ist-664-final-project-pardes-033021

March 30, 2021

[1]:
```
'''
Sammy Pardes
IST 664
Final Project
3/30/21
'''
```

[1]:
```
'\nSammy Pardes\nIST 664\nFinal Project\n3/30/21\n'
```

[2]:
```
'''
  This program shell reads email data for the spam classification problem.
  The input to the program is the path to the Email directory "corpus" and a␣
↪limit number.
  The program reads the first limit number of ham emails and the first limit␣
↪number of spam.
  It creates an "emaildocs" variable with a list of emails consisting of a pair
    with the list of tokenized words from the email and the label either spam␣
↪or ham.
  It prints a few example emails.
  Your task is to generate features sets and train and test a classifier.

  Usage:  python classifySPAM.py  <corpus directory path> <limit number>
'''
# open python and nltk packages needed for processing
import os
import sys
import random
import nltk
from nltk.corpus import stopwords

# define a feature definition function here

# function to read spam and ham files, train and test a classifier
def processspamham(dirPath,limitStr):
    # convert the limit argument from a string to an int
    limit = int(limitStr)
    # start lists for spam and ham email texts
    hamtexts = []
    spamtexts = []
    os.chdir(dirPath)
    # process all files in directory that end in .txt up to the limit
    #    assuming that the emails are sufficiently randomized
    for file in os.listdir("./spam"):
        if (file.endswith(".txt")) and (len(spamtexts) < limit):
            # open file for reading and read entire file into a string
            f = open("./spam/"+file, 'r', encoding="latin-1")
            spamtexts.append (f.read())
            f.close()
    for file in os.listdir("./ham"):
        if (file.endswith(".txt")) and (len(hamtexts) < limit):
            # open file for reading and read entire file into a string
            f = open("./ham/"+file, 'r', encoding="latin-1")
            hamtexts.append (f.read())
            f.close()

    # print number emails read
    print ("Number of spam files:",len(spamtexts))
    print ("Number of ham files:",len(hamtexts))
    print

    # create list of mixed spam and ham email documents as (list of words,␣
↪label)

    #make emaildocs a global variable
    global emaildocs

    emaildocs = []
    # add all the spam
    for spam in spamtexts:
        tokens = nltk.word_tokenize(spam)
        emaildocs.append((tokens, 'spam'))

    # add all the regular emails
    for ham in hamtexts:
        tokens = nltk.word_tokenize(ham)
        emaildocs.append((tokens, 'ham'))

    # randomize the list
    random.shuffle(emaildocs)

    # print a few token lists
    for email in emaildocs[:4]:
        print (email)
        print
```

[3]:

```
processspamham("C:/Users/slpar/OneDrive/Desktop/graduate/IST664/final-project/
 ↪FinalProjectData/EmailSpamCorpora/corpus",
                1500)
```

```
Number of spam files: 1500
Number of ham files: 1500
(['Subject', ':', 'prozacs', 'meds', '30', 'million', 'people', 'now', 'rely',
'on', 'this', 'wonder', 'drug', '!', 'get', 'prozac', 'now', 'relax', 'and',
'enjoy', 'life', '!', '!', '!', '!', '!', '!', 'if', 'you', 'do', 'not', 'wish',
'to', 'receive', 'further', 'emails', 'please', 'click', 'here', '.', 'we',
'honor', 'all', 'unsubscribe', 'requests', 'immediatly', '.'], 'spam')
(['Subject', ':', 're', ':', 'industrial', 'report', 'i', 'am', 'referring',
'to', 'ken', 'seaman', "'", 's', 'file', 'which', 'provides', 'additional',
'information', '.', 'we', 'definitely', 'need', 'ken', "'", 's', 'file',
'completed', 'by', 'aug', 'lst', '.', 'and', 'distributed', 'to', 'our',
'group', ',', 'daren', 'farmer', ',', 'and', 'gary', 'hanks', '.', 'both',
'files', 'should', 'be', 'completed', 'and', 'can', 'be', 'linked', 'on',
'duplicate', 'information', '.', 'thanks', ',', 'pat', 'robert', 'e', 'lloyd',
'@', 'ect', '07', '/', '31', '/', '2000', '09', ':', '29', 'am', 'to', ':',
'pat', 'clynes', '/', 'corp', '/', 'enron', '@', 'enron', 'cc', ':', 'robert',
'e', 'lloyd', '/', 'hou', '/', 'ect', '@', 'ect', ',', 'daren', 'j', 'farmer',
'/', 'hou', '/', 'ect', '@', 'ect', ',', 'gary', 'a', 'hanks', '/', 'hou', '/',
'ect', '@', 'ect', 'subject', ':', 're', ':', 'industrial', 'report', 'i', 'am',
'confused', '.', 'what', 'file', 'do', 'you', 'want', 'me', 'to', 'update', ';',
'julie', 'meyers', 'file', 'or', 'ken', 'seaman', 'file', '?', 'both', 'files',
'provide', 'similar', 'data', 'which', 'is', 'helpful', 'to', 'all', 'of',
'our', 'support', 'groups', '.', 'to', 'prepare', 'both', 'files', 'i', 'feel',
'is', 'a', 'duplicate', 'of', 'my', 'efforts', '.', 'please', 'let', 'know',
'if', 'you', 'want', 'both', 'files', 'updated', 'and', 'the', 'approximate',
'time', 'table', 'for', 'completion', '.', 'enron', 'north', 'america', 'corp',
'.', 'from', ':', 'pat', 'clynes', '@', 'enron', '07', '/', '31', '/', '2000',
'09', ':', '02', 'am', 'to', ':', 'robert', 'e', 'lloyd', '/', 'hou', '/',
'ect', '@', 'ect', 'cc', ':', 'daren', 'j', 'farmer', '/', 'hou', '/', 'ect',
'@', 'ect', ',', 'gary', 'a', 'hanks', '/', 'hou', '/', 'ect', '@', 'ect',
'subject', ':', 're', ':', 'indutrial', 'report', 'robert', ',', 'this', 'is',
'the', 'file', 'that', 'i', 'referenced', 'in', 'my', 'last', 'email', '.',
'please', 'get', 'this', 'file', 'going', 'again', '.', 'thanks', ',', 'pat',
'daren', 'j', 'farmer', '@', 'ect', '02', '/', '25', '/', '2000', '04', ':',
'52', 'pm', 'to', ':', 'robert', 'e', 'lloyd', '/', 'hou', '/', 'ect', '@',
'ect', 'cc', ':', 'pat', 'clynes', '/', 'corp', '/', 'enron', '@', 'enron',
'subject', ':', 'indutrial', 'report', 'robert', ',', 'ken', 'developed', 'an',
'industrial', 'report', 'before', 'he', 'left', '.', 'it', 'can', 'be', 'found',
'at', 'o', '/', 'logistics', '/', 'kenseaman', '/', 'industrialsmonthly', '/',
'.', '.', '.', 'there', 'is', 'one', 'file', 'for', 'each', 'month', 'of',
'2000', '.', 'i', 'need', 'you', 'to', 'update', 'this', 'for', 'march', '.',
'this', 'will', 'need', 'to', 'be', 'distributed', 'to', 'gas', 'control', ',',
'logistics', ',', 'and', 'myself', '.', 'let', 'me', 'know', 'if', 'you',
'have', 'any', 'questions', '.', 'd'], 'ham')
(['Subject', ':', 'young', 'pussies', 'tonya', 'could', 'feel', 'the', 'glow',
'of', 'the', 'hundreds', 'of', 'candles', 'on', 'her', 'bare', 'skin', '.',
'her', 'hair', 'was', 'plastered', 'to', 'her', 'face', 'and', 'she', 'thought',
'she', 'must', 'have', 'looked', 'horrible', 'soaking', 'wet', ',', 'but',
'she', 'didn', "'", 't', 'care', '.', 'gabriel', 'thought', 'she', 'was',
'beautiful', 'and', 'that', 'was', 'all', 'she', 'needed', 'to', 'know', '.',
'tonya', 'slid', 'toward', 'him', 'in', 'the', 'warm', 'water', '.', 'all',
'inside', '!', 'remove', 'your', 'email'], 'spam')
(['Subject', ':', 'shell', 'houston', 'open', 'first', 'come', 'first', 'serve',
':', 'i', 'have', ':', '10', 'shell', 'houston', 'open', 'badges', 'available',
'.', '(', 'let', "'", 's', 'try', 'to', 'share', 'these', 'as', 'best', 'we',
'can', '.', ')', '10', 'one', 'day', 'passes', '2', 'invitations', 'to',
'tonights', 'mardi', 'gras', 'party', '4', 'tickets', 'to', 'this', 'saturday',
"'", 's', 'beauty', 'n', 'blues', 'concert', '.', 'this', 'might', 'be', 'a',
'good', 'time', 'to', 'remind', 'you', 'that', 'this', 'week', 'is',
'secretary', "'", 's', 'week', '(', 'since', 'i', 'am', 'holding', 'all', 'the',
'goodies', ')', 'just', 'joking', 'brenda', '.', 'yvette', 'x', '3', '.',
'5953'], 'ham')
```

[13]:

```python
#get stopwords list from NLTK
stopwords = nltk.corpus.stopwords.words('english') #get basic list of stopwords

#create new list of stopwords
mystop = [":", "+", "_", "``", "'", ".", ",", "!", "/", "-", ")", "(", "*",␣
 ↪"%", "=", "@",
          "|", "\\", "[", "]", "?", "#", "{", "}",";", "enron","subject",␣
 ↪"steve", "vance",
          "susan", "lloyd", "brenda", "jackie", "howard", "stacey", "lisa",␣
 ↪"gary", "hanks",
          "meyers", "carlos", "donald", "julie", "taylor"]

#add nltk and custom stopwords together
all_stop = stopwords + mystop
```

[14]:

```python
#define function to filter out stopwords

def filter_stopwords(email_list):
    global filtered_emails #store filtered emails as a global variable
    filtered_emails = [] #initialize empty list
    for email in email_list: #for each email/category in the email in the given␣
 ↪list (emaildocs)
        email_words = [] #initialize an empty list to store words
        for word in email[0]: #for each word in the email text
            if word.lower() not in all_stop and len(word) > 3: #if lowercased␣
 ↪word is not a stopword and is over 3 characters
```

```
                email_words.append(word.lower()) #append lowercased word to␣
↪email_words list
        filtered_emails.append(tuple((email_words, email[1]))) #add tuple of␣
↪email and category to filtered_emails list
```

[15]:
```
#run filter_stopwords function on emaildocs
filter_stopwords(emaildocs)

#compare filtered vs. non-filtered email
print(filtered_emails[0], '\n')
print(emaildocs[0])
```

```
(['prozacs', 'meds', 'million', 'people', 'rely', 'wonder', 'drug', 'prozac',
'relax', 'enjoy', 'life', 'wish', 'receive', 'emails', 'please', 'click',
'honor', 'unsubscribe', 'requests', 'immediatly'], 'spam')

(['Subject', ':', 'prozacs', 'meds', '30', 'million', 'people', 'now', 'rely',
'on', 'this', 'wonder', 'drug', '!', 'get', 'prozac', 'now', 'relax', 'and',
'enjoy', 'life', '!', '!', '!', '!', '!', '!', 'if', 'you', 'do', 'not', 'wish',
'to', 'receive', 'further', 'emails', 'please', 'click', 'here', '.', 'we',
'honor', 'all', 'unsubscribe', 'requests', 'immediatly', '.'], 'spam')
```

[16]:
```
#extract only the words from the filtered emails
filtered_words = [] #initialize empty list
for email in filtered_emails: #for each email in the filtered_emails list
    for word in email[0]: #for each word in the email text
        filtered_words.append(word) #append the word to filtered_words

#preview some of the filtered words
print(filtered_words[:20])
```

```
['prozacs', 'meds', 'million', 'people', 'rely', 'wonder', 'drug', 'prozac',
'relax', 'enjoy', 'life', 'wish', 'receive', 'emails', 'please', 'click',
'honor', 'unsubscribe', 'requests', 'immediatly']
```

[17]:
```
#get frequency distribution of filtered words
filtered_freq_dist = nltk.FreqDist(filtered_words)
filtered_freq_dist
```

[17]:
```
FreqDist({'2000': 2826, 'please': 1753, 'meter': 1285, 'deal': 1075, 'corp':
1069, 'http': 1062, 'company': 907, 'thanks': 876, 'know': 789, 'information':
788, …})
```

[18]:
```
#get the top 100 most common words
filtered_freq_dist_common = filtered_freq_dist.most_common(100)

print("Most frequent filtered words with counts:", filtered_freq_dist_common[:
↪100])
```

```
Most frequent filtered words with counts: [('2000', 2826), ('please', 1753),
('meter', 1285), ('deal', 1075), ('corp', 1069), ('http', 1062), ('company',
907), ('thanks', 876), ('know', 789), ('information', 788), ('forwarded', 764),
('need', 747), ('daren', 743), ('price', 703), ('time', 648), ('mmbtu', 604),
('email', 598), ('would', 581), ('robert', 528), ('font', 515), ('mail', 507),
('sitara', 493), ('report', 492), ('statements', 482), ('month', 481), ('july',
453), ('attached', 448), ('also', 442), ('contract', 440), ('energy', 436),
('like', 426), ('farmer', 423), ('free', 423), ('nbsp', 418), ('volume', 408),
('deals', 407), ('business', 405), ('message', 404), ('want', 393),
('questions', 392), ('contact', 381), ('make', 379), ('change', 374),
('volumes', 372), ('within', 372), ('height', 362), ('production', 359),
('call', 357), ('well', 350), ('could', 343), ('stock', 341), ('forward', 333),
('back', 333), ('today', 331), ('line', 326), ('number', 323), ('list', 322),
('ticket', 319), ('following', 319), ('money', 313), ('order', 311), ('pills',
311), ('size', 310), ('width', 306), ('best', 305), ('click', 302), ('take',
301), ('first', 300), ('2004', 300), ('texas', 299), ('investment', 296),
('looking', 291), ('online', 291), ('days', 287), ('available', 283), ('system',
283), ('file', 282), ('products', 282), ('america', 278), ('june', 277),
('flow', 273), ('securities', 271), ('future', 269), ('sent', 267), ('next',
265), ('north', 263), ('effective', 263), ('management', 263), ('sale', 263),
('product', 263), ('chokshi', 262), ('service', 258), ('group', 257), ('help',
257), ('made', 256), ('prices', 256), ('services', 255), ('office', 254),
('many', 254), ('based', 252)]
```

[19]:

```python
#store most common words only, no counts
freq_words_only = [] #initalize empty list
for (word, count) in filtered_freq_dist_common: #for each word and count in the␣
 ↪frequency distribution
    freq_words_only.append(word) #append only the word to freq_words_only list

#distplay top 20 most frequent words and the total length of the␣
 ↪freq_words_only list (100)
print(freq_words_only[:20])
len(freq_words_only)
```

```
['2000', 'please', 'meter', 'deal', 'corp', 'http', 'company', 'thanks', 'know',
'information', 'forwarded', 'need', 'daren', 'price', 'time', 'mmbtu', 'email',
'would', 'robert', 'font']
```

[19]:

```
100
```

[20]:

```python
#define feature function based on frequency
def freq_features(email, word_features): #initalize function given email and␣
 ↪word_feature variables as input
    email_words = set(email) #tokenize the email, store as email_words
    features = {} #initialize empty dictionary
    for word in word_features: #for each word in the email
```

```
        features['is_freq_{}'.format(word)] = (word in email_words) #add␣
 ↪"is_freq_", check if word is in the word_features list
    return features
```

[56]:

```
#run feature function on email list
freq_feature_set = [(freq_features(email, freq_words_only), category) #run␣
 ↪freq_features() on each email given freq_words_only list and keep spam/ham␣
 ↪classifier
                    for (email, category) in filtered_emails] #for each email␣
 ↪and spam/ham class in filtered list

email_words = []
for email in emaildocs:
    for word in email[0]:
        email_words.append(word)

unfiltered_freq_dist = nltk.FreqDist(email_words)
unfiltered_freq_dist

unfiltered_freq_dist_common = unfiltered_freq_dist.most_common(100)

unfiltered_freq_words_only = []
for (word, count) in unfiltered_freq_dist_common:
    unfiltered_freq_words_only.append(word)

unfiltered_freq_feature_set = [(freq_features(email,␣
 ↪unfiltered_freq_words_only), category) #run freq_features() on each email␣
 ↪given freq_words_only list and keep spam/ham classifier
                    for (email, category) in emaildocs] #for each email and␣
 ↪spam/ham class in filtered list

#show first email after running the freq_features function
#freq_feature_set[0]
unfiltered_freq_feature_set[0]
```

[56]:

```
({'is_freq_-': False,
  'is_freq_.': True,
  'is_freq_/': False,
  'is_freq_,': False,
  'is_freq_:': True,
  'is_freq_the': False,
  'is_freq_to': True,
  'is_freq_ect': False,
  'is_freq_and': True,
  'is_freq_of': False,
  'is_freq_@': False,
  'is_freq_a': False,
  'is_freq_for': False,
  'is_freq_?': False,
  'is_freq_you': True,
  'is_freq_in': False,
  'is_freq_this': True,
  'is_freq_is': False,
  'is_freq_hou': False,
  'is_freq_on': True,
  'is_freq_i': False,
  "is_freq_'": False,
  'is_freq_)': False,
  'is_freq_=': False,
  'is_freq_(': False,
  'is_freq_enron': False,
  'is_freq_Subject': True,
  'is_freq_!': True,
  'is_freq_be': False,
  'is_freq_your': False,
  'is_freq_2000': False,
  'is_freq_that': False,
  'is_freq_with': False,
  'is_freq_from': False,
  'is_freq__': False,
  'is_freq_will': False,
  'is_freq_have': False,
  'is_freq_we': True,
  'is_freq_s': False,
  'is_freq_as': False,
  'is_freq_are': False,
  'is_freq_it': False,
  'is_freq_$': False,
  'is_freq_>': False,
  'is_freq_or': False,
  'is_freq_3': False,
  'is_freq_at': False,
  'is_freq_not': True,
  'is_freq_by': False,
  'is_freq_please': True,
  'is_freq_``': False,
  'is_freq_com': False,
  'is_freq_if': True,
  'is_freq_|': False,
  'is_freq_1': False,
  'is_freq_;': False,
  'is_freq_#': False,
  'is_freq_our': False,
  'is_freq_me': False,
  'is_freq_2': False,
  'is_freq_e': False,
  'is_freq_subject': False,
  'is_freq_all': True,
  'is_freq_gas': False,
  'is_freq_00': False,
  'is_freq_%': False,
  'is_freq_*': False,
  'is_freq_meter': False,
  'is_freq_am': False,
  'is_freq_can': False,
  'is_freq_any': False,
  'is_freq_cc': False,
  'is_freq_pm': False,
  'is_freq_d': False,
  'is_freq_000': False,
  'is_freq_deal': False,
  'is_freq_corp': False,
  'is_freq_http': False,
  'is_freq_has': False,
  'is_freq_no': False,
  'is_freq_an': False,
  'is_freq_0': False,
  'is_freq_re': False,
  'is_freq_4': False,
  'is_freq_10': False,
  'is_freq_new': False,
  'is_freq_hpl': False,
  'is_freq_company': False,
  'is_freq_5': False,
  'is_freq_was': False,
  'is_freq_thanks': False,
  'is_freq_up': False,
  'is_freq_7': False,
  'is_freq_get': True,
  'is_freq_t': False,
  'is_freq_99': False,
  'is_freq_&': False,
  'is_freq_know': False,
  'is_freq_information': False,
  'is_freq_may': False},
 'spam')
```

[22]:

```
#split data for testing and training

#get 30% of data
thirty_percent = int(len(filtered_emails)*0.3)
```

```
thirty_percent

#reserve 70% of data for training, 30% for testing
freq_train_set, freq_test_set = unfiltered_freq_feature_set[thirty_percent:],␣
 ↪unfiltered_freq_feature_set[:thirty_percent]

#run NLTK Naive Bayes classifier on training data
freq_classifier = nltk.NaiveBayesClassifier.train(freq_train_set)

#display accuracy of running the classifier on the test data
print(nltk.classify.accuracy(freq_classifier, freq_test_set))
```

```
0.8933333333333333
```

[23]:

```
#create confusion matrix function
def confusion_matrix(train_set, test_set, classifier):
    actual_list = [] #initalize empty lists for actual and predicted results
    predicted_list = []
    for (email, category) in test_set: #for each email in the test data
        actual_list.append(category) #add the true spam or ham tag to the␣
 ↪actual_list
        predicted_list.append(classifier.classify(email)) #add the predicted␣
 ↪class to the predicted_list

    #check out at the first 30 examples
    print(actual_list[:30])
    print(predicted_list[:30])

    #create a confusion matrix with ConfusionMatrix()
    cm = nltk.ConfusionMatrix(actual_list, predicted_list)
    print(cm.pretty_format(sort_by_count=True, truncate=9))

    #evaluation metrics
    labels = list(set(actual_list))
    recall_list = [] #initialize empty lists
    precision_list = []
    f1_list = []
    for label in labels:
        # for each label, compare gold and predicted lists and compute values
        TP = FP = FN = TN = 0
        for i, val in enumerate(actual_list):
            if val == label and predicted_list[i] == label:  TP += 1
            if val == label and predicted_list[i] != label:  FN += 1
            if val != label and predicted_list[i] == label:  FP += 1
            if val != label and predicted_list[i] != label:  TN += 1
        # use these to compute recall, precision, F1
        recall = TP / (TP + FP)
        precision = TP / (TP + FN)
        recall_list.append(recall)
        precision_list.append(precision)
        f1_list.append( 2 * (recall * precision) / (recall + precision))

    # the evaluation measures in a table with one row per label
    print('\tPrecision\tRecall\t\tF1')
    # print measures for each label
    for i, lab in enumerate(labels):
        print(lab, '\t', "{:10.3f}".format(precision_list[i]), \
          "{:10.3f}".format(recall_list[i]), "{:10.3f}".format(f1_list[i]))

confusion_matrix(freq_train_set, freq_test_set, freq_classifier)
```

```
['spam', 'ham', 'spam', 'ham', 'ham', 'ham', 'spam', 'ham', 'spam', 'spam',
'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam', 'ham', 'spam', 'spam',
'ham', 'ham', 'spam', 'spam', 'spam', 'spam', 'ham', 'ham', 'spam', 'spam']
['spam', 'ham', 'spam', 'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam',
'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam', 'ham', 'spam', 'spam',
'ham', 'ham', 'spam', 'spam', 'spam', 'spam', 'ham', 'ham', 'spam', 'spam']
     |   s     |
     |   p   h |
     |   a   a |
     |   m   m |
-----+---------+
spam |<472> 17 |
 ham |  79<332>|
-----+---------+
(row = reference; col = test)

        Precision       Recall          F1
spam          0.965      0.857      0.908
ham           0.808      0.951      0.874
```

[25]:
```
#utilize cross_validation_accuracy function
def cross_validation_accuracy(num_folds, featureset): #take number of folds,␣
 ↪feature set as input
    subset_size = int(len(featureset)/num_folds) #create subsets depending on␣
 ↪folds/feature set size
    print('Each fold size:', subset_size) #display subset size
    accuracy_list = [] #initalize empty accuracy_list

    for i in range(num_folds): #iterate over the folds
        test_this_round = featureset[(i*subset_size):][:subset_size]
        train_this_round = featureset[:(i*subset_size)] +␣
 ↪featureset[((i+1)*subset_size):]
        #train using train_this_round
        classifier = nltk.NaiveBayesClassifier.train(train_this_round)
        #evaluate against test_this_round and save accuracy
        accuracy_this_round = nltk.classify.accuracy(classifier,␣
 ↪test_this_round)
        print (i, accuracy_this_round)
        accuracy_list.append(accuracy_this_round)
    #find mean accuracy over all rounds
    print ('mean accuracy', sum(accuracy_list) / num_folds)

#run 10-fold validation
num_folds = 10
cross_validation_accuracy(num_folds, unfiltered_freq_feature_set)
```

```
Each fold size: 300
0 0.89
1 0.89
2 0.9
3 0.86
4 0.88
5 0.88
6 0.9133333333333333
7 0.8666666666666667
8 0.9033333333333333
9 0.88
mean accuracy 0.8863333333333333
```

[26]:

```
#show the most informative features
print(freq_classifier.show_most_informative_features(20))
```

```
Most Informative Features
             is_freq_hou = True              ham : spam   =    202.1 : 1.0
             is_freq_ect = True              ham : spam   =    126.1 : 1.0
              is_freq_cc = True              ham : spam   =     53.9 : 1.0
              is_freq_pm = True              ham : spam   =     32.3 : 1.0
            is_freq_2000 = True              ham : spam   =     22.8 : 1.0
             is_freq_gas = True              ham : spam   =     18.6 : 1.0
               is_freq_| = True             spam : ham    =     17.7 : 1.0
            is_freq_corp = True              ham : spam   =     15.5 : 1.0
            is_freq_deal = True              ham : spam   =     14.8 : 1.0
            is_freq_http = True             spam : ham    =     11.2 : 1.0
              is_freq_am = True              ham : spam   =      7.7 : 1.0
          is_freq_thanks = True              ham : spam   =      6.6 : 1.0
         is_freq_subject = True              ham : spam   =      5.9 : 1.0
               is_freq_& = True              ham : spam   =      5.7 : 1.0
               is_freq_. = False            spam : ham    =      5.0 : 1.0
               is_freq_* = True             spam : ham    =      4.8 : 1.0
               is_freq_% = True             spam : ham    =      4.2 : 1.0
               is_freq_! = True             spam : ham    =      3.7 : 1.0
            is_freq_know = True               ham : spam    =      3.5 : 1.0
               is_freq_= = True              spam : ham     =      3.3 : 1.0
None
```

[27]:

```
#create POS features function
def pos_features_func(email): #take email as input
    tagged_words = nltk.pos_tag(email) #run pos_tag function on the email to get parts-of-speech
    features = {} #initialize empty dictionary

    numNoun = 0 #set inital counts of parts-of-speech to 0
    numVerb = 0
    numAdj = 0
    numAdverb = 0
    for (word, tag) in tagged_words: #for each word and spam/ham tag in the tagged_words list
        if tag.startswith('N'): numNoun += 1 #add 1 for each POS, depending on first letter
        if tag.startswith('V'): numVerb += 1
        if tag.startswith('J'): numAdj += 1
        if tag.startswith('R'): numAdverb += 1
    features['20+_nouns'] = numNoun > 20 #add POS counts to dictionary, T/F depending on if total count is over 20
    features['20+_verbs'] = numVerb > 20
    features['20+_adjectives'] = numAdj > 20
    features['20+_adverbs'] = numAdverb > 20
    return features
```

[28]:

```
#run pos_features_func function on filtered_emails list
pos_feature_set = [(pos_features_func(email), category)
                   for (email, category) in filtered_emails]

print(len(pos_feature_set[0][0].keys()))
print(pos_feature_set[0])
```

```
4
({'20+_nouns': False, '20+_verbs': False, '20+_adjectives': False,
'20+_adverbs': False}, 'spam')
```

[29]:

```
#train and test the classifier
pos_train_set, pos_test_set = pos_feature_set[thirty_percent:], pos_feature_set[:thirty_percent]

pos_classifier = nltk.NaiveBayesClassifier.train(pos_train_set)

nltk.classify.accuracy(pos_classifier, pos_test_set)
```

[29]: 0.5188888888888888

[30]:
```
#perform 10-fold cross validation
num_folds = 10
cross_validation_accuracy(num_folds, pos_feature_set)
```

```
Each fold size: 300
0 0.5
1 0.5133333333333333
2 0.5166666666666667
3 0.5866666666666667
4 0.5266666666666666
5 0.51
6 0.5133333333333333
7 0.6166666666666667
8 0.5166666666666667
9 0.5366666666666666
mean accuracy 0.5336666666666666
```

[31]:
```
confusion_matrix(pos_train_set, pos_test_set, pos_classifier)
```

```
['spam', 'ham', 'spam', 'ham', 'ham', 'ham', 'spam', 'ham', 'spam', 'spam',
'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam', 'ham', 'spam', 'spam',
'ham', 'ham', 'spam', 'spam', 'spam', 'spam', 'ham', 'ham', 'spam', 'spam']
['ham', 'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'ham', 'ham', 'ham', 'ham',
'ham', 'ham', 'spam', 'ham', 'ham', 'ham', 'spam', 'ham', 'ham', 'spam', 'spam',
'ham', 'spam', 'ham', 'ham', 'ham', 'spam', 'ham', 'ham']
     |    s    |
     |    p  h |
     |    a  a |
     |    m  m |
-----+---------+
spam |<130>359 |
 ham |  74<337>|
-----+---------+
(row = reference; col = test)

        Precision       Recall          F1
spam          0.266      0.637      0.375
ham           0.820      0.484      0.609
```

[32]:
```
#most informative features
print(pos_classifier.show_most_informative_features(20))
```

```
Most Informative Features
          20+_adjectives = True             spam : ham    =      1.9 : 1.0
             20+_adverbs = True             spam : ham    =      1.9 : 1.0
               20+_nouns = False             ham : spam   =      1.3 : 1.0
               20+_nouns = True             spam : ham    =      1.2 : 1.0
          20+_adjectives = False             ham : spam   =      1.2 : 1.0
               20+_verbs = True             spam : ham    =      1.1 : 1.0
```

```
                 20+_verbs = False              ham : spam    =      1.0 : 1.0
               20+_adverbs = False              ham : spam    =      1.0 : 1.0
None
```

[33]:
```
#create bigram features and function

#get top bigrams, save as bg_features
bigrams = list(nltk.bigrams(filtered_words))
#get frequency distribution of bigram
bg_freq_dist = nltk.FreqDist(bigrams)
#save top bigrams
bg_common = bg_freq_dist.most_common(100)

#initialize empty list
bg_features = []

#store top bigrams without counts in bg_features
for bigram in bg_common:
    bg_features.append(bigram[0])

bg_features[:20]
```

[33]:
```
[('daren', 'farmer'),
 ('nbsp', 'nbsp'),
 ('please', 'know'),
 ('north', 'america'),
 ('chokshi', 'corp'),
 ('july', '2000'),
 ('corp', '2000'),
 ('href', 'http'),
 ('america', 'corp'),
 ('looking', 'statements'),
 ('pills', 'pills'),
 ('clynes', 'corp'),
 ('attached', 'file'),
 ('width', 'height'),
 ('would', 'like'),
 ('2000', 'robert'),
 ('forward', 'looking'),
 ('melissa', 'graves'),
 ('forwarded', 'chokshi'),
 ('2000', 'daren')]
```

[34]:
```
def bigram_features(email, bigram_features):
    email_bigrams = nltk.bigrams(email) #get bigrams for each email
    features = {} #inialize empty dictionary
    for bigram in bigram_features: #for each bigram in the email_bigrams list
```

```
        features['bg_{}_{}'.format(bigram[0], bigram[1])] = bigram in␣
↪email_bigrams #add bg_word1_word2, T/F if email has common bigrams

    return features
```

[35]:

```
#run bigram_features() on filtered_emails list
bg_feature_set = [(bigram_features(email, bg_features), category)
                    for (email, category) in filtered_emails]

bg_feature_set[0]
```

[35]:

```
({'bg_daren_farmer': False,
  'bg_nbsp_nbsp': False,
  'bg_please_know': False,
  'bg_north_america': False,
  'bg_chokshi_corp': False,
  'bg_july_2000': False,
  'bg_corp_2000': False,
  'bg_href_http': False,
  'bg_america_corp': False,
  'bg_looking_statements': False,
  'bg_pills_pills': False,
  'bg_clynes_corp': False,
  'bg_attached_file': False,
  'bg_width_height': False,
  'bg_would_like': False,
  'bg_2000_robert': False,
  'bg_forward_looking': False,
  'bg_melissa_graves': False,
  'bg_forwarded_chokshi': False,
  'bg_2000_daren': False,
  'bg_investment_advice': False,
  'bg_font_size': False,
  'bg_anita_luong': False,
  'bg_robert_2000': False,
  'bg_august_2000': False,
  'bg_please_call': False,
  'bg_2000_north': False,
  'bg_robert_cotten': False,
  'bg_questions_please': False,
  'bg_june_2000': False,
  'bg_george_weissman': False,
  'bg_teco_iferc': False,
  'bg_farmer_2000': False,
  'bg_rita_wynne': False,
  'bg_align_center': False,
  'bg_2000_activity': False,
  'bg_thanks_forwarded': False,
  'bg_2000_teco': False,
  'bg_duty_free': False,
  'bg_please_reply': False,
  'bg_aimee_lannou': False,
  'bg_deal_ticket': False,
  'bg_font_family': False,
  'bg_within_email': False,
  'bg_information_provided': False,
  'bg_sitara_deal': False,
  'bg_risks_uncertainties': False,
  'bg_copy_paste': False,
  'bg_make_sure': False,
  'bg_soft_tabs': False,
  'bg_know_questions': False,
  'bg_please_send': False,
  'bg_third_party': False,
  'bg_2000_attached': False,
  'bg_section_securities': False,
  'bg_cotton_valley': False,
  'bg_border_http': False,
  'bg_fuels_cotton': False,
  'bg_meter_1266': False,
  'bg_http_moopid': False,
  'bg_moopid_hotlist': False,
  'bg_march_2000': False,
  'bg_albrecht_well': False,
  'bg_file_hplo': False,
  'bg_hplo_hplo': False,
  'bg_2000_chokshi': False,
  'bg_meter_4179': False,
  'bg_visit_http': False,
  'bg_best_regards': False,
  'bg_allocation_exception': False,
  'bg_adobe_photoshop': False,
  'bg_statements_made': False,
  'bg_forwarded_robert': False,
  'bg_height_font': False,
  'bg_style_line': False,
  'bg_line_height': False,
  'bg_family_knle': False,
  'bg_knle_font': False,
  'bg_mmbtu_mmbtu': False,
  'bg_entered_sitara': False,
  'bg_please_contact': False,
  'bg_deal_tickets': False,
  'bg_without_notice': False,
  'bg_lauri_allen': False,
  'bg_within_report': False,
  'bg_money_back': False,
  'bg_mary_smith': False,
  'bg_email_address': False,
  'bg_current_price': False,
  'bg_stephanie_gomes': False,
  'bg_united_states': False,
  'bg_within_meaning': False,
  'bg_2000_young': False,
  'bg_securities_1933': False,
  'bg_please_note': False,
  'bg_fred_boas': False,
  'bg_reliantenergy_2000': False,
  'bg_acton_corp': False,
  'bg_microsoft_office': False,
  'bg_next_week': False},
 'spam')
```

[36]:

```
#train and test the classifier with 70/30 split
bg_train_set, bg_test_set = bg_feature_set[thirty_percent:], bg_feature_set[:
 ↪thirty_percent]

bg_classifier = nltk.NaiveBayesClassifier.train(bg_train_set)

nltk.classify.accuracy(bg_classifier, bg_test_set)
```

[36]:

```
0.6077777777777778
```

[37]:

```
#most informative bigram features
print(bg_classifier.show_most_informative_features(20))
```

```
Most Informative Features
         bg_daren_farmer = False            spam : ham    =      1.2 : 1.0
        bg_2000_activity = False             ham : spam   =      1.0 : 1.0
        bg_2000_attached = False             ham : spam   =      1.0 : 1.0
         bg_2000_chokshi = False             ham : spam   =      1.0 : 1.0
           bg_2000_daren = False             ham : spam   =      1.0 : 1.0
           bg_2000_north = False             ham : spam   =      1.0 : 1.0
          bg_2000_robert = False             ham : spam   =      1.0 : 1.0
            bg_2000_teco = False             ham : spam   =      1.0 : 1.0
           bg_2000_young = False             ham : spam   =      1.0 : 1.0
           bg_acton_corp = False             ham : spam   =      1.0 : 1.0
      bg_adobe_photoshop = False             ham : spam   =      1.0 : 1.0
         bg_aimee_lannou = False             ham : spam   =      1.0 : 1.0
        bg_albrecht_well = False             ham : spam   =      1.0 : 1.0
         bg_align_center = False             ham : spam   =      1.0 : 1.0
 bg_allocation_exception = False             ham : spam   =      1.0 : 1.0
```

```
         bg_america_corp = False           ham : spam   =      1.0 : 1.0
          bg_anita_luong = False           ham : spam   =      1.0 : 1.0
        bg_attached_file = False           ham : spam   =      1.0 : 1.0
          bg_august_2000 = False           ham : spam   =      1.0 : 1.0
         bg_best_regards = False           ham : spam   =      1.0 : 1.0
None
```

[38]:

```
confusion_matrix(bg_train_set, bg_test_set, bg_classifier)
```

```
['spam', 'ham', 'spam', 'ham', 'ham', 'ham', 'spam', 'ham', 'spam', 'spam',
'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam', 'ham', 'spam', 'spam',
'ham', 'ham', 'spam', 'spam', 'spam', 'spam', 'ham', 'ham', 'spam', 'spam']
['spam', 'ham', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam',
'ham', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam', 'spam']
     |   s     |
     |   p   h |
     |   a   a |
     |   m   m |
-----+---------+
spam |<489>  . |
 ham | 353 <58>|
-----+---------+
(row = reference; col = test)

        Precision       Recall          F1
spam          1.000      0.581      0.735
ham           0.141      1.000      0.247
```

[39]:

```
cross_validation_accuracy(num_folds, bg_feature_set)
```

```
Each fold size: 300
0 0.6066666666666667
1 0.63
2 0.5866666666666667
3 0.5333333333333333
4 0.58
5 0.6
6 0.5966666666666667
7 0.5066666666666667
8 0.5866666666666667
9 0.56
mean accuracy 0.5786666666666667
```

[40]:

```
#combine frequency, pos, and bigram features

#take email and list of bigrams as input
def all_features(email, word_features, bigram_features):
    email_words = set(email) #tokenize email
    email_bigrams = nltk.bigrams(email) #get email bigrams
    tagged_words = nltk.pos_tag(email) #run pos_tag function on the email to␣
↪get parts-of-speech
    features = {}
    for word in word_features:
        features['is_freq_{}'.format(word)] = (word in email_words)
    for bigram in bigram_features:
        features['bg_{}_{}'.format(bigram[0], bigram[1])] = bigram in␣
↪email_bigrams
    numNoun = 0 #set inital counts of parts-of-speech to 0
    numVerb = 0
    numAdj = 0
    numAdverb = 0
    for (word, tag) in tagged_words: #for each word and spam/ham tag in the␣
↪tagged_words list
        if tag.startswith('N'): numNoun += 1 #add 1 for each POS, depending on␣
↪first letter
        if tag.startswith('V'): numVerb += 1
        if tag.startswith('J'): numAdj += 1
        if tag.startswith('R'): numAdverb += 1
    features['20+_nouns'] = numNoun > 20 #add POS counts to dictionary, T/F␣
↪depending on if total count is over 20
    features['20+_verbs'] = numVerb > 20
    features['20+_adjectives'] = numAdj > 20
    features['20+_adverbs'] = numAdverb > 20
    return features
```

[41]:

```
all_feature_set = [(all_features(email, freq_words_only, bg_features), category)
                   for (email, category) in filtered_emails]

all_feature_set[0]
```

[41]:

```
({'is_freq_2000': False,
  'is_freq_please': True,
  'is_freq_meter': False,
  'is_freq_deal': False,
  'is_freq_corp': False,
  'is_freq_http': False,
  'is_freq_company': False,
  'is_freq_thanks': False,
  'is_freq_know': False,
  'is_freq_information': False,
  'is_freq_forwarded': False,
  'is_freq_need': False,
  'is_freq_daren': False,
  'is_freq_price': False,
  'is_freq_time': False,
  'is_freq_mmbtu': False,
  'is_freq_email': False,
  'is_freq_would': False,
  'is_freq_robert': False,
  'is_freq_font': False,
  'is_freq_mail': False,
  'is_freq_sitara': False,
  'is_freq_report': False,
  'is_freq_statements': False,
  'is_freq_month': False,
  'is_freq_july': False,
  'is_freq_attached': False,
  'is_freq_also': False,
  'is_freq_contract': False,
  'is_freq_energy': False,
  'is_freq_like': False,
  'is_freq_farmer': False,
  'is_freq_free': False,
  'is_freq_nbsp': False,
  'is_freq_volume': False,
  'is_freq_deals': False,
  'is_freq_business': False,
  'is_freq_message': False,
  'is_freq_want': False,
  'is_freq_questions': False,
  'is_freq_contact': False,
  'is_freq_make': False,
  'is_freq_change': False,
  'is_freq_volumes': False,
  'is_freq_within': False,
  'is_freq_height': False,
  'is_freq_production': False,
  'is_freq_call': False,
  'is_freq_well': False,
  'is_freq_could': False,
  'is_freq_stock': False,
  'is_freq_forward': False,
  'is_freq_back': False,
  'is_freq_today': False,
  'is_freq_line': False,
  'is_freq_number': False,
  'is_freq_list': False,
  'is_freq_ticket': False,
  'is_freq_following': False,
  'is_freq_money': False,
  'is_freq_order': False,
  'is_freq_pills': False,
  'is_freq_size': False,
  'is_freq_width': False,
  'is_freq_best': False,
  'is_freq_click': True,
  'is_freq_take': False,
  'is_freq_first': False,
  'is_freq_2004': False,
  'is_freq_texas': False,
  'is_freq_investment': False,
  'is_freq_looking': False,
  'is_freq_online': False,
  'is_freq_days': False,
  'is_freq_available': False,
  'is_freq_system': False,
  'is_freq_file': False,
  'is_freq_products': False,
  'is_freq_america': False,
  'is_freq_june': False,
  'is_freq_flow': False,
  'is_freq_securities': False,
  'is_freq_future': False,
  'is_freq_sent': False,
  'is_freq_next': False,
  'is_freq_north': False,
  'is_freq_effective': False,
  'is_freq_management': False,
  'is_freq_sale': False,
  'is_freq_product': False,
  'is_freq_chokshi': False,
  'is_freq_service': False,
  'is_freq_group': False,
  'is_freq_help': False,
  'is_freq_made': False,
  'is_freq_prices': False,
  'is_freq_services': False,
  'is_freq_office': False,
  'is_freq_many': False,
  'is_freq_based': False,
  'bg_daren_farmer': False,
  'bg_nbsp_nbsp': False,
  'bg_please_know': False,
  'bg_north_america': False,
  'bg_chokshi_corp': False,
  'bg_july_2000': False,
  'bg_corp_2000': False,
  'bg_href_http': False,
  'bg_america_corp': False,
  'bg_looking_statements': False,
  'bg_pills_pills': False,
  'bg_clynes_corp': False,
  'bg_attached_file': False,
  'bg_width_height': False,
  'bg_would_like': False,
  'bg_2000_robert': False,
  'bg_forward_looking': False,
  'bg_melissa_graves': False,
  'bg_forwarded_chokshi': False,
  'bg_2000_daren': False,
  'bg_investment_advice': False,
  'bg_font_size': False,
  'bg_anita_luong': False,
  'bg_robert_2000': False,
  'bg_august_2000': False,
  'bg_please_call': False,
  'bg_2000_north': False,
  'bg_robert_cotten': False,
  'bg_questions_please': False,
  'bg_june_2000': False,
  'bg_george_weissman': False,
  'bg_teco_iferc': False,
  'bg_farmer_2000': False,
  'bg_rita_wynne': False,
  'bg_align_center': False,
  'bg_2000_activity': False,
  'bg_thanks_forwarded': False,
  'bg_2000_teco': False,
  'bg_duty_free': False,
  'bg_please_reply': False,
  'bg_aimee_lannou': False,
  'bg_deal_ticket': False,
  'bg_font_family': False,
  'bg_within_email': False,
  'bg_information_provided': False,
  'bg_sitara_deal': False,
  'bg_risks_uncertainties': False,
  'bg_copy_paste': False,
  'bg_make_sure': False,
  'bg_soft_tabs': False,
  'bg_know_questions': False,
  'bg_please_send': False,
  'bg_third_party': False,
  'bg_2000_attached': False,
  'bg_section_securities': False,
  'bg_cotton_valley': False,
  'bg_border_http': False,
  'bg_fuels_cotton': False,
  'bg_meter_1266': False,
  'bg_http_moopid': False,
  'bg_moopid_hotlist': False,
  'bg_march_2000': False,
  'bg_albrecht_well': False,
  'bg_file_hplo': False,
  'bg_hplo_hplo': False,
  'bg_2000_chokshi': False,
  'bg_meter_4179': False,
  'bg_visit_http': False,
  'bg_best_regards': False,
  'bg_allocation_exception': False,
  'bg_adobe_photoshop': False,
  'bg_statements_made': False,
  'bg_forwarded_robert': False,
  'bg_height_font': False,
  'bg_style_line': False,
  'bg_line_height': False,
  'bg_family_knle': False,
  'bg_knle_font': False,
  'bg_mmbtu_mmbtu': False,
  'bg_entered_sitara': False,
  'bg_please_contact': False,
  'bg_deal_tickets': False,
  'bg_without_notice': False,
  'bg_lauri_allen': False,
  'bg_within_report': False,
  'bg_money_back': False,
  'bg_mary_smith': False,
  'bg_email_address': False,
  'bg_current_price': False,
  'bg_stephanie_gomes': False,
  'bg_united_states': False,
  'bg_within_meaning': False,
  'bg_2000_young': False,
  'bg_securities_1933': False,
  'bg_please_note': False,
  'bg_fred_boas': False,
  'bg_reliantenergy_2000': False,
  'bg_acton_corp': False,
  'bg_microsoft_office': False,
  'bg_next_week': False,
  '20+_nouns': False,
  '20+_verbs': False,
  '20+_adjectives': False,
  '20+_adverbs': False},
 'spam')
```

[42]:
```
#train and test a new classifier with 70/30 split
all_train_set, all_test_set = all_feature_set[thirty_percent:],␣
 ↪all_feature_set[:thirty_percent]

all_classifier = nltk.NaiveBayesClassifier.train(all_train_set)

nltk.classify.accuracy(all_classifier, all_test_set)
```

[42]:
```
0.9255555555555556
```

[43]:
```
#cross-validation
cross_validation_accuracy(num_folds, all_feature_set)
```

```
Each fold size: 300
0 0.9266666666666666
1 0.92
2 0.9133333333333333
3 0.9166666666666666
4 0.91
5 0.9133333333333333
6 0.9266666666666666
7 0.91
8 0.9333333333333333
9 0.9333333333333333
mean accuracy 0.9203333333333333
```

[44]:
```
#most important features
print(all_classifier.show_most_informative_features(20))
```

```
Most Informative Features
          is_freq_farmer = True              ham : spam   =     48.4 : 1.0
        is_freq_attached = True              ham : spam   =     24.1 : 1.0
            is_freq_2000 = True              ham : spam   =     22.8 : 1.0
      is_freq_securities = True             spam : ham    =     18.3 : 1.0
          is_freq_robert = True              ham : spam   =     17.0 : 1.0
           is_freq_money = True             spam : ham    =     16.5 : 1.0
            is_freq_corp = True              ham : spam   =     15.5 : 1.0
            is_freq_deal = True              ham : spam   =     14.8 : 1.0
            is_freq_size = True             spam : ham    =     14.0 : 1.0
          is_freq_prices = True             spam : ham    =     13.7 : 1.0
            is_freq_july = True              ham : spam   =     13.5 : 1.0
      is_freq_investment = True             spam : ham    =     13.2 : 1.0
          is_freq_volume = True              ham : spam   =     12.1 : 1.0
            is_freq_http = True             spam : ham    =     11.2 : 1.0
      is_freq_statements = True              spam : ham    =     10.2 : 1.0
       is_freq_questions = True               ham : spam   =      9.2 : 1.0
           is_freq_texas = True               ham : spam   =      9.2 : 1.0
            is_freq_june = True               ham : spam   =      9.1 : 1.0
          is_freq_ticket = True               ham : spam   =      9.0 : 1.0
            is_freq_file = True               ham : spam   =      8.0 : 1.0
None
```

[45]:
```
#confusion matrix
confusion_matrix(all_train_set, all_test_set, all_classifier)
```

```
['spam', 'ham', 'spam', 'ham', 'ham', 'ham', 'spam', 'ham', 'spam', 'spam',
'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam', 'ham', 'spam', 'spam',
'ham', 'ham', 'spam', 'spam', 'spam', 'spam', 'ham', 'ham', 'spam', 'spam']
['spam', 'ham', 'spam', 'spam', 'ham', 'ham', 'spam', 'ham', 'spam', 'spam',
'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam', 'spam', 'spam', 'spam',
'ham', 'ham', 'spam', 'spam', 'spam', 'spam', 'spam', 'ham', 'spam', 'spam']
     |   s     |
     |   p   h |
     |   a   a |
     |   m   m |
-----+---------+
spam |<487>  2 |
 ham |  65<346>|
-----+---------+
(row = reference; col = test)

        Precision       Recall          F1
spam          0.996      0.882      0.936
ham           0.842      0.994      0.912
```

[46]:
```
#get top trigrams
trigrams = list(nltk.trigrams(filtered_words))

tri_freq_dist = nltk.FreqDist(trigrams)

tri_common = tri_freq_dist.most_common(100)

tri_features = []

for trigram in tri_common:
    tri_features.append(trigram[0])

tri_features[:20]
```

[46]:
```
[('nbsp', 'nbsp', 'nbsp'),
 ('north', 'america', 'corp'),
 ('forwarded', 'chokshi', 'corp'),
 ('forward', 'looking', 'statements'),
 ('chokshi', 'corp', '2000'),
 ('2000', 'daren', 'farmer'),
 ('2000', 'north', 'america'),
 ('daren', 'farmer', '2000'),
 ('pills', 'pills', 'pills'),
 ('http', 'moopid', 'hotlist'),
 ('fuels', 'cotton', 'valley'),
 ('attached', 'file', 'hplo'),
 ('file', 'hplo', 'hplo'),
 ('2000', 'teco', 'iferc'),
 ('2000', 'attached', 'file'),
 ('style', 'line', 'height'),
 ('line', 'height', 'font'),
 ('height', 'font', 'family'),
 ('font', 'family', 'knle'),
 ('family', 'knle', 'font')]
```

[47]:

```
#define trigram features function, takes and trigram features list as input
def trigram_features(email, trigram_features):
    email_trigrams = nltk.trigrams(email)
    features = {}

    for trigram in trigram_features:
        features['tri_{}_{}_{}'.format(trigram[0], trigram[1], trigram[2])] =␣
 ↪trigram in email_trigrams

    return features
```

[48]:

```
#run trigram_features() on filtered_emails list
tri_feature_set = [(trigram_features(email, tri_features), category)
                    for (email, category) in filtered_emails]

tri_feature_set[0]
```

[48]:

```
({'tri_nbsp_nbsp_nbsp': False,
  'tri_north_america_corp': False,
  'tri_forwarded_chokshi_corp': False,
  'tri_forward_looking_statements': False,
  'tri_chokshi_corp_2000': False,
  'tri_2000_daren_farmer': False,
  'tri_2000_north_america': False,
  'tri_daren_farmer_2000': False,
  'tri_pills_pills_pills': False,
  'tri_http_moopid_hotlist': False,
  'tri_fuels_cotton_valley': False,
  'tri_attached_file_hplo': False,
  'tri_file_hplo_hplo': False,
  'tri_2000_teco_iferc': False,
  'tri_2000_attached_file': False,
  'tri_style_line_height': False,
  'tri_line_height_font': False,
  'tri_height_font_family': False,
  'tri_font_family_knle': False,
  'tri_family_knle_font': False,
  'tri_knle_font_size': False,
  'tri_reliantenergy_2000_chokshi': False,
  'tri_jebel_duty_free': False,
  'tri_meter_1266_july': False,
  'tri_july_2000_activity': False,
  'tri_questions_please_call': False,
  'tri_statements_within_meaning': False,
  'tri_created_entered_sitara': False,
  'tri_1266_july_2000': False,
  'tri_2000_activity_allocation': False,
  'tri_activity_allocation_exception': False,
  'tri_2000_robert_2000': False,
  'tri_height_href_http': False,
  'tri_george_weissman_2000': False,
  'tri_http_0424040_ftar': False,
  'tri_would_like_removed': False,
  'tri_inherent_conflict_interest': False,
  'tri_aimee_lannou_2000': False,
  'tri_2000_robert_cotten': False,
  'tri_plain_text_format': False,
  'tri_robert_cotten_2000': False,
  'tri_meter_0986725_production': False,
  'tri_coastal_corporation_albrecht': False,
  'tri_corporation_albrecht_well': False,
  'tri_albrecht_well_meter': False,
  'tri_well_meter_4179': False,
  'tri_meter_4179_goliad': False,
  'tri_moopid_hotlist_images': False,
  'tri_receive_special_offers': False,
  'tri_reason_would_like': False,
  'tri_reply_remove_line': False,
  'tri_logos_trademarks_property': False,
  'tri_trademarks_property_respective': False,
  'tri_8834464_prices_dollars': False,
  'tri_prices_dollars_works': False,
  'tri_duty_free_zone': False,
  'tri_prices_availability_change': False,
  'tri_special_offers_plain': False,
  'tri_offers_plain_text': False,
  'tri_text_format_reply': False,
  'tri_format_reply_mail': False,
  'tri_reply_mail_request': False,
  'tri_email_considered_spam': False,
  'tri_considered_spam_long': False,
  'tri_spam_long_include': False,
  'tri_long_include_contact': False,
  'tri_include_contact_information': False,
  'tri_contact_information_remove': False,
  'tri_information_remove_instructions': False,
  'tri_remove_instructions_message': False,
  'tri_instructions_message_intended': False,
  'tri_message_intended_dealer': False,
  'tri_intended_dealer_resellers': False,
  'tri_dealer_resellers_somehow': False,
  'tri_resellers_somehow_gotten': False,
  'tri_somehow_gotten_list': False,
  'tri_gotten_list_error': False,
  'tri_list_error_reason': False,
  'tri_error_reason_would': False,
  'tri_like_removed_please': False,
  'tri_removed_please_reply': False,
  'tri_please_reply_remove': False,
  'tri_remove_line_message': False,
  'tri_line_message_message': False,
  'tri_message_message_sent': False,
  'tri_message_sent_compliance': False,
  'tri_sent_compliance_federal': False,
  'tri_compliance_federal_legislation': False,
  'tri_federal_legislation_commercial': False,
  'tri_legislation_commercial_mail': False,
  'tri_commercial_mail_4176': False,
  'tri_mail_4176_section': False,
  'tri_4176_section_paragraph': False,
  'tri_section_paragraph_bill': False,
  'tri_paragraph_bill_1618': False,
  'tri_bill_1618_title': False,
  'tri_1618_title_passed': False,
  'tri_title_passed_congress': False,
  'tri_passed_congress_logos': False,
  'tri_congress_logos_trademarks': False},
 'spam')
```

[49]:

```
#train and test the classifier with 70/30 split
tri_train_set, tri_test_set = tri_feature_set[thirty_percent:],␣
 ↪tri_feature_set[:thirty_percent]
```

```
tri_classifier = nltk.NaiveBayesClassifier.train(tri_train_set)

nltk.classify.accuracy(tri_classifier, tri_test_set)
```

[49]: 0.46

[50]:
```
#most informative bigram features
print(tri_classifier.show_most_informative_features(20))
```

```
Most Informative Features
      tri_nbsp_nbsp_nbsp = False             ham : spam   =      1.0 : 1.0
      tri_1266_july_2000 = False             ham : spam   =      1.0 : 1.0
   tri_1618_title_passed = False             ham : spam   =      1.0 : 1.0
tri_2000_activity_allocation = False             ham : spam   =      1.0 : 1.0
  tri_2000_attached_file = False             ham : spam   =      1.0 : 1.0
   tri_2000_daren_farmer = False             ham : spam   =      1.0 : 1.0
  tri_2000_north_america = False             ham : spam   =      1.0 : 1.0
    tri_2000_robert_2000 = False             ham : spam   =      1.0 : 1.0
  tri_2000_robert_cotten = False             ham : spam   =      1.0 : 1.0
     tri_2000_teco_iferc = False             ham : spam   =      1.0 : 1.0
tri_4176_section_paragraph = False             ham : spam   =      1.0 : 1.0
tri_8834464_prices_dollars = False             ham : spam   =      1.0 : 1.0
tri_activity_allocation_exception = False             ham : spam   =      1.0 :
1.0
   tri_aimee_lannou_2000 = False             ham : spam   =      1.0 : 1.0
 tri_albrecht_well_meter = False             ham : spam   =      1.0 : 1.0
  tri_attached_file_hplo = False             ham : spam   =      1.0 : 1.0
     tri_bill_1618_title = False             ham : spam   =      1.0 : 1.0
   tri_chokshi_corp_2000 = False             ham : spam   =      1.0 : 1.0
tri_coastal_corporation_albrecht = False             ham : spam   =      1.0 :
1.0
tri_commercial_mail_4176 = False             ham : spam   =      1.0 : 1.0
None
```

[51]:
```
confusion_matrix(tri_train_set, tri_test_set, tri_classifier)
```

```
['spam', 'ham', 'spam', 'ham', 'ham', 'ham', 'spam', 'ham', 'spam', 'spam',
'spam', 'ham', 'ham', 'ham', 'ham', 'spam', 'spam', 'ham', 'spam', 'spam',
'ham', 'ham', 'spam', 'spam', 'spam', 'spam', 'ham', 'ham', 'spam', 'spam']
['ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham',
'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham',
'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham', 'ham']
     |   s     |
     |   p   h |
     |   a   a |
     |   m   m |
-----+---------+
spam |  <3>486 |
 ham |    .<411>|
-----+---------+
(row = reference; col = test)

        Precision       Recall          F1
spam          0.006      1.000      0.012
ham           1.000      0.458      0.628
```

[52]:
```
cross_validation_accuracy(num_folds, tri_feature_set)
```

```
Each fold size: 300
0 0.47
1 0.4266666666666667
2 0.48333333333333334
3 0.43666666666666665
4 0.5033333333333333
5 0.47333333333333333
6 0.49666666666666665
7 0.4266666666666667
8 0.48333333333333334
9 0.51
mean accuracy 0.471
```

[57]:
```
import sys
import nltk
import random

# for testing, allow different sizes for word features
vocab_size = 100

# Function writeFeatureSets:
# takes featuresets defined in the nltk and convert them to weka input csv file
#    any feature value in the featuresets should not contain ",", "'" or "␣
 ↪itself
#    and write the file to the outpath location
#    outpath should include the name of the csv file
def writeFeatureSets(featuresets, outpath):
    # open outpath for writing
    f = open(outpath, 'w')
    # get the feature names from the feature dictionary in the first featureset
    featurenames = featuresets[0][0].keys()
    # create the first line of the file as comma separated feature names
    #    with the word class as the last feature name
    featurenameline = ''
    for featurename in featurenames:
        # replace forbidden characters with text abbreviations
        featurename = featurename.replace(',','CM')
        featurename = featurename.replace("'","DQ")
        featurename = featurename.replace('"','QU')
        featurenameline += featurename + ','
    featurenameline += 'class'
    # write this as the first line in the csv file
    f.write(featurenameline)
    f.write('\n')
    # convert each feature set to a line in the file with comma separated␣
 ↪feature values,
    # each feature value is converted to a string
    #   for booleans this is the words true and false
    #   for numbers, this is the string with the number
    for featureset in featuresets:
        featureline = ''
        for key in featurenames:
            featureline += str(featureset[0][key]) + ','
        featureline += featureset[1]
        # write each feature set values to the file
        f.write(featureline)
        f.write('\n')
    f.close()

# define features (keywords) of a document for a BOW/unigram baseline
# each feature is 'contains(keyword)' and is true or false depending
# on whether that keyword is in the document
def document_features(document, word_features):
    document_words = set(document)
    features = {}
    for word in word_features:
        features['is_freq_{}'.format(word)] = (word in document_words)
    return features

# Main program to produce movie review feature sets in order to show how to use
#   the writeFeatureSets function
if __name__ == '__main__':
    # Make a list of command line arguments, omitting the [0] element
    # which is the script itself.
    args = sys.argv[1:]
    if not args:
        print ('usage: python save_features.py [file]')
        sys.exit(1)
    outpath = args[0]


    # get features sets for a document, including keyword features and category␣
 ↪feature
    featuresets = freq_feature_set
    # write the feature sets to the csv file
    writeFeatureSets(featuresets, outpath)

    print ('Wrote spam/ham features to:', outpath)
```

```
Wrote spam/ham features to: -f
```

[58]:

```
# function to read features, perform cross-validation with (several)␣
↪classifiers and report results

import sys
import pandas
import numpy
from sklearn import preprocessing
from sklearn.svm import LinearSVC
from sklearn.naive_bayes import MultinomialNB
from sklearn.model_selection import cross_val_predict
from sklearn.naive_bayes import GaussianNB
from sklearn.metrics import classification_report
from sklearn.metrics import confusion_matrix
from sklearn.linear_model import LogisticRegression

def process(filepath):
    # number of folds for cross-validation
    kFolds = 10

    # read in the file with the pandas package
    train_set = pandas.read_csv(filepath)

    # this is a data frame for the data
    print ('Shape of feature data - num instances with num features + class␣
↪label')
    print (train_set.shape)

    # convert to a numpy array for sklearn
    train_array = train_set.values

    # get the last column with the class labels into a vector y
    train_y = train_array[:,-1]

    # get the remaining rows and columns into the feature matrix X
    train_X = train_array[:,:-1]

    print('** Results from Naive Bayes')
    classifier = MultinomialNB()

    y_pred = cross_val_predict(classifier, train_X, train_y, cv=kFolds)
    # classification report compares predictions from the k fold test sets with␣
↪the gold
    print(classification_report(train_y, y_pred))

    # confusion matrix from same
    cm = confusion_matrix(train_y, y_pred)
    #print_cm(cm, labels)
    print('\n')
    print(pandas.crosstab(train_y, y_pred, rownames=['Actual'],␣
↪colnames=['Predicted'], margins=True))


# use a main so can get feature file as a command line argument
if __name__ == '__main__':
    # Make a list of command line arguments, omitting the [0] element
    # which is the script itself.
    args = sys.argv[1:]
    if not args:
        print ('usage: python run_sklearn_model_performance.py [featurefile]')
        sys.exit(1)
    infile = args[0]
    process(infile)
```

```
Shape of feature data - num instances with num features + class label
(3000, 101)
** Results from Naive Bayes
              precision    recall  f1-score   support

         ham       0.86      0.91      0.88      1500
        spam       0.91      0.85      0.88      1500

    accuracy                           0.88      3000
   macro avg       0.88      0.88      0.88      3000
weighted avg       0.88      0.88      0.88      3000



Predicted   ham  spam   All
Actual
ham        1368   132  1500
spam        229  1271  1500
All        1597  1403  3000
```

[ ]: