'], k)
        klast = k
        mout.append(["%d" % k, ", ".join(val)])
        print("The %d nearest neighbors are:" % k, knnestimate(data, data[fmeasure_idk]['input'], k))
    headers = ['k', 'nearest neighbors']
    with open("tables/wsdl-knearest-nonstemmed.text", "w+") as out:
        out.write(tabulate(mout, headers=headers, tablefmt="latex"))


def stemmeda8():
    # behave exactly as its non-stemmed counterpart except using the stemmed file
    blognames, words, data = readfile("datafiles/blogtop500_stemmed.txt")
    fmeasure_idk = blognames.index('F-Measure')
    wsdl_idk = blognames.index('Web Science and Digital Libraries Research Group')
    takeout = data.pop(fmeasure_idk)
    mout = []
```

```
97      klast = None
98      print("http://f-measure.blogspot.com/")
99      for k in [1, 2, 5, 10, 20]:
100         if k != 1:
101             val = knnestimate(data, data[fmeasure_idk]['input'], k)
    [klast:]
102         else:
103             val = knnestimate(data, data[fmeasure_idk]['input'], k)
104         klast = k
105         mout.append(["%d"%k, ", ".join(val)])
106         print("The %d nearest neighbors are:" % k, knnestimate(data
    , data[fmeasure_idk]['input'], k))
107     headers = ['k', 'nearest neighbors']
108     with open("tables/fmeasure-knearest-stemmed.text", "w+") as out
    :
109         out.write(tabulate(mout, headers=headers, tablefmt="latex")
    )
110     data.insert(fmeasure_idk, takeout)
111     print("http://ws-dl.blogspot.com/")
112     data.pop(wsdl_idk)
113     mout.clear()
114     for k in [1, 2, 5, 10, 20]:
115         if k != 1:
116             val = knnestimate(data, data[fmeasure_idk]['input'], k)
    [klast:]
117         else:
118             val = knnestimate(data, data[fmeasure_idk]['input'], k)
119         klast = k
120         mout.append(["%d" % k, ", ".join(val)])
121         print("The %d nearest neighbors are:" % k, knnestimate(data
    , data[fmeasure_idk]['input'], k))
122     with open("tables/wsdl-knearest-stemmed.text", "w+") as out:
123         out.write(tabulate(mout, headers=headers, tablefmt="latex")
    )
124
125
126 # execute question 1
127 def a8():
128     nonstemmeda8()
129     print("-------------------------------------")
130     stemmeda8()
131
132
133 # helper method for classification using SVM
134 # extracts the label for blog structure classification
135 def blog_structure_helper(name):
136     cn = findClass.match(name).group(1).lower().strip()
137     if "concert" in cn or 'spotlight' in cn:
138         cn = 'concert/spotlight'
139     return cn
140
141
142 # helper method for classification using SVM
143 # extracts the label for genre classification
144 def blog_genre_helper(name):
145     m = artistsExtractor.match(name)
146     if m is not None:
```

```
        cn = aTg[m.group(1).rstrip(" -")]
    else:
        cn = "R&B/Jazz/Mowtown/Country/Other"
    return cn


# method to execute the 10 fold SVM classification
def tfold_blog(fname, labels, lextractor, outname):
    """
    :param fname: the data file to read in
    :param labels: the labels to be used for classification
    :param lextractor: the label extractor
    :param outname:  the output file name
    """
    # read in the data
    blognames, words, data = readfile(fname)

    # create lookup table for the blog names
    blognameidk = {}
    for i in range(len(blognames)):
        blognameidk[blognames[i]] = i
    # reporting data list
    mout = []
    mout2 = []

    # do 10 fold validation per label
    # each label is expanded to label or not label
    for clazz in labels:
        print(clazz)
        # get a fold instance
        kf = KFold(len(data), 10)
        # holds the results per fold
        kfhMetrics = defaultdict(list)
        ac = []
        tcIndex = 1
        # loop through each folds train and text index
        for train_index, test_index in kf:
            # holders for data results
            train = []
            test = []
            lab = []
            tlab = []
            actualLabels = []
            # make training data
            for i in train_index:
                clazz1 = lextractor(blognames[i])
                if clazz in clazz1:
                    tlab.append(1)
                else:
                    tlab.append(-1)
                train.append(data[i]['input'])
            # make test data
            for i in test_index:
                clazz1 = lextractor(blognames[i])
                test.append(data[i]['input'])
                if clazz in clazz1:
                    lab.append(1)
```

```
                    actualLabels.append(clazz)
                else:
                    lab.append(-1)
                    actualLabels.append("not " + clazz)
            # create a problem
            # libsvm 3.21 changed the interface from the example from Toby Segaran
            prob = svm_problem(tlab, train)
            # create parameters: -s 2[one-class svm], -t 0 linear kernel
            param = svm_parameter('-s 2 -t 0')
            # train a model
            m = svm_train(prob, param)
            # get the predicted labels, prediction accuracy
            p_label, p_acc, p_val = svm_predict(lab, test, m=m)
            ac.append(p_acc[0])
            print( p_acc)
            # transform the labels into reportable format
            predictedLabels = [clazz if l == 1.0 else "not " + clazz for l in p_label]
            # take mean of the values as the scikit learn scores returns the values for each label
            kfhMetrics['precision'].append(
                statistics.mean(precision_score(actualLabels, predictedLabels, average=None)))
            kfhMetrics['recall'].append(statistics.mean(recall_score(actualLabels, predictedLabels, average=None)))
            kfhMetrics['f1'].append(statistics.mean(f1_score(actualLabels, predictedLabels, average=None)))
            ac.append(p_acc[0])
            tcIndex += 1
        # create a entry for our report tabel
        ret1 = [clazz, statistics.mean(kfhMetrics['precision']), statistics.mean(kfhMetrics['recall']),
                statistics.mean(kfhMetrics['f1'])]
        mout.append(ret1)
        acc = "%.2f" % statistics.mean(ac)
        mout2.append([ "%s, not %s" % (clazz, clazz), acc + "%"])


    # write out report
    headers = ["label", "precision", 'recall', 'f1']
    with open(outname, "w+") as out:
        out.write(tabulate(mout, headers=headers, tablefmt="latex"))
    headers = [ "label", 'mean accuracy']
    with open(outname+"acc", "w+") as out:
        out.write(tabulate(mout2, headers=headers, tablefmt="latex"))


def do_ten_fold():
    # do ten fold SVM, repeating process done in A9
    bf_nonstemmed = "datafiles/fmeasure.txt"
    bf_stemmed = "datafiles/fmeasure_stemmed.txt"
    blog_structure_labels = ['the song remains the same', 'concert/spotlight', 'lp review', 'forgotten song']
```

```
250     blog_genre_labels = ['Rock', 'R&B/Jazz/Mowtown/Country/Other',
        'Pop/Electronic/Hip-Hop', 'Metal/Punk/Hardcore',
251                          'Indie Rock/Alternative']
252
253     tfold_blog(bf_nonstemmed, blog_structure_labels,
        blog_structure_helper, "tables/tenfold-metrics-structure.txt")
254     tfold_blog(bf_stemmed, blog_structure_labels,
        blog_structure_helper, "tables/tenfold-metrics-structure-
        stemmed.txt")
255
256     tfold_blog(bf_nonstemmed, blog_genre_labels, blog_genre_helper,
         "tables/tenfold-metrics-genre.txt")
257     tfold_blog(bf_stemmed, blog_genre_labels, blog_genre_helper, "
        tables/tenfold-metrics-genre-stemmed.txt")
258
259
260 if __name__ == '__main__':
261     # read in data-file
262     consume_all("datafiles/f-measure.xml")
263     generate_blogfile()
264     generate_blogfile_stem()
265     nonstemmeda8()
266     stemmeda8()
267     do_ten_fold()
```

Listing 1: KNN and SVM Classification

```
import re
from collections import defaultdict
from operator import add
import feedparser
import nltk
from bs4 import BeautifulSoup
from functional import seq
from nltk.corpus import stopwords
from nltk.stem.snowball import EnglishStemmer
import json as jjson

removeExtra = re.compile('[^a-zA-Z]')
stop = stopwords.words('english')


def getwords(doc):
    splitter = re.compile('\\W*')
    # print doc
    ## Remove all the HTML tags
    doc = re.compile(r'<[^>]+>').sub('', doc)
    # Split the words by non-alpha characters
    words = [s.lower().replace('\'', '') for s in nltk.
    word_tokenize(doc)
            if len(s) > 2 and len(s) < 20]
    words = seq(words ).map(lambda word: (word, 1)) \
        .reduce_by_key(add) \
        .order_by(lambda x: x[1]).to_dict()
    # Return the unique set of words only
    return dict([(w, 1) for w in words])

# modified process text
def process_text(text):
    # clean text
    t = removeExtra.sub(' ', BeautifulSoup(text.content[0].value, '
    html5lib').text).lower()
    ret = []
    # tokenize according to word and emit word if it is not a
    stopword
    for word in nltk.word_tokenize(t):
        if len(word) > 2:
            ret.append(word)
    ret = seq(ret).map(lambda word: (word, 1)) \
        .reduce_by_key(add) \
        .order_by(lambda x: x[1]).to_dict()
    return {'title': text.title, 'text': ret}

# modified consume all since
def consume_all(fileName):
    def foldHelp(acum,v):
        acum[v['title']] = v['text']
        return acum
    with open("datafiles/fmeasure.json","w+") as out:
        out.write(jjson.dumps(seq(feedparser.parse(fileName).
    entries).map(process_text).fold_left({},foldHelp),indent=1))

    with open("datafiles/inorder.txt","w+") as out:
        seq(feedparser.parse(fileName).entries).map(lambda e: out.
```

```
    write("%s\n"%e.title)).to_list()

# this class represents the word data for a particular feed
class feed:
    # pass flag if we are to do the stemming
    def __init__(self, fentry, doStem=False):
        self.title = fentry[0]
        self.wordCount = fentry[1]
        self.stemCount = defaultdict(int)
        if doStem:
            self._stem_count()

    def _stem_count(self):
        eng = EnglishStemmer()
        for word in self.wordCount.keys():
            self.stemCount[eng.stem(word)] += 1

    def words(self):
        return set(self.wordCount.keys())

    def __str__(self):
        return "%s: %s" % (self.title, ' '.join(list(self.wordCount
    .keys())))


# fake tfidf
def filter_fun(wc):
    frac = float(wc[1]) / float(100)
    return 0.1 < frac < 0.5


# output for the non-stemmed data file
def output(f, top):
    out = [f.title]
    for wd in top:
        out.append("%d" % f.wordCount.get(wd, 0))
    return '\t'.join(out) + "\n"


# output for the stemmed data file
def output_stem(f, top):
    out = [f.title]
    for wd in top:
        out.append("%d" % f.stemCount.get(wd, 0))
    return '\t'.join(out) + "\n"


def generate_blogfile():
    # read the data in as json and then transform to feeds
    feedData = seq.json("datafiles/fmeasure.json").map(lambda fe:
    feed(fe)).to_list()  # type: list[feed]
    '''
    get the top 500 words by word count over all words:
        for feed <- feeds, (word,count) <- feed.wordCount.items():
    yeild (word,count)
        keep all wordCounts > 10
        groupby + reduce word:(word1,c1),(word1,c2) -> (word1,c1,c2
```

```
    ,c3,c4) -> (word1,sumC)
107         keep all wordCounts that meet fake tfidf
108         transform (word,sumC) -> word
109         take top 500
110         transform to list
111     '''
112     top500 = seq(feedData).flat_map(lambda f: list(f.wordCount.
    items())) \
113         .map(lambda wc: (wc[0], wc[1])) \
114         .reduce_by_key(add) \
115         .filter(filter_fun) \
116         .order_by(lambda wc: -wc[1]) \
117         .map(lambda wc: wc[0]) \
118         .to_list()
119     # sort alphabetically
120     top500 = sorted(top500)
121     print(len(top500))
122     # write resultant to file
123     with open("datafiles/fmeasure.txt", "w+") as out:
124         out.write("Blog\t%s\n" % '\t'.join(top500))
125         for tf in sorted(feedData, key=lambda f: f.title):
126             out.write(output(tf, top500))
127
128
129 def generate_blogfile_stem():
130     # same as non-stem except use stemmed data
131     feedData = seq.json("datafiles/fmeasure.json").map(lambda fe:
    feed(fe, True)).to_list()  # type: list[feed]
132     top500 = seq(feedData).flat_map(lambda f: list(f.stemCount.
    items())) \
133         .map(lambda wc: (wc[0], wc[1])) \
134         .reduce_by_key(add) \
135         .filter(filter_fun) \
136         .order_by(lambda wc: -wc[1]) \
137         .map(lambda wc: wc[0]) \
138         .to_list()
139     top500 = sorted(top500)
140     print(len(top500))
141     with open("datafiles/fmeasure_stemmed.txt", "w+") as out:
142         out.write("Blog\t%s\n" % '\t'.join(top500))
143         for tf in sorted(feedData, key=lambda f: f.title):
144             out.write(output_stem(tf, top500))
```

Listing 2: Proccess Data for A10