DMG2 Assignment : Problem 6

Naive Bayes Text Classifier

Number of classes : 20

In each class, there are a number of documents, each one corresponding to a date. The test-train split will be based on the date.

Preprocessing in each document :

- Keep only From, Subject, Host, Organization, Data
- Remove special characters, stop words
- Stem the words
- There are numbers in the data, as addresses, phone numbers, currency, etc. Should they be removed?

In [1]:
```
import os,re
import pandas as pd
import numpy as np
import nltk,unicodedata
import operator,math
import inflect

from nltk.corpus import stopwords
from nltk.stem import WordNetLemmatizer
from nltk.stem.lancaster import LancasterStemmer
from sklearn.feature_extraction.text import CountVectorizer
```

Reading Files

In [2]:
```
DATA_DIR = '/home/jishnu/Documents/ISB/Term3/dmg2/assignments/hw_assignment
1/dmg2/datasets/20_newsgroups'
```

In [3]:
```
labels,files_list = [],[]
for root, dirs, files in os.walk(DATA_DIR):
    for file in files:
        labels.append(re.sub(r'/home/jishnu/Documents/ISB/Term3/dmg2/assignm
ents/hw_assignment1/dmg2/datasets/20_newsgroups/','',root))
        files_list.append(os.path.join(root,file))
```

In [4]:
```
files_df = pd.DataFrame({'filename':files_list, 'label' : labels})
list(files_df['label'].unique())
```

Out[4]:
```
['comp.graphics',
 'talk.religion.misc',
 'rec.sport.baseball',
 'comp.sys.ibm.pc.hardware',
 'rec.motorcycles',
 'talk.politics.guns',
 'misc.forsale',
 'alt.atheism',
 'talk.politics.misc',
 'talk.politics.mideast',
 'rec.autos',
 'comp.windows.x',
 'sci.crypt',
 'sci.electronics',
 'soc.religion.christian',
 'rec.sport.hockey',
 'comp.os.ms-windows.misc',
 'sci.space',
 'comp.sys.mac.hardware',
 'sci.med']
```

Train - Test Split

For each class, splitting the documents to training and test based on a 70-30 rule.

In [5]:
```
train = pd.DataFrame(columns=['filename','label'])
test = pd.DataFrame(columns=['filename','label'])

for label in list(files_df['label'].unique()):
    threshold = files_df.loc[files_df['label'] == label].shape[0] * 0.7
    threshold = int(np.floor(threshold))
    train = train.append(files_df.loc[files_df['label'] == label].iloc[:thre
shold,:],ignore_index=True)
    test = test.append(files_df.loc[files_df['label'] == label].iloc[thresho
ld:,:],ignore_index=True)

print(train.shape[0],test.shape[0])
```

```
13997 6000
```

Creating dictionary of 5000 most frequent words in each class

Calculating P(W|C) for each word in each class, by normalizing using Laplace smoothing parameter of 30.

Here, CountVectorizer class from scikit-learn has been used to create the Document-Term Matrix. The class has an inbuilt preprocessing module. After calculating the document term matrix, the counts of document in which each word occurs has been calculated to find the most frequent ones for each class. Then, the probability of word given class has been calculated for the top 5000 words. Laplace smoothing parameter of 30 has been used when calculating P(W|C).

In [6]:
```
def remove_non_ascii(words):
    """Remove non-ASCII characters from list of tokenized words"""
    new_words = []
    for word in words:
```

```
        new_word = unicodedata.normalize('NFKD', word).encode('ascii', 'igno
re').decode('utf-8', 'ignore')
        new_words.append(new_word)
    return new_words

def to_lowercase(words):
    """Convert all characters to lowercase from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = word.lower()
        new_words.append(new_word)
    return new_words

def remove_punctuation(words):
    """Remove punctuation from list of tokenized words"""
    new_words = []
    for word in words:
        new_word = re.sub(r'[^\w\s]', '', word)
        if new_word != '':
            new_words.append(new_word)
    return new_words

def replace_numbers(words):
    """Replace all interger occurrences in list of tokenized words with text
ual representation"""
    p = inflect.engine()
    new_words = []
    for word in words:
        if word.isdigit():
            pass
            #new_word = p.number_to_words(word)
            #new_words.append(new_word)
        else:
            new_words.append(word)
    return new_words

def remove_stopwords(words):
    """Remove stop words from list of tokenized words"""
    new_words = []
    for word in words:
        if word not in stopwords.words('english'):
            new_words.append(word)
    return new_words

def stem_words(words):
    """Stem words in list of tokenized words"""
    stemmer = LancasterStemmer()
    stems = []
    for word in words:
        stem = stemmer.stem(word)
        stems.append(stem)
    return stems

def lemmatize_verbs(words):
    """Lemmatize verbs in list of tokenized words"""
    lemmatizer = WordNetLemmatizer()
    lemmas = []
    for word in words:
        lemma = lemmatizer.lemmatize(word, pos='v')
        lemmas.append(lemma)
    return lemmas

def normalize(words):
    words = remove_non_ascii(words)
    words = to_lowercase(words)
    words = remove_punctuation(words)
```

```
    words = replace_numbers(words)
    words = remove_stopwords(words)
    return words
```

In [7]:
```
# Dictionary to hold vectorizer objects
vect_dict = {}
# Dictionary to hold Document term matrix for each class.
# The document term matrix is converted to a Pandas DataFrame
class_dict = {}
for label in list(train['label'].unique()):
    # List to hold words for each label
    class_words_preprocessed = []
    for filename in train.loc[train['label'] == label]['filename']:
        with open(filename,'r',errors='ignore') as filein:
            data = filein.read()
        words = re.split('\W+',data)
        words = normalize(words)
        class_words_preprocessed.append(' '.join(words))
    vect_dict[label] = CountVectorizer(input='content',analyzer='word',decod
e_error='ignore')
    class_dict[label] = pd.DataFrame(vect_dict[label].fit_transform(class_wo
rds_preprocessed).todense().T)
    class_dict[label]['count_docs'] = class_dict[label].sum(axis=1)
    class_dict[label]['word'] = vect_dict[label].get_feature_names()
    class_dict[label] = class_dict[label].sort_values(by='count_docs',ascend
ing=False).iloc[:5000,:]
    tot_freq = class_dict[label]['count_docs'].sum()
    class_dict[label]['p(w|c)'] =  (class_dict[label]['count_docs'] + 30) /
(tot_freq + (5000 * 30))
```

In [8]:
```
# # Dictionary to hold vectorizer objects
# vect_dict = {}
# # Dictionary to hold Document term matrix for each class.
# # The document term matrix is converted to a Pandas DataFrame
# class_dict = {}
# for label in list(train['label'].unique()):
#     vect_dict[label] = CountVectorizer(input='filename',analyzer='word',st
op_words='english',decode_error='ignore')
#     class_dict[label] = pd.DataFrame(vect_dict[label].fit_transform(list(t
rain.loc[train['label'] == label]['filename'])).todense().T)
#     class_dict[label]['count_docs'] = class_dict[label].sum(axis=1)
#     class_dict[label]['word'] = vect_dict[label].get_feature_names()
#     class_dict[label] = class_dict[label].sort_values(by='count_docs',asce
nding=False).iloc[:5000,:]
#     tot_freq = class_dict[label]['count_docs'].sum()
#     class_dict[label]['p(w|c)'] =  (class_dict[label]['count_docs'] + 30)
 / (tot_freq + (5000 * 30))
```

Considering the top 25 most frequent words for all labels, are there any words which occur in all the documents?

In [9]:
```
top25_list = []
for label in list(train['label'].unique()):
    top25_list.append(class_dict[label].iloc[:25,:]['word'])
intersect = set(top25_list[0])
for list_ in top25_list[1:]:
    intersect.intersection_update(list_)
print(intersect)
```

```
{'srv', 'edu', 'cmu', 'message', 'cs', 'net', 'com', 'subject', 'cantaloup
e'}
```

Removing these words from each dictionary and recalculating probabilities

In [10]:
```
for label in list(train['label'].unique()):
    class_dict[label] = class_dict[label].loc[~ class_dict[label]['word'].is
in(list(intersect))]
for label in list(train['label'].unique()):
    tot_freq = class_dict[label]['count_docs'].sum()
    class_dict[label]['p(w|c)'] =  (class_dict[label]['count_docs'] + 30) /
(tot_freq + (5000 * 30))
```

In [11]:
```
# Final Word Dictionary for each class
for label in list(train['label'].unique()):
    class_dict[label] = pd.Series(class_dict[label]['p(w|c)'].values,index=c
lass_dict[label]['word']).to_dict()
```

In [12]:
```
#class_dict['comp.graphics']
```

The words **cmu, edu,com,cs** can be removed for better results

Calculating Class Priors

In [13]:
```
class_priors_dict = {}
total_freq = 0
for label in list(files_df['label'].unique()):
    class_priors_dict[label] = files_df.loc[files_df['label'] == label].shap
e[0]
    total_freq += class_priors_dict[label]
for label in list(files_df['label'].unique()):
    class_priors_dict[label] = np.round(class_priors_dict[label] / total_fre
q, 4)
```

In [14]:
```
class_priors_dict
```

Out[14]:
```
{'comp.graphics': 0.050000000000000003,
 'talk.religion.misc': 0.050000000000000003,
 'rec.sport.baseball': 0.050000000000000003,
 'comp.sys.ibm.pc.hardware': 0.050000000000000003,
 'rec.motorcycles': 0.050000000000000003,
 'talk.politics.guns': 0.050000000000000003,
 'misc.forsale': 0.050000000000000003,
 'alt.atheism': 0.050000000000000003,
 'talk.politics.misc': 0.050000000000000003,
 'talk.politics.mideast': 0.050000000000000003,
 'rec.autos': 0.050000000000000003,
 'comp.windows.x': 0.050000000000000003,
 'sci.crypt': 0.050000000000000003,
 'sci.electronics': 0.050000000000000003,
 'soc.religion.christian': 0.0499,
 'rec.sport.hockey': 0.050000000000000003,
 'comp.os.ms-windows.misc': 0.050000000000000003,
 'sci.space': 0.050000000000000003,
 'comp.sys.mac.hardware': 0.050000000000000003,
 'sci.med': 0.050000000000000003}
```

Calculating Training Accuracy

In [15]:
```
train_predicted = pd.DataFrame(columns=['predicted','max_class_posterior_pro
b'])
for train_doc in list(train['filename']):
    with open(train_doc,'r',errors='ignore') as filein:
            data = filein.read()
    words = re.split('\W+',data)
    words = normalize(words)
    log_posterior_dict = class_priors_dict
    log_posterior_dict = dict([(k,math.log(v)) for (k,v) in log_posterior_di
ct.items()])
    for word in words:
        for k,v in log_posterior_dict.items():
            try:
                log_posterior_dict[k] = log_posterior_dict[k] + math.log(cla
ss_dict[k][word])
            except:
                pass
    log_posterior_dict = dict([(k,np.exp(v)) for (k,v) in
log_posterior_dict.items()])
    train_predicted = train_predicted.append({'predicted':max(log_posterior_
dict, key=log_posterior_dict.get),'max_class_posterior_prob':max(log_posteri
or_dict.values())},ignore_index=True)
train_predicted['actual'] = train['label']
print('Training Accuracy : {}'.format(np.round(train_predicted.loc[train_pre
dicted['predicted'] == train_predicted['actual']].shape[0]/train_predicted.s
hape[0],4)))
```

```
Training Accuracy : 0.0119
```

In [16]:
```
# train_predicted = pd.DataFrame(columns=['predicted','max_class_posterior_p
rob'])
# for train_doc in list(train['filename']):
#     vect_train = CountVectorizer(input='filename',analyzer='word',stop_wor
ds='english',decode_error='ignore')
#     train_docterm = pd.DataFrame(vect_train.fit_transform([train_doc]).tod
ense().T)
#     #print(vect_train.get_feature_names())
#     log_posterior_dict = class_priors_dict
#     log_posterior_dict = dict([(k,math.log(v)) for (k,v) in log_posterior_
dict.items()])
#     for word in vect_train.get_feature_names():
#         for k,v in log_posterior_dict.items():
#             try:
#                 log_posterior_dict[k] = log_posterior_dict[k] + math.log(c
lass_dict[k][word])
#             except:
#                 pass
#     log_posterior_dict = dict([(k,np.exp(v)) for (k,v) in log_posterior_di
ct.items()])
#     train_predicted = train_predicted.append({'predicted':max(log_posterio
r_dict, key=log_posterior_dict.get),'max_class_posterior_prob':max(log_poste
rior_dict.values())},ignore_index=True)
# train_predicted['actual'] = train['label']
# print('Training Accuracy : {}'.format(np.round(train_predicted.loc[train_p
redicted['predicted'] == train_predicted['actual']].shape[0]/train_predicte
d.shape[0],4)))
```

Calculating Test Accuracy

In [17]:
```
test_predicted = pd.DataFrame(columns=['predicted','max_class_posterior_pro
b'])
for test_doc in list(test['filename']):
    with open(test_doc,'r',errors='ignore') as filein:
            data = filein.read()
    words = re.split('\W+',data)
    words = normalize(words)
    log_posterior_dict = class_priors_dict
    log_posterior_dict = dict([(k,math.log(v)) for (k,v) in log_posterior_di
ct.items()])
    for word in words:
        for k,v in log_posterior_dict.items():
            try:
                log_posterior_dict[k] = log_posterior_dict[k] + math.log(cla
ss_dict[k][word])
            except:
                pass
    log_posterior_dict = dict([(k,np.exp(v)) for (k,v) in
log_posterior_dict.items()])
    test_predicted = test_predicted.append({'predicted':max(log_posterior_di
ct, key=log_posterior_dict.get),'max_class_posterior_prob':max(log_posterior
_dict.values())},ignore_index=True)
test_predicted['actual'] = test['label']
print('Testing Accuracy : {}'.format(np.round(test_predicted.loc[test_predic
ted['predicted'] ==
test_predicted['actual']].shape[0]/test_predicted.shape[0],4)))
```

```
Testing Accuracy : 0.0128
```

It is seen that the training and test accuracy are 1.19% and 1.28% when using dictionary of 5000 words.

Creating dictionary of 10,000 most frequent words in each class

In [18]:
```
# Dictionary to hold vectorizer objects
vect_dict = {}
# Dictionary to hold Document term matrix for each class.
# The document term matrix is converted to a Pandas DataFrame
class_dict = {}
for label in list(train['label'].unique()):
    # List to hold words for each label
    class_words_preprocessed = []
    for filename in train.loc[train['label'] == label]['filename']:
        with open(filename,'r',errors='ignore') as filein:
            data = filein.read()
        words = re.split('\W+',data)
        words = normalize(words)
        class_words_preprocessed.append(' '.join(words))
    vect_dict[label] = CountVectorizer(input='content',analyzer='word',decod
e_error='ignore')
    class_dict[label] = pd.DataFrame(vect_dict[label].fit_transform(class_wo
rds_preprocessed).todense().T)
    class_dict[label]['count_docs'] = class_dict[label].sum(axis=1)
    class_dict[label]['word'] = vect_dict[label].get_feature_names()
    class_dict[label] = class_dict[label].sort_values(by='count_docs',ascend
ing=False).iloc[:10000,:]
    tot_freq = class_dict[label]['count_docs'].sum()
    class_dict[label]['p(w|c)'] =  (class_dict[label]['count_docs'] + 30) /
(tot_freq + (10000 * 30))
```

In [19]:
```
# # Dictionary to hold vectorizer objects
# vect_dict = {}
# # Dictionary to hold Document term matrix for each class.
# # The document term matrix is converted to a Pandas DataFrame
# class_dict = {}
# for label in list(train['label'].unique()):
#     vect_dict[label] = CountVectorizer(input='filename',analyzer='word',st
op_words='english',decode_error='ignore')
#     class_dict[label] = pd.DataFrame(vect_dict[label].fit_transform(list(t
rain.loc[train['label'] == label]['filename'])).todense().T)
#     class_dict[label]['count_docs'] = class_dict[label].sum(axis=1)
#     class_dict[label]['word'] = vect_dict[label].get_feature_names()
#     class_dict[label] = class_dict[label].sort_values(by='count_docs',asce
nding=False).iloc[:10000,:]
#     tot_freq = class_dict[label]['count_docs'].sum()
#     class_dict[label]['p(w|c)'] =  (class_dict[label]['count_docs'] + 30)
 / (tot_freq + (10000 * 30))
```

In [20]:
```
top25_list = []
for label in list(train['label'].unique()):
    top25_list.append(class_dict[label].iloc[:25,:]['word'])
intersect = set(top25_list[0])
for list_ in top25_list[1:]:
    intersect.intersection_update(list_)
print(intersect)
```

```
{'srv', 'edu', 'cmu', 'message', 'cs', 'net', 'com', 'subject', 'cantaloup
e'}
```

Removing these words from each dictionary and recalculating probabilities

In [21]:
```
for label in list(train['label'].unique()):
    class_dict[label] = class_dict[label].loc[~ class_dict[label]['word'].is
in(list(intersect))]
for label in list(train['label'].unique()):
    tot_freq = class_dict[label]['count_docs'].sum()
    class_dict[label]['p(w|c)'] =  (class_dict[label]['count_docs'] + 30) /
(tot_freq + (5000 * 30))
```

In [22]:
```
# Final Word Dictionary for each class
for label in list(train['label'].unique()):
    class_dict[label] = pd.Series(class_dict[label]['p(w|c)'].values,index=c
lass_dict[label]['word']).to_dict()
```

Calculating Training Accuracy

In [23]:
```
train_predicted = pd.DataFrame(columns=['predicted','max_class_posterior_pro
b'])
for train_doc in list(train['filename']):
    with open(train_doc,'r',errors='ignore') as filein:
            data = filein.read()
    words = re.split('\W+',data)
    words = normalize(words)
    log_posterior_dict = class_priors_dict
    log_posterior_dict = dict([(k,math.log(v)) for (k,v) in log_posterior_di
ct.items()])
    for word in words:
        for k,v in log_posterior_dict.items():
            try:
                log_posterior_dict[k] = log_posterior_dict[k] + math.log(cla
ss_dict[k][word])
            except:
                pass
    log_posterior_dict = dict([(k,np.exp(v)) for (k,v) in
log_posterior_dict.items()])
    train_predicted = train_predicted.append({'predicted':max(log_posterior_
dict, key=log_posterior_dict.get),'max_class_posterior_prob':max(log_posteri
or_dict.values())},ignore_index=True)
train_predicted['actual'] = train['label']
print('Training Accuracy : {}'.format(np.round(train_predicted.loc[train_pre
dicted['predicted'] == train_predicted['actual']].shape[0]/train_predicted.s
hape[0],4)))
```

```
Training Accuracy : 0.0162
```

Calculating Test Accuracy

In [24]:
```
test_predicted = pd.DataFrame(columns=['predicted','max_class_posterior_pro
b'])
for test_doc in list(test['filename']):
    with open(test_doc,'r',errors='ignore') as filein:
            data = filein.read()
    words = re.split('\W+',data)
    words = normalize(words)
    log_posterior_dict = class_priors_dict
    log_posterior_dict = dict([(k,math.log(v)) for (k,v) in log_posterior_di
ct.items()])
    for word in words:
        for k,v in log_posterior_dict.items():
            try:
                log_posterior_dict[k] = log_posterior_dict[k] + math.log(cla
ss_dict[k][word])
            except:
                pass
    log_posterior_dict = dict([(k,np.exp(v)) for (k,v) in
log_posterior_dict.items()])
    test_predicted = test_predicted.append({'predicted':max(log_posterior_di
ct, key=log_posterior_dict.get),'max_class_posterior_prob':max(log_posterior
_dict.values())},ignore_index=True)
test_predicted['actual'] = test['label']
print('Testing Accuracy : {}'.format(np.round(test_predicted.loc[test_predic
ted['predicted'] ==
test_predicted['actual']].shape[0]/test_predicted.shape[0],4)))
```

```
Testing Accuracy : 0.0175
```

It is seen that increasing the dictionary to 10,000 increases the training and test accuracies to 1.62% and 1.75%.