In [41]:

```
import os
import re
from pathlib import Path
from nltk.tokenize import RegexpTokenizer
from collections import Counter
import pandas as pd
from sklearn.linear_model import LogisticRegression
from sklearn.naive_bayes import GaussianNB, BernoulliNB, MultinomialNB
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score, confusion_matrix
from sklearn.exceptions import ConvergenceWarning
import numpy as np
from scipy import stats
from scipy import sparse
import matplotlib
import matplotlib.pyplot as plt
%matplotlib inline
import random
random.seed()
import warnings
warnings.filterwarnings("ignore", category=FutureWarning)
warnings.filterwarnings("ignore", category=ConvergenceWarning)
```

In [42]:

```
DATA_DIR = "20_newsgroups"
```

Functions from lab

In [43]:

```
# Cleaning up of text files to remove the following strings and replacing them with empty quotes
def clean_file_text(text):
    new_text = re.sub("Newsgroups:.*?\n", "", text)
    new_text = re.sub("Xref:.*?\n", "", new_text)
    new_text = re.sub("Path:.*?\n", "", new_text)
    new_text = re.sub("Date:.*?\n", "", new_text)
    new_text = re.sub("Followup-To:.*?\n", "", new_text)
    return new_text
```

In [44]:

```
# returns a counter collection of all the words in each of the files.
def corpus_count_words(file_list):
    tokenizer = RegexpTokenizer(r'\w+')
    word_counter = Counter()
    for file_path in file_list:
        with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
            file_data = file.read()
            file_data = clean_file_text(file_data)
            file_words = tokenizer.tokenize(file_data)
            word_counter.update(file_words)
    return word_counter
```

In [45]:

```
# Returns the topic names
def get_topic_name(file_path):
    return file_path.parent.name

# Returns in the index that matches the topic names- returns a number from 0-19
def get_target(topic_name):
    topics = ['talk.politics.mideast', 'rec.autos', 'comp.sys.mac.hardware', 'alt.atheism', 'rec.sp
ort.baseball',
     'comp.os.ms-windows.misc', 'rec.sport.hockey', 'sci.crypt', 'sci.med', 'talk.politics.misc',
     'rec.motorcycles', 'comp.windows.x', 'comp.graphics', 'comp.sys.ibm.pc.hardware', 'sci.electro
nics',
```

```
    'talk.politics.guns', 'sci.space', 'soc.religion.christian', 'misc.forsale',
'talk.religion.misc']
    return topics.index(topic_name)
```

In [46]:

```
def plot_confusion_matrix(cm):
    # plot the confusion matrix
    plt.figure(figsize=(10,10))
    plt.matshow(cm, fignum=1)

    # add labels for all targets
    num_targets = cm.shape[0]
    plt.xticks(list(range(num_targets+1)))
    plt.yticks(list(range(num_targets+1)))
```

Q1: Binary Encoding

In [47]:

```
all_files = [pth for pth in Path(DATA_DIR).glob("**/*") if pth.is_file() and not pth.name.startswit
h(".")]
```

In [48]:

```
def binary_baseline_data(file_list, num_words = 1000):
    # Calculate word count in corpus
    news_cnt = corpus_count_words(file_list)
    print (news_cnt.most_common(10))

    # Select the most common 1000 words from all ~20000 documents
    word_list = [word for (word, freq) in news_cnt.most_common(num_words)]

"""
Create a binary encoding of dataset based on the selected features (X) such that if the word exist
ed in "file_word"
the value was to be 1 and if the word existed in the entire collection ("word_list"), then the val
ue was to be 0
"""

    tokenizer = RegexpTokenizer(r'\w+')
    df_rows = []
    for file_path in file_list:
        with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
            file_data = file.read()
            file_data = clean_file_text(file_data)
            file_words = tokenizer.tokenize(file_data)
            df_rows.append([1 if word in file_words else 0 for word in word_list])
    X = pd.DataFrame(df_rows, columns = word_list)

    # Create a dataframe of targets (y) - where y are the topic indices from 0 - 19 that are prede
fined in the dataset given
    y = [get_target(get_topic_name(file_path)) for file_path in file_list]

    return X, y
```

In [49]:

```
# get the baseline data
X, y = binary_baseline_data(all_files)

# split to train and test set
X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

# train a logistic regression classifier
clf = LogisticRegression(C=1.0).fit(X_train, y_train)

# predict on train and test set
y_train_predict = clf.predict(X_train)
y_test_predict = clf.predict(X_test)

# calculate train and test accuracy
```

```
# calculate train and test accuracy
train_accuracy = accuracy_score(y_train, y_train_predict)
test_accuracy = accuracy_score(y_test, y_test_predict)

# report results
print("Train accuracy: {}".format(train_accuracy))
print("Test accuracy: {}".format(test_accuracy))
```

```
[('the', 227359), ('to', 126510), ('of', 120164), ('a', 100558), ('and', 96676), ('I', 89455), ('i
s', 72055), ('in', 68295), ('that', 67484), ('AX', 62406)]
Train accuracy: 0.9124812459812817
Test accuracy: 0.6978333333333333
```

Top ten popular words with their frequency:
[('the', 227359), ('to', 126510), ('of', 120164), ('a', 100558), ('and', 96676), ('I', 89455), ('is', 72055), ('in', 68295), ('that', 67484), ('AX', 62406)]
Train accuracy: 0.9124812459812817
Test accuracy: 0.6978333333333333

Q1 (a): please describe the feature set, the amount of data, and the hyper-parameters used in this baseline

Feature set: top 1000 words in the most common words in news_cnt counter.
The Amount of data: 70% of the entire data is used for the training set and 30% is used for the test set
Hyperparameter: for Logistic Regression is set to 1.0 (C)-inverse regularization coefficient

Q1 (b)

Modify the following function:

In [242]:

```
from whoosh.analysis import *
```

In [243]:

```
nltk.download("stopwords")
```

```
[nltk_data] Downloading package stopwords to
[nltk_data]     /Users/krutheekarajkumar/nltk_data...
[nltk_data]   Package stopwords is already up-to-date!
```

Out[243]:

```
True
```

In [244]:

```
def corpus_count_words_improved(file_list):
    tokenizer = RegexpTokenizer(r'\w+')
    word_counter = Counter()
    my_analyzer=RegexTokenizer() | LowercaseFilter() | IntraWordFilter() | StopFilter() | StemFilte
r()
    for file_path in file_list:
        with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
            file_data = file.read()
            file_data = clean_file_text(file_data)
            file_words=[acb.text for acb in my_analyzer(file_data)]
            word_counter.update(file_words)
    return word_counter
```

In [246]:

```
"""
The feature set was improved by removing the applying a lowercase filter, removing stop words ther
e by eliminating
common words that do not add any value to the document, stemfilter - which returns the root word o
f the feature that
```

```
views concatenated words as two seperate words.
"""
def binary_improved_data(file_list, num_words = 1000):

    news_cnt = corpus_count_words_improved(file_list)
    # Printing 10 of the 1000 common words post tokenization of the corpus.
    print (news_cnt.most_common(10))

    word_list = [word for (word, freq) in news_cnt.most_common(num_words)]
    my_analyzer = RegexTokenizer() | LowercaseFilter() | IntraWordFilter() | StopFilter() | StemFil
ter()

    df_rows = []

    for file_path in file_list:
        with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
            file_data = file.read()
            file_data = clean_file_text(file_data)
            file_words=[acb.text for acb in my_analyzer(file_data)]

        df_rows.append([1 if word in file_words else 0 for word in word_list])

    X = pd.DataFrame(df_rows, columns = word_list)

    y = [get_target(get_topic_name(file_path)) for file_path in file_list]


    # validate return types
    assert isinstance(X, pd.DataFrame) and isinstance(y, list), "return types"

    return X, y
```

Q1 (c)

Modify the following partial code to calculate the train and test accuracy and answer the question in the markdown cell below

In [54]:

```
# get the improved data after text processing
X_improved, y_improved = binary_improved_data(all_files)
```

```
[('edu', 66031), ('ax', 62713), ('com', 36790), ('wa', 24738), ('1993', 23326), ('but', 23275), ('
thei', 23217), ('line', 23215), ('new', 23100), ('apr', 22837)]
```

In [247]:

```
X_improved
```

Out[247]:

	edu	ax	com	wa	1993	but	thei	line	new	apr	...	armi	entri	wasn	outsid	chanc	select	basebal	score	48	knowled
0	1	0	1	1	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	
1	1	0	0	1	1	1	1	1	1	0	...	0	0	0	1	1	0	0	0	0	
2	1	0	0	1	0	1	1	1	1	0	...	0	1	0	0	0	0	0	0	0	
3	1	0	0	1	0	1	1	1	1	0	...	0	0	0	0	0	0	0	0	0	
4	0	1	1	0	1	0	1	1	0	1	...	0	0	0	0	0	0	0	0	0	
5	1	0	1	1	1	1	1	1	0	1	...	0	1	0	0	0	0	0	0	0	
6	1	0	0	1	1	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	
7	1	0	0	1	0	1	1	1	0	0	...	0	1	0	0	0	0	0	0	0	
8	1	0	1	1	0	1	0	1	0	0	...	0	0	0	0	0	0	0	0	0	
9	1	0	0	0	1	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	
10	1	0	1	0	1	1	0	1	0	0	...	0	0	0	0	0	0	0	0	0	
11	1	0	1	1	1	1	0	1	0	0	...	0	0	0	0	0	0	0	0	0	
12	1	0	1	0	1	1	0	1	0	0	...	0	0	0	0	0	0	0	0	0	

13	edu	ax	com	wa	1993	but	thei	line	new	apr	...	arm	entr	wasn	outsid	chang	select	basebal	score	48	knowledg
14	1	0	1	0	0	1	0	1	0	0	...	0	0	0	0	0	0	0	0	0	
15	1	0	1	1	0	1	0	1	0	0	...	0	0	0	0	0	0	0	0	0	
16	1	0	1	0	0	1	1	1	0	0	...	0	0	0	0	0	0	0	0	0	
17	0	0	1	0	0	1	1	1	0	0	...	0	0	0	0	0	0	0	0	0	
18	1	0	1	0	1	1	1	1	0	1	...	0	0	0	0	0	0	0	0	0	
19	1	0	1	0	1	1	1	1	0	1	...	0	0	1	0	0	0	0	0	0	
20	1	0	1	1	0	1	0	1	1	0	...	0	0	0	0	0	0	0	0	0	
21	1	0	1	0	0	1	0	1	1	0	...	0	0	0	0	0	0	0	0	0	
22	1	0	1	0	0	1	0	1	1	0	...	0	0	0	0	0	1	0	0	0	
23	1	0	1	0	0	1	0	1	1	0	...	0	0	0	0	0	0	0	0	0	
24	1	0	0	0	1	0	0	1	1	1	...	0	0	0	0	1	0	0	0	0	
25	1	0	1	1	0	1	1	1	0	0	...	0	0	0	0	0	0	0	0	0	
26	1	0	1	0	1	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	
27	1	0	0	0	1	1	0	1	1	1	...	0	0	0	0	0	0	0	0	0	
28	1	0	1	1	0	0	0	1	1	0	...	0	0	0	0	0	0	0	0	0	
29	1	0	1	0	1	0	1	1	0	1	...	0	0	0	0	0	0	0	0	0	
...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	...	
19967	1	0	1	1	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	
19968	1	0	0	1	1	1	1	1	1	1	...	0	0	0	1	1	0	0	0	0	
19969	1	0	1	0	1	0	0	1	0	1	...	0	0	0	0	0	0	0	0	0	
19970	1	0	0	0	0	0	1	1	1	0	...	0	0	0	0	0	0	0	0	0	
19971	1	0	1	1	0	0	1	1	0	0	...	0	0	0	0	0	0	0	0	0	
19972	0	0	1	1	0	1	1	1	0	0	...	0	0	0	0	0	0	0	0	0	
19973	1	0	1	1	0	0	0	1	0	1	...	0	0	0	0	0	0	0	0	0	
19974	1	0	0	0	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	
19975	1	0	0	1	1	1	1	1	1	1	...	0	0	1	0	1	0	0	0	0	
19976	1	0	1	1	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	
19977	1	0	1	1	1	1	0	1	1	1	...	0	0	0	0	0	0	0	0	0	
19978	1	0	1	1	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	
19979	1	0	0	0	0	1	0	1	1	1	...	0	0	0	0	0	0	0	0	0	
19980	1	0	1	1	1	1	1	1	0	1	...	0	0	1	0	0	0	0	0	0	
19981	1	0	0	0	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	
19982	1	0	1	1	1	1	1	1	0	1	...	0	1	0	1	0	0	0	0	0	
19983	0	0	1	0	0	1	1	1	0	0	...	0	0	0	0	0	0	0	0	0	
19984	1	0	0	0	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	
19985	1	0	1	1	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	
19986	1	0	1	0	1	0	0	1	0	1	...	0	0	0	0	0	0	0	0	0	
19987	1	0	0	1	1	1	1	1	1	1	...	0	0	0	0	0	0	0	0	0	
19988	1	0	0	0	0	1	0	1	1	0	...	0	0	0	0	0	0	0	0	0	
19989	0	0	0	1	1	1	0	1	1	1	...	0	0	0	0	0	0	0	0	0	
19990	1	0	0	1	1	1	1	1	1	1	...	0	0	1	0	1	0	0	0	0	
19991	1	0	1	1	1	1	1	1	0	1	...	0	0	0	0	0	0	0	0	0	
19992	0	0	0	1	0	1	0	1	1	0	...	0	0	0	0	0	0	0	0	0	
19993	0	0	1	1	1	0	1	1	1	1	...	0	0	0	0	0	0	0	0	0	
19994	1	0	1	1	0	0	1	1	0	0	...	0	0	0	0	0	0	0	0	0	
19995	1	0	1	0	1	0	1	1	1	1	...	0	0	0	0	0	1	0	0	0	
19996	1	0	1	1	0	0	0	1	0	0	...	0	0	0	0	0	0	0	0	0	

19997 rows × 1000 columns

In [248]:

```
# Splitting train and test dataset-7:3
X_train, X_test, y_train, y_test = train_test_split(X_improved, y_improved, test_size=0.3, random_s
tate=42)

# train a logistic regression classifier
clf = LogisticRegression(C=1.0).fit(X_train, y_train)

# predict on train and test set
y_train_predict = clf.predict(X_train)
y_test_predict = clf.predict(X_test)

# calculate train and test accuracy
train_accuracy = accuracy_score(y_train, y_train_predict)
test_accuracy = accuracy_score(y_test, y_test_predict)

# report results
print("Train accuracy: {}".format(train_accuracy))
print("Test accuracy: {}".format(test_accuracy))
```

```
Train accuracy: 0.9336286347074373
Test accuracy: 0.7376666666666667
```

Top 10 popular words and their frequencies:
[('edu', 66031), ('ax', 62713), ('com', 36790), ('wa', 24738), ('1993', 23326), ('but', 23275), ('thei', 23217), ('line', 23215), ('new', 23100), ('apr', 22837)]
Train accuracy: 0.9336286347074373
Test accuracy: 0.7376666666666667

Q1 (c) How did the result change:

The train and test accuracy increased after the features were improved and this is due to the elimination of data points that produced noise thereby skewing the classifier. Previously the top ten of the thousand features were stop words that added no real value to the definition of the classifier, therefore when the classifier was applied to the test dataset, there the error was higher-there by decreasing the accuracy.

In [57]:

```
print("X_train shape: ",X_train.shape)
print("X_test shape: ",X_test.shape)
print("y_train shape: ",np.shape(y_train))
print("y_test shape: ",np.shape(y_test))
print("x_improved shape: ",X_improved.shape[1])
```

```
X_train shape:  (13997, 1000)
X_test shape:  (6000, 1000)
y_train shape:  (13997,)
y_test shape:  (6000,)
x_improved shape:  1000
```

Q1 (d) Random-mean with 95% confidence interval

Code modified below

In [58]:

```
def random_mean_ci(X, y, num_tests):

    train_results=[]
    test_results = []
    # 10 random integers in the range of 0-999 are stored in rs
    rs= np.random.randint(1000, size=num_tests)

    for i in range(num_tests):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=rs[i]
```

```
)
        clf = LogisticRegression(C=1.0).fit(X_train, y_train)
        y_train_predict = clf.predict(X_train)
        y_test_predict = clf.predict(X_test)
        train_accuracy = accuracy_score(y_train, y_train_predict)
        test_accuracy = accuracy_score(y_test, y_test_predict)
    # train_results is a list of train accuracy results for the differrent random splits of the da
taset
        train_results.append(train_accuracy)

    # test_results is a list of test accuracy results for the differrent random splits of the data
set
        test_results.append(test_accuracy)

    # calculate the train mean and the 95% confidence interval for the list of results
    train_mean = np.mean(train_results)
    train_ci_low, train_ci_high = stats.t.interval(0.95, len(train_results)-1, loc=train_mean, scal
e=stats.sem(train_results))

    # calculate the test mean and the 95% confidence interval for the list of results
    test_mean = np.mean(test_results)
    test_ci_low, test_ci_high = stats.t.interval(0.95, len(test_results)-1, loc=test_mean, scale=st
ats.sem(test_results))

    # validate return types
    assert isinstance(train_mean, float) and isinstance(train_ci_low, float) and isinstance(train_c
i_high, float), "return types"
    assert isinstance(test_mean, float) and isinstance(test_ci_low, float) and isinstance(test_ci_h
igh, float), "return types"

    return train_mean, train_ci_low, train_ci_high, test_mean, test_ci_low, test_ci_high
```

In [59]:

```
train_mean10, train_low10, train_high10, test_mean10, test_low10, test_high10 = random_mean_ci(X_im
proved, y_improved, num_tests = 10)
print("Train mean accuracy over 10 random splits: {}".format(train_mean10))
print("Train confidence interval over 10 random splits: [{}, {}]".format(train_low10, train_high10
))
print("Test mean accuracy over 10 random splits: {}".format(test_mean10))
print("Test confidence interval over 10 random splits: [{}, {}]".format(test_low10, test_high10))
```

```
Train mean accuracy over 10 random splits: 0.9348646138458239
Train confidence interval over 10 random splits: [0.9342970070217598, 0.935432220669888]
Test mean accuracy over 10 random splits: 0.7391499999999999
Test confidence interval over 10 random splits: [0.7361949173196005, 0.7421050826803992]
```

Q1 (e) Are these results more or less informative than the single trial ?

The average of the dataset would would take into consideration the outliers and blackswan events as well, however 95% of the confidence interval is preferred as it is more informative as to where 95% of the data point occur.

The mean accuracies marginally increased from the previous function. When a single number for a random value is chosen, it is ensured that the same subsection of the dataset is taken and studied - this ensure consistency in the results. However, when an array of random numbers are chosen and their effect on the dataset is studied as an average, a comprehensive understanding of the model and the results are presented.

Q1 (f): Producing a confusion matrix:

In [60]:

```
def random_cm(X, y, num_tests):
    # cm_list is a list of confusion matrices for the different random splits of the dataset
    cm_list = []

    rs= np.random.randint(1000, size=num_tests)
    #print(rs)
    for i in range(num_tests):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=rs[i]
)
        clf = LogisticRegression(C=1.0).fit(X_train, y_train)
```

```
        y_train_predict = clf.predict(X_train)
        y_test_predict = clf.predict(X_test)

        cm_list.append(confusion_matrix(y_test, y_test_predict))
    # sum the confusion matrices and return the combined confusion matrix
    combined_cm = pd.Panel(cm_list).sum(axis=0)

    # validate return type
    assert isinstance(combined_cm, pd.DataFrame), "return type"

    return combined_cm
```

Q1 (g) Study of the results of the confusion matrix:

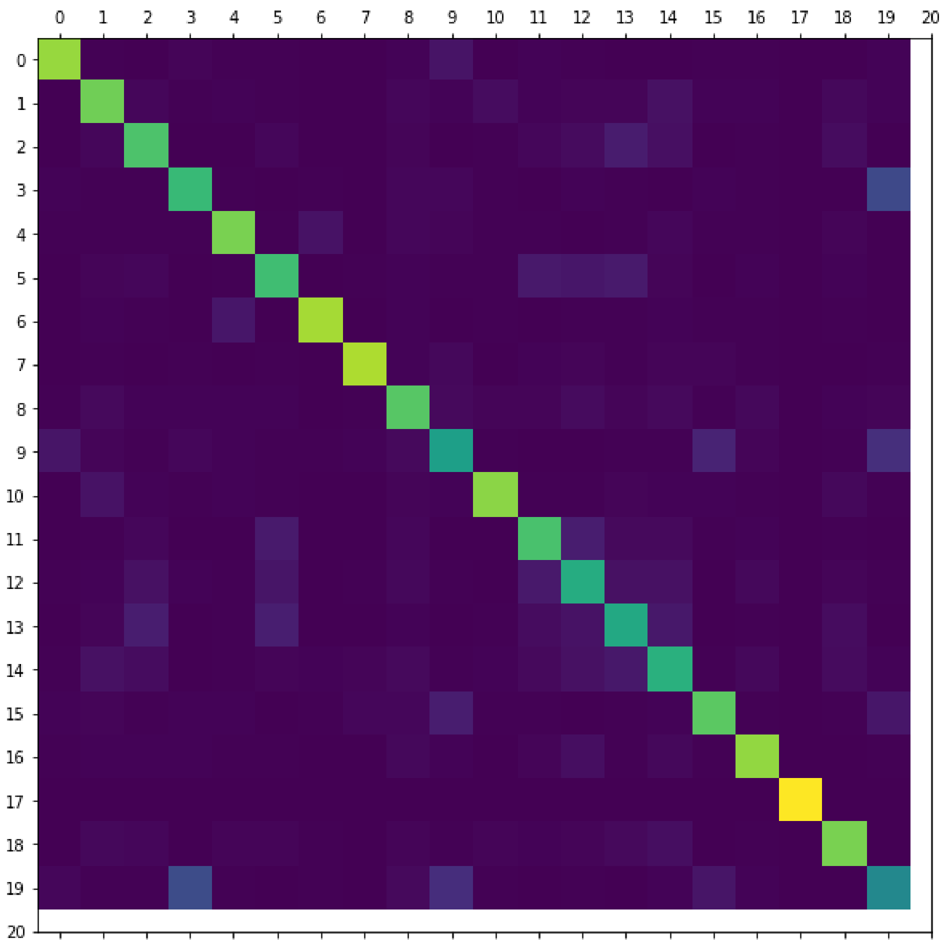
Use the following code to produce a confusion matrix for 10 random splits

In [61]:

```
cm10 = random_cm(X_improved, y_improved, num_tests = 10)
plot_confusion_matrix(cm10)
```

```
/Users/krutheekarajkumar/anaconda3/lib/python3.6/site-packages/ipykernel_launcher.py:1:
DeprecationWarning:
Panel is deprecated and will be removed in a future version.
The recommended way to represent these types of 3-dimensional data are with a MultiIndex on a
DataFrame, via the Panel.to_frame() method
Alternatively, you can use the xarray package http://xarray.pydata.org/en/stable/.
Pandas provides a `.to_xarray()` method to help automate this conversion.

  """Entry point for launching an IPython kernel.
```



In [73]:

```
pd.DataFrame(cm10)
```

Out[73]:

	0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19
0	2536	15	5	40	20	12	9	2	31	156	4	24	22	3	8	23	15	0	3	30
1	11	2343	51	17	35	17	9	10	49	31	94	25	39	41	129	33	25	0	64	27
2	2	48	2157	9	8	54	9	10	41	7	16	56	90	227	123	7	21	6	100	4
3	26	16	13	2016	15	4	18	1	58	47	19	5	31	13	10	35	23	1	2	660
4	12	22	12	12	2392	22	146	8	47	38	21	16	8	12	49	19	20	0	38	8
5	1	36	52	3	13	2078	6	14	32	12	14	207	177	221	46	6	25	0	27	7
6	3	32	19	4	179	6	2591	3	31	11	14	7	13	22	30	18	20	0	23	8
7	2	14	10	13	5	22	4	2631	31	59	10	28	45	22	43	39	21	0	8	18
8	16	72	32	26	31	25	4	21	2223	72	39	46	89	36	81	22	65	1	24	43
9	174	45	18	56	34	12	23	30	78	1687	13	11	11	12	16	298	36	2	13	403
10	3	144	31	13	28	18	10	10	37	29	2478	21	12	39	29	33	15	2	59	21
11	10	15	54	5	8	211	11	11	50	6	7	2133	241	81	73	15	27	0	18	6
12	16	29	129	24	17	166	8	16	70	21	11	205	1855	140	139	4	62	4	38	14
13	5	41	243	11	16	249	7	10	24	6	18	98	146	1809	196	5	19	0	99	9
14	15	132	102	9	21	43	25	45	79	17	29	72	134	197	1903	16	62	2	85	24
15	28	37	21	26	26	6	13	50	48	240	20	20	8	17	35	2240	19	0	15	181
16	15	27	25	29	15	15	8	7	68	34	10	39	117	23	68	20	2513	0	7	13
17	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	2996	1	0
18	8	70	56	2	37	43	14	11	36	13	39	30	43	74	107	9	13	0	2395	9
19	48	12	17	701	14	7	14	11	76	396	17	9	23	5	27	170	32	5	10	1398

Yes, there are some documents that are easily confused with each other, documents 19 and 3 are such examples. There were 701 documents incorrectly identified as document 3 and similarly, 660 documents incorrectly identified as document 19. Document 3 is in reference to "alt.atheism" and document 19 is regarding "talk.religion.misc", it is concievable that these two topic discuss a lot of overlapping contents.

Q2: Number of Features

Q2 (a)

In [74]:

```
def feature_num(X, y):
    # result_list is a list of tuples (num_features, train_accuracy, test_accuracy)

    result_list = []

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)

    for p in [0.1, 0.2, 0.4, 0.6, 0.8, 1.0]:
    # subset size ensures the size is a fraction of the actual size of X
        subset_size = int(p*X.shape[1])
    # the training subset is selected based on the position of the subset-size starting from posit
ion 0
        X_train_subset = X_train.t[:, 0:subset_size]
        X_test_subset = X_test.iloc[:, 0:subset_size]
        clf = LogisticRegression(C=1.0).fit(X_train_subset, y_train)
        y_train_predict = clf.predict(X_train_subset)
        y_test_predict = clf.predict(X_test_subset)
        train_accuracy = accuracy_score(y_train, y_train_predict)
        test_accuracy = accuracy_score(y_test, y_test_predict)
        # add to result_list
        result_list.append((p, train_accuracy, test_accuracy))

    # Make a dataframe of the results
    result_df = pd.DataFrame(result_list, columns=["num_features", "train_accuracy", "test_accuracy
"])
```

```python
    # validate return type
    assert isinstance(result_df, pd.DataFrame), "return type"

    return result_df
```

Q2 (b) Plotting the accuracy VS number of features curve

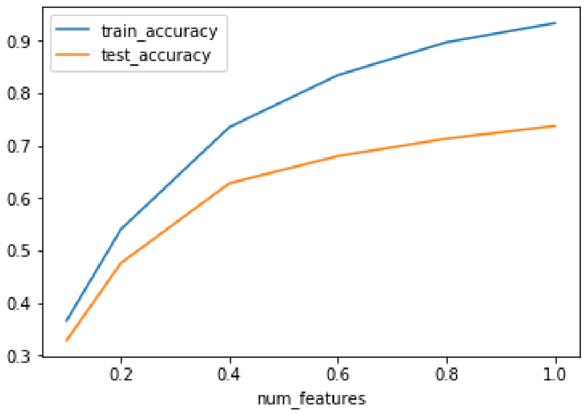
In [75]:

```python
feature_num_df = feature_num(X_improved, y_improved)
feature_num_df.plot(x="num_features", y=["train_accuracy", "test_accuracy"])
```

```
/Users/krutheekarajkumar/anaconda3/lib/python3.6/site-packages/pandas/plotting/_core.py:1716:
UserWarning: Pandas doesn't allow columns to be created via a new attribute name - see
https://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-access
  series.name = label
```

Out[75]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1bab157e48>
```



As the number of features increased the train accuracy and test accuracy also increases. The variancy increases with the incease in the number of features (and increase in model complexity)

Q3) Hyperparameter Tuning

Q3 (a) Code:

In [249]:

```python
def hyperparameter(X, y):
    # result_list is a list of tuples (num_features, train_accuracy, test_accuracy)
    result_list = []

    X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=42)
    # the hyperparameter coefficient in an array of seven numbers to estimate which value would be
the best
    for param in [0.001, 0.01, 0.1, 1, 10, 100, 1000]:

        # train a logistic regression classifier with a variable hyperparameter
        clf = LogisticRegression(C=param).fit(X_train, y_train)

        # predict on train and test set
        y_train_predict = clf.predict(X_train)
        y_test_predict = clf.predict(X_test)

        # calculate train and test accuracy
        train_accuracy = accuracy_score(y_train, y_train_predict)
        test_accuracy = accuracy_score(y_test, y_test_predict)

        # add to result_list
```

```
        result_list.append((param, train_accuracy, test_accuracy))

    # Make a dataframe of the results
    result_df = pd.DataFrame(result_list, columns=["param", "train_accuracy", "test_accuracy"])

    # validate return type
    assert isinstance(result_df, pd.DataFrame), "return type"

    return result_df
```

Q3 (b) Plotting of the performance of the accuracies with each chosen hyperparameters

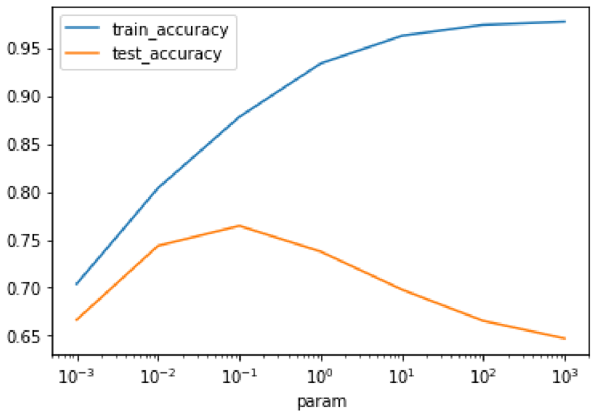
In [250]:

```
param_df = hyperparameter(X_improved, y_improved)
param_df.plot(x="param", y=["train_accuracy", "test_accuracy"], logx=True)
```

```
/Users/krutheekarajkumar/anaconda3/lib/python3.6/site-packages/pandas/plotting/_core.py:1716:
UserWarning: Pandas doesn't allow columns to be created via a new attribute name - see
https://pandas.pydata.org/pandas-docs/stable/indexing.html#attribute-access
  series.name = label
```

Out[250]:

```
<matplotlib.axes._subplots.AxesSubplot at 0x1a0ff2f898>
```



The model shows an optimum point at C~0.1 as this point is the tradeoff point between variation and bias where the left side of the curve experiences high bias and the right experiences high variance. The inverse-regularization coefficient (C) controls this tradeoff such that the best fit line would not deviate too much (high variance) to account for small deviations in data points.

Q4: Feature Encoding:

Q4 (a) Modified Code:

In [82]:

```
def tf_improved_data(file_list, num_words = 1000):
    news_cnt = corpus_count_words_improved(file_list)

    word_list = [word for (word, freq) in news_cnt.most_common(num_words)]
    # applying the same feature cleaning as before
    my_analyzer = RegexTokenizer() | LowercaseFilter() | IntraWordFilter() | StopFilter() | StemFil
ter()

    df_rows = []
    for file_path in file_list:

        with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
            file_data = file.read()
            file_data = clean_file_text(file_data)
            file_words=[acb.text for acb in my_analyzer(file_data)]
            temp = []
```

```
        for word in word_list:
            # storing the number of times each of the popular feature words occur rather than thei
binary outputs (1 if present and 0 if not)
                temp.append(file_words.count(word))
        df_rows.append(temp)

    X = pd.DataFrame(df_rows, columns = word_list)
    # Create a dataframe of targets (y)
    y = [get_target(get_topic_name(file_path)) for file_path in file_list]

    assert isinstance(X, pd.DataFrame) and isinstance(y, list), "return types"

    return X, y
```

Q4 (b) returning dataframe with feature set, confidence interval over multiple splits:

In [83]:

```
X_tf, y_tf = tf_improved_data(all_files)
```

```
[('edu', 66031), ('ax', 62713), ('com', 36790), ('wa', 24738), ('1993', 23326), ('but', 23275), ('
thei', 23217), ('line', 23215), ('new', 23100), ('apr', 22837)]
```

In [84]:

```
train_mean10, train_low10, train_high10, test_mean10, test_low10, test_high10 = random_mean_ci(X_tf
, y_tf, num_tests = 10)
print("Train mean accuracy over 10 random splits: {}".format(train_mean10))
print("Train confidence interval over 10 random splits: [{}, {}]".format(train_low10, train_high10
))
print("Test mean accuracy over 10 random splits: {}".format(test_mean10))
print("Test confidence interval over 10 random splits: [{}, {}]".format(test_low10, test_high10))
```

```
Train mean accuracy over 10 random splits: 0.9336643566478532
Train confidence interval over 10 random splits: [0.9310607795737914, 0.936267933721915]
Test mean accuracy over 10 random splits: 0.7221333333333334
Test confidence interval over 10 random splits: [0.7169599013415523, 0.7273067653251145]
```

Binary encoding seemes to perform marginally better than the term frequency encoding. This is due to the fact that the term frequencies were not normalized. The number (frequency) of each term was taken as is, and this does not necessarily mean that the term was valuable to the class/document (it could mean that the document was simply long).

Q5: Naive Bayes

Q5 (a) Navie Bayers code:

In [110]:

```
from sklearn.naive_bayes import GaussianNB
from sklearn.naive_bayes import MultinomialNB
from sklearn.naive_bayes import BernoulliNB
```

In [111]:

```
def nb_random_mean_ci(X, y, num_tests):
    train_results=[]
    test_results = []
    rs= np.random.randint(1000, size=num_tests)
    #print(rs)
    for i in range(num_tests):
        X_train, X_test, y_train, y_test = train_test_split(X, y, test_size=0.3, random_state=rs[i]
)
    # testing multiple variations of the NB module
        #gnb = GaussianNB()
        #bmn = BernoulliNB()
        mnb = MultinomialNB()
```

```
        y_train_predict = mnb.fit(X_train, y_train).predict(X_train)
        y_test_predict = mnb.fit(X_train, y_train).predict(X_test)
        train_accuracy = accuracy_score(y_train, y_train_predict)
        test_accuracy = accuracy_score(y_test, y_test_predict)
    # train_results is a list of train accuracy results for the differrent random splits of the da
taset
        train_results.append(train_accuracy)

    # test_results is a list of test accuracy results for the differrent random splits of the data
set
        test_results.append(test_accuracy)

    # calculate the train mean and the 95% confidence interval for the list of results
    train_mean = np.mean(train_results)
    train_ci_low, train_ci_high = stats.t.interval(0.95, len(train_results)-1, loc=train_mean, scal
e=stats.sem(train_results))

    # calculate the test mean and the 95% confidence interval for the list of results
    test_mean = np.mean(test_results)
    test_ci_low, test_ci_high = stats.t.interval(0.95, len(test_results)-1, loc=test_mean, scale=st
ats.sem(test_results))

    # validate return types
    assert isinstance(train_mean, float) and isinstance(train_ci_low, float) and isinstance(train_c
i_high, float), "return types"
    assert isinstance(test_mean, float) and isinstance(test_ci_low, float) and isinstance(test_ci_h
igh, float), "return types"

    return train_mean, train_ci_low, train_ci_high, test_mean, test_ci_low, test_ci_high
```

Q5 (b) returning dataframe with feature set, confidence interval over multiple splits:

In [241]:

```
train_mean10, train_low10, train_high10, test_mean10, test_low10, test_high10 = nb_random_mean_ci(X
_improved, y_improved, num_tests = 10)
print("Train mean accuracy over 10 random splits: {}".format(train_mean10))
print("Train confidence interval over 10 random splits: [{}, {}]".format(train_low10, train_high10
))
print("Test mean accuracy over 10 random splits: {}".format(test_mean10))
print("Test confidence interval over 10 random splits: [{}, {}]".format(test_low10, test_high10))
```

```
Train mean accuracy over 10 random splits: 0.767543044938201
Train confidence interval over 10 random splits: [0.7662327039990747, 0.7688533858773273]
Test mean accuracy over 10 random splits: 0.7180000000000001
Test confidence interval over 10 random splits: [0.7163798106289145, 0.7196201893710856]
```

NB - Gaussian:
Train mean accuracy over 10 random splits: 0.5512538401085946

Train confidence interval over 10 random splits: [0.5439689629440503, 0.5585387172731389]

Test mean accuracy over 10 random splits: 0.49045000000000005

Test confidence interval over 10 random splits: [0.48576042122115604, 0.49513957877884407]

NB - MultinomialNB
Train mean accuracy over 10 random splits: 0.7677002214760306

Train confidence interval over 10 random splits: [0.7660730868286525, 0.7693273561234086]

Test mean accuracy over 10 random splits: 0.71585

Test confidence interval over 10 random splits: [0.7121263374236403, 0.7195736625763597]

NB - BernoulliNB
Train mean accuracy over 10 random splits: 0.5889833535757661

Train confidence interval over 10 random splits: [0.5861526150205324, 0.5918140921309999]

Test mean accuracy over 10 random splits: 0.5386166666666667

Test confidence interval over 10 random splits: [0.5339338286489252, 0.5432995046844082]

Multinominal NB function was the best, however it still performed worse than Logistic regression prediction:

This was due to the fact that navie bayers classifier assumes conditional independence where the attribute values are assmed to be independent of each other in the same class. Conditional independence does not hold for text data.

Q6: Binary Logistic Regression

In [187]:

```
def is_graduate_student():
    # ** Graduate students: change the return value to True **
    return True
```

In [188]:

```
import os
```

Q6 (a)

Modify the partial code below

In [234]:

```
def binary_med_data(file_list, num_words = 1000):

    news_cnt = corpus_count_words_improved(file_list)
    print (news_cnt.most_common(10))

    word_list = [word for (word, freq) in news_cnt.most_common(num_words)]
    my_analyzer = RegexTokenizer() | LowercaseFilter() | IntraWordFilter() | StopFilter() | StemFil
ter()

    df_rows = []
    y =[]
    for file_path in file_list:
    # first populating the y-topic list to include two topics, "sci.med" and "NOT"
        if (get_topic_name(file_path) == "sci.med"):
            y.append(get_target(get_topic_name(file_path)))
        else:
            y.append("NOT")

        with open(file_path, 'r', encoding='utf-8', errors='ignore') as file:
            file_data = file.read()
            file_data = clean_file_text(file_data)
            #file_words=tokenizer.tokenize(file_data)
            file_words=[acb.text for acb in my_analyzer(file_data)]
            #print(file_words)
            temp =[]
            for word in word_list:
    # populating the dataframe in binary context to include if a the feature was part of the desir
ed document
                if get_target(get_topic_name(file_path)) == "sci.med":
                    temp.append(1)
                else:
                    temp.append(0)

        df_rows.append(temp)

    X = pd.DataFrame(df_rows, columns = word_list)

    assert isinstance(X, pd.DataFrame) and isinstance(y, list), "return types"

    return X, y
```

Q6 (b)

Use the following code to calculate the mean accuracy and 95% confidence interval over multiple random splits

In [235]:

```
X_med, y_med = binary_med_data(all_files)
```

```
[('edu', 66031), ('ax', 62713), ('com', 36790), ('wa', 24738), ('1993', 23326), ('but', 23275), ('
thei', 23217), ('line', 23215), ('new', 23100), ('apr', 22837)]
```

In [236]:

```
print(len(X_med))
print(len(y_med))
```

```
19997
19997
```

In [237]:

```
train_mean10, train_low10, train_high10, test_mean10, test_low10, test_high10 = random_mean_ci(X_me
d, y_med, num_tests = 10)
print("Train mean accuracy over 10 random splits: {}".format(train_mean10))
print("Train confidence interval over 10 random splits: [{}, {}]".format(train_low10, train_high10
))
print("Test mean accuracy over 10 random splits: {}".format(test_mean10))
print("Test confidence interval over 10 random splits: [{}, {}]".format(test_low10, test_high10))
```

```
Train mean accuracy over 10 random splits: 0.9496606415660498
Train confidence interval over 10 random splits: [0.9488828609971923, 0.9504384221349073]
Test mean accuracy over 10 random splits: 0.9507666666666668
Test confidence interval over 10 random splits: [0.948952234229617, 0.9525810991037165]
```

In [229]:

```
print("X_train shape: ",X_train.shape)
print("X_test shape: ",X_test.shape)
print("y_train shape: ",np.shape(y_train))
print("y_test shape: ",np.shape(y_test))
```

```
X_train shape:  (13997, 1000)
X_test shape:  (6000, 1000)
y_train shape:  (13997,)
y_test shape:  (6000,)
```

The test accuracies are significnatly higher than any of the other tests done which mean that the classifier was able to classify the 30% of the dataset with minimum error.
The accuracy values are significatly higher than the values in the the multiclass logistic regression because multiclass logistic regression had to split the probablities of a document ebing more relevant in one class to the other - split over 19 documents. The Binary logistic regression had to do that just over two documents there by optimizing the fit as best as possible.

In [ ]:

```

```