Appendix
A – Task 1

A.1 Emoticons List

=)
:)
:-)
:D
:-D
(=
(:
(-:
=D
B-)
B)
^_^
C:
C-:
^^
=(
:(
:-(
)=
):
)-:
D:
D-:
D=
B-(
:O
O:
:-O
O-:
=O
O=
:S
S:
:-S
S-:
=S
S=
=#
#=
><
=X
X=
:X
X:
-_-
-.-
:|
|:

=|
|=
:-|
|-:
:\
:/
:-\
:-/
\:
\-:
/:
/-:
=\
=/
\=
/=
:$
=$
$:
$=
:C
:-C
=C
C=

A.2 Negations List

never
nothing
nowhere
noone
none
not
havent
haven't
hasnt
hasn't
hadnt
hadn't
cant
can't
couldnt
couldn't
shouldnt
shouldn't
wont
won't
wouldnt
wouldn't
dont
don't
doesnt

```
doesn't
didnt
didn't
isnt
isn't
arent
aren't
aint
```

A.3 - Code

```
import re
import csv
import time
from random import sample
import matplotlib.pyplot as plt

################################
######### OPEN STUFF ###########
################################
#opens lists of data and words to select features
print 'open stuff'
#list of all reddit comments to evaluate
reddit_data = open("../data/reddit_data")
reddit_str = reddit_data.read()
reddit_data.close()
#list of known swear words
swears = open("../data/swearwordslist.lst")
swears_str = swears.read().splitlines()
swears.close()
#list of several emoticons used, to express emotion
emoticons = open("../data/emoticonslist.lst")
emoticons_str = emoticons.read().splitlines()
emoticons.close()
#list of negative words, like 'not', 'ain't, etc
negations = open("../data/negationslist.lst")
negations_str = negations.read().splitlines()
negations.close()
#list of words used to express a negative intent
negative_words = open("../data/negative-words.txt")
neg_words_str = negative_words.read().splitlines()
negative_words.close()
#list of words used to express a positive intent
positive_words = open("../data/positive-words.txt")
pos_words_str = positive_words.read().splitlines()
positive_words.close()


################################
######### READ CONTENT #########
################################
#from the list of reddit comments, divides up votes, down votes and
```

```
comments in a list of lists
print 'read content'
everything = re.findall("\<UP\>(?P<upVotes>\d+)\</UP\>\n\<DOWN\>(?
P<downVotes>\d+)\</DOWN\>\n(?P<document>.*)", reddit_str)
#content is a list of only the comments, without the votes
content = [x[2] for x in everything]


################################
######## MAKE PATTERN ##########
################################
#create a pattern to extract features. any word equal to this
pattern will be kept, and all the others removed.
# the pattern is made of all the words in the previously opened
lists plus any word that has 3 or more consecutive
# vowels (to catch words like goood, although words like
'prestigious' are also caught) plus only uppercase words
# (words like 'SHUT' or similiar, which express strong emotions) and
also sequences of '?' and '!' longer than 1.
print 'make pattern'
all_words_list = swears_str + emoticons_str + negations_str +
neg_words_str + pos_words_str
pattern = "(?:[\?\!]{2,}|[A-Z]{3,}|[aeiou]{3,}|%s)" %
"|".join(map(re.escape, all_words_list))
pattern = re.compile(pattern)


################################
###### EXTRACT FEATURES ########
################################
#1st run: compares the list of content with the pattern made
#2nd run: due to long calculations time, and for further tests
unrelated to this section, a features list was
# saved in a file and is being extracted from there. If any change
in this or previous sections is made, please
# comment 2nd method and uncomment 1st to rerun comparison. (expect
times around 500s)
print 'extract features'
#features = [re.findall(pattern, article) for article in content]
with open('features.csv', 'Ur') as f:
    features = list(rec for rec in csv.reader(f, delimiter=','))


################################
#### MAKE TRAINING/TEST SET ####
################################
n_tests = 50
population = range(0,len(features))
indexes = sample(population, n_tests)


################################
###### COMPUTE DISTANCES #######
################################
```

```
# measure the distance between the test lists and the training ones,
with the KNN classifier
print 'calculating distance for test set'
big = [99999999999, 9999]
#k nearest neighbours
k = 10
indexes_dist = []
for i in indexes:
        test_distance = [big]*k
        test_feat = features[i]
        for feat_list in features:
                if len(feat_list) > len(test_feat)/5 and
features.index(feat_list) not in indexes:
                        temp_test_feat = [word for word in
test_feat]
                        [[feat_list.remove(word),
temp_test_feat.remove(word)] for word in feat_list if word in
temp_test_feat]
                        distance = [len(temp_test_feat) +
len(feat_list), features.index(feat_list)]
                        if distance < max(test_distance):

test_distance.remove(max(test_distance))
                                test_distance.append(distance)
        indexes_dist.append(test_distance)



################################
####### PREDICT RESULT #########
################################

print 'Averaging neighbours ratios'
dist_sums = []
for index in indexes_dist:

        ratio_list = [(float(everything[x[1]][0])+1.0)/
(float(everything[x[1]][1])+1.0) for x in index]
        dist_sums.append(sum(ratio_list)/k)


################################
####### GET DIFFERENCES ########
################################

print "Comparing predition with real results"
print "predition vs real"

real_results = [(float(everything[x][0]) + 1.0)/(float(everything[x]
[1]) + 1.0) for x in indexes]

avg_sum = 0
for x in range(0, len(dist_sums)):
        print"{0:.2f}".format(real_results[x]) + " vs " +
```

```
"{0:.2f}".format(dist_sums[x])
        avg_sum = avg_sum + abs(real_results[x] - dist_sums[x])

print avg_sum/len(dist_sums)

################################
###### PLOTTING RESULTS ########
################################

x = range(0, len(dist_sums))
y1 = real_results
y2 = dist_sums
fig, ax = plt.subplots()
ax.plot(x, y1, 'k--')
ax.plot(x, y2, 'ro')

# set ticks and tick labels
ax.set_xlim((0, len(dist_sums)+5))
ax.set_xticks(range(0, len(dist_sums)+1, 5))

# Only draw spine between the y-ticks
ax.spines['left'].set_bounds(-1, 1)
# Hide the right and top spines
ax.spines['right'].set_visible(False)
ax.spines['top'].set_visible(False)
# Only show ticks on the left and bottom spines
ax.yaxis.set_ticks_position('left')
ax.xaxis.set_ticks_position('bottom')

plt.show()
```