In [1]:
```
import csv
import pandas as pd
import numpy as np
import scipy as sp
from sklearn.model_selection import train_test_split
import collections
import nltk
from nltk.corpus import stopwords
from sklearn.feature_extraction.text import TfidfVectorizer
import string
import operator
from sklearn.metrics import accuracy_score
from sklearn.linear_model import Perceptron
from sklearn.svm import SVC
import matplotlib.pyplot as plt
```

In [2]:
```
!ls
```

```
Homework_3.pdf    Untitled1.ipynb   hw_3_yinyin.pdf
Untitled.ipynb    hw3.ipynb         wine_original.csv
```

Load the wine dataset

In [3]:
```
wine_ori = pd.read_csv('wine_original.csv')
labels = wine_ori['class']
del wine_ori['class']
```

In [4]:
```
wine_ori.head()
```

Out[4]:

	Alcohol	Malic acid	Ash	Alcalinity of ash	Magnesium	Total phenols	Flavanoids	Nonflavanoid phenols	Proantho
0	14.23	1.71	2.43	15.6	127	2.80	3.06	0.28	2.29
1	13.20	1.78	2.14	11.2	100	2.65	2.76	0.26	1.28
2	13.16	2.36	2.67	18.6	101	2.80	3.24	0.30	2.81
3	14.37	1.95	2.50	16.8	113	3.85	3.49	0.24	2.18
4	13.24	2.59	2.87	21.0	118	2.80	2.69	0.39	1.82

Question 1: Perform a 80-20 split using train test split on the data to obtain the train and the test data (random state=3). Use Logistic Regression to classify the wines according to their cultivators. Tune parameters ’penalty’ and ’C’ using GridSearchCV implementation. Report the accuracy on test data. (10 marks)

In [5]:
```
#data split into train and test dataset
X_train, X_test, y_train, y_test = train_test_split(wine_ori,labels, test_si
```

In [6]:
```
from sklearn.linear_model import LogisticRegression
from sklearn.model_selection import GridSearchCV
from sklearn.metrics import accuracy_score

parameters = { 'penalty': ['l1','l2'],
              'C':[0.1, 0.5, 1, 2, 3, 4, 5, 10]}
logreg = LogisticRegression()
clf = GridSearchCV(logreg, parameters, verbose=True, n_jobs=-1)
clf.fit(X_train, y_train)
y_pred = clf.predict(X_test)

accuracy = accuracy_score(y_pred, y_test)
print ('Selected Parameters: ', clf.best_params_)
print ('Test Accuracy = ' + str(accuracy))
```

```
Fitting 3 folds for each of 16 candidates, totalling 48 fits
Selected Parameters:  {'C': 1, 'penalty': 'l1'}
Test Accuracy = 0.888888888889

[Parallel(n_jobs=-1)]: Done  48 out of  48 | elapsed:    0.3s finished
```

3.1 Process text data. Download the newsgroups data (train and test) using fetch 20newsgroups for categories: ’alt.atheism’, ’comp.graphics’, ’sci.space’ and ’talk.politics.mideast’ after removing ’headers’, ’footers’ and ’quotes’ from the data.

In [7]:
```
#fetch newgroups data
from sklearn import datasets
train = datasets.fetch_20newsgroups(subset = 'train', categories=('alt.athei
test = datasets.fetch_20newsgroups(subset = 'test', categories=('alt.atheism
train_data = train.data
test_data = test.data
y_train = train.target
y_test = test.target
```

In [8]:
```
print (len(train_data),len(y_train),len(y_test),len(test_data))
```

```
2221 2221 1478 1478
```

In [9]:
```
lab_count1=dict(collections.Counter(y_train))
lab_count = collections.OrderedDict(sorted(lab_count1.items()))
print (lab_count, lab_count1)
```

```
OrderedDict([(0, 480), (1, 584), (2, 593), (3, 564)]) {2: 593, 3: 564, 0:
480, 1: 584}
```

In [10]:
```
#train data processing
X_train = []
exclude = set(string.punctuation)
stop_words = set(stopwords.words('english'))
for i in range(len(train_data)):
    A = train_data[i].lower() #lowercase for each word
    A = ''.join(ch for ch in A if ch not in exclude) #remove punctuation
    filtered_words = [word for word in A.split() if word not in stop_words]
    B = ' '.join(word for word in filtered_words)
    X_train.append(B)
```

In [11]:
```
#test data processing
X_test = []
exclude = set(string.punctuation)
stop_words = set(stopwords.words('english'))
for i in range(len(test_data)):
    C = test_data[i].lower() #lowercase for each word
    C = ''.join(ch for ch in C if ch not in exclude) #remove punctuation
    filtered_words = [word for word in C.split() if word not in stop_words]
    D = ' '.join(word for word in filtered_words)
    X_test.append(D)
```

In [12]:
```
# TfidfVectorizer for train data
count_vect = TfidfVectorizer()
X_train_counts = count_vect.fit_transform(X_train) #fit and transform
X_test_counts = count_vect.transform(X_test)
print (X_train_counts.shape, X_test_counts.shape)
```

```
(2221, 34564) (1478, 34564)
```

In [13]:
```
#top 2000 features word
E = count_vect.vocabulary_
sorted_x = sorted(E.items(), key=operator.itemgetter(1), reverse = True)
sorted_x[0:1999]
```

Out[13]:
```
[('úz', 34563),
 ('zzc2', 34562),
 ('zz', 34561),
 ('zyxel', 34560),
 ('zwischen', 34559),
 ('zwarte', 34558),
 ('zware', 34557),
 ('zwakke', 34556),
 ('zwak', 34555),
 ('zwaartepunten', 34554),
 ('zvi', 34553),
 ('zvfd', 34552),
 ('zurueckfuehren', 34551),
 ('zurich', 34550),
 ('zurbrins', 34549),
 ('zur', 34548),
 ('zumrut', 34547),
 ('zuma', 34546),
 ('zulu', 34545),
```

Question 2: After obtaining the tf-idf vectors for train and test data, use the perceptron model (no penalty) to train on the training vectors and compute the accuracy on the test vectors. (5 marks)

In [14]:
```
# Model
clf = Perceptron()
# fit
clf.fit(X_train_counts, y_train)
# predict
pred = clf.predict(X_test_counts)
#evaluate
print ('Test accuracy = ' + str(accuracy_score(y_test, pred)))
```

```
Test accuracy = 0.797699594046
```

Question 3: Keeping all the above data processing steps same observe how the test accuracy changes by varying the number of top features selected for 100, 200, 500, 1000, 1500, 2000, 3000 for a perceptron model. Report and plot the results.(10 mark)

In [15]:
```
accu=pd.DataFrame(columns=['Feature_Number','accuracy'])
for i in (100, 200, 500, 1000, 1500, 2000, 3000):
    count_vect1 = TfidfVectorizer(max_features = i)
    X_train_counts1 = count_vect1.fit_transform(X_train) #fit and transform
    X_test_counts1 = count_vect1.transform(X_test)
    clf1 = Perceptron()
    clf1.fit(X_train_counts1, y_train)
    pred1 = clf1.predict(X_test_counts1)
    accuracy=accuracy_score(y_test,pred1)
    data = pd.DataFrame({'Feature_Number': [i],'accuracy':[accuracy]})
    accu=accu.append(data)
    print ('Test accuracy = ' + str(accuracy_score(y_test, pred1)) + ' when
```

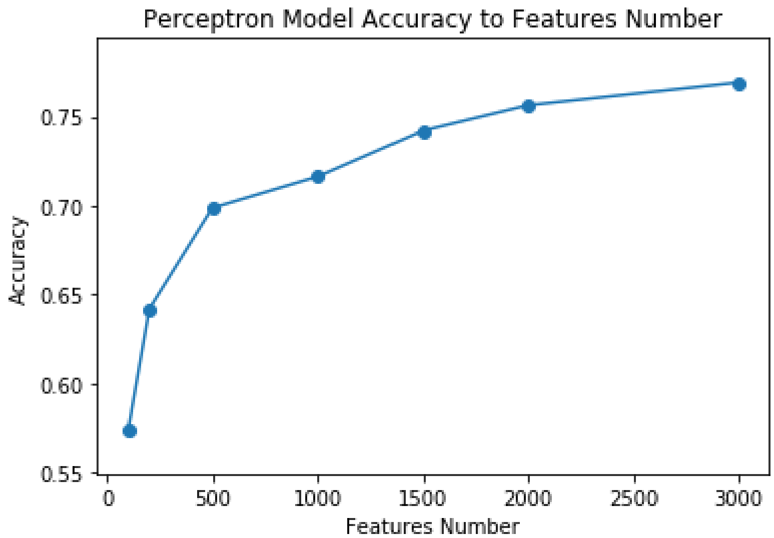
```
Test accuracy = 0.574424898512 when the number of top features is 100
Test accuracy = 0.642083897158 when the number of top features is 200
Test accuracy = 0.698917456022 when the number of top features is 500
Test accuracy = 0.71650879567 when the number of top features is 1000
Test accuracy = 0.742219215156 when the number of top features is 1500
Test accuracy = 0.756427604871 when the number of top features is 2000
Test accuracy = 0.769282814614 when the number of top features is 3000
```

In [16]:
```
accu
```

Out[16]:

	Feature_Number	accuracy
0	100.0	0.574425
0	200.0	0.642084
0	500.0	0.698917
0	1000.0	0.716509
0	1500.0	0.742219
0	2000.0	0.756428
0	3000.0	0.769283

In [17]:
```
#plot accuracy with Feature_Number value
plt.scatter(accu['Feature_Number'], accu['accuracy'])
plt.plot(accu['Feature_Number'], accu['accuracy'])
plt.xlabel('Features Number')
plt.ylabel('Accuracy')
plt.title("Perceptron Model Accuracy to Features Number")
plt.show()
```



Question 4: After obtaining the tf-idf vectors for train and test data, use the SVM model to train on the training vectors and compute the accuracy on the test vectors. Use linear kernel and default parameters. (5 mark)

In [18]:
```
#Model
clf_svm = SVC(kernel='linear')
#fit
clf_svm.fit(X_train_counts, y_train)
#predict
pred = clf_svm.predict(X_test_counts)
#Evaluate
print ('Test accuracy = ' + str(accuracy_score(y_test, pred)))
```

```
Test accuracy = 0.822056833559
```

Question 5: Keeping all the above data processing steps same observe how the test accuracy changes by varying the number of top features selected for 100, 200, 500, 1000, 1500, 2000, 3000 for a linear SVM model. Report and plot the results.(10 mark)

In [19]:

```
accu2=pd.DataFrame(columns=['Feature_Number','accuracy'])
for i in (100, 200, 500, 1000, 1500, 2000, 3000):
    count_vect2 = TfidfVectorizer(max_features = i)
    X_train_counts2 = count_vect2.fit_transform(X_train) #fit and transform
    X_test_counts2 = count_vect2.transform(X_test)
    clf2 = SVC(kernel='linear')
    clf2.fit(X_train_counts2, y_train)
    pred2 = clf2.predict(X_test_counts2)
    accuracy=accuracy_score(y_test,pred2)
    data = pd.DataFrame({'Feature_Number': [i],'accuracy':[accuracy]})
    accu2=accu2.append(data)
    print ('Test accuracy = ' + str(accuracy_score(y_test, pred2)) + ' when
```

```
Test accuracy = 0.631935047361 when the number of top features is 100
Test accuracy = 0.665087956698 when the number of top features is 200
Test accuracy = 0.723951285521 when the number of top features is 500
Test accuracy = 0.752368064953 when the number of top features is 1000
Test accuracy = 0.767253044655 when the number of top features is 1500
Test accuracy = 0.78349120433 when the number of top features is 2000
Test accuracy = 0.803788903924 when the number of top features is 3000
```

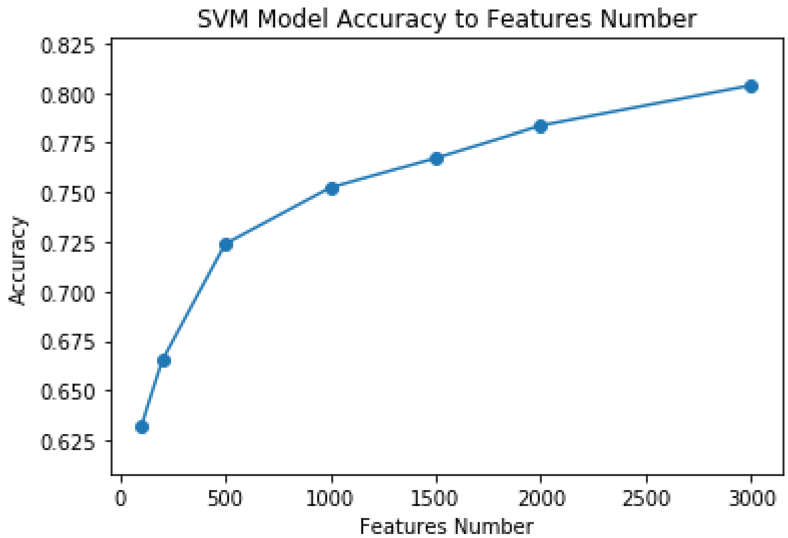
In [20]:

```
accu2
```

Out[20]:

	Feature_Number	accuracy
0	100.0	0.631935
0	200.0	0.665088
0	500.0	0.723951
0	1000.0	0.752368
0	1500.0	0.767253
0	2000.0	0.783491
0	3000.0	0.803789

In [21]:
```
#plot accuracy with Feature_Number value
plt.scatter(accu2['Feature_Number'], accu2['accuracy'])
plt.plot(accu2['Feature_Number'], accu2['accuracy'])
plt.xlabel('Features Number')
plt.ylabel('Accuracy')
plt.title("SVM Model Accuracy to Features Number")
plt.show()
```



Question 6: Perform 80-20 split of the training data to obtain validation data using train test split (random state=10). Use this validation data to tune the cost parameter 'C' for values 0.01,0.1,1,10,100. Select the best value compute the accuracy for the test data. Report the validation and test accuracies. Note: Use full data of 2000 vectors here. (10 marks)

In [22]:
```
#data split into train and test dataset
train_data1, val_data, y_train1, y_val = train_test_split(train_data,y_trair
```

In [23]:
```
#train1 data processing
X_train1 = []
exclude = set(string.punctuation)
stop_words = set(stopwords.words('english'))
for i in range(len(train_data1)):
    G = train_data1[i].lower() #lowercase for each word
    G = ''.join(ch for ch in G if ch not in exclude) #remove punctuation
    filtered_words = [word for word in G.split() if word not in stop_words]
    H = ' '.join(word for word in filtered_words)
    X_train1.append(H)
```

In [24]:
```
#validataion data processing
X_val = []
exclude = set(string.punctuation)
stop_words = set(stopwords.words('english'))
for i in range(len(val_data)):
    G = val_data[i].lower() #lowercase for each word
    G = ''.join(ch for ch in G if ch not in exclude) #remove punctuation
    filtered_words = [word for word in G.split() if word not in stop_words]
    H = ' '.join(word for word in filtered_words)
    X_val.append(H)
```

In [25]:
```
count_vect3 = TfidfVectorizer(max_features = 2000)
X_train_counts3 = count_vect3.fit_transform(X_train1) #fit and transform
X_test_counts3 = count_vect3.transform(X_test)
X_val_counts = count_vect3.transform(X_val)
print (X_val_counts.shape)
```

```
(445, 2000)
```

In [26]:
```
for C in (0.01, 0.1, 1, 10, 100):
    clf3 = SVC(kernel='linear', C=C)
    clf3.fit(X_train_counts3, y_train1) #splited train data
    pred_val = clf3.predict(X_val_counts)
    accuracy_val = accuracy_score(y_val, pred_val)
    print ('when the cost parameter is %s' % C)
    print ('we get validation accuracy = %s' %accuracy_val)
```

```
when the cost parameter is 0.01
we get validation accuracy = 0.244943820225
when the cost parameter is 0.1
we get validation accuracy = 0.737078651685
when the cost parameter is 1
we get validation accuracy = 0.829213483146
when the cost parameter is 10
we get validation accuracy = 0.795505617978
when the cost parameter is 100
we get validation accuracy = 0.786516853933
```

In [27]:
```
clf4 = SVC(kernel='linear', C=1)
clf4.fit(X_train_counts, y_train) #original train data
pred_test = clf4.predict(X_test_counts)
accuracy_test = accuracy_score(y_test, pred_test)
print ('when the cost parameter C=1')
print ('we get test accuracy = %s' %accuracy_test)
```

```
when the cost parameter C=1
we get test accuracy = 0.822056833559
```

Question 7: Train a kernelized SVM (with ’C’=10000) with kernel values - ’poly’ with degree 1, 2, 3, ’rbf ’ and ’sigmoid’, and report the one with best accuracy on validation data. Also report the test accuracy for the selected kernel. (10 marks)

In [28]:
```
count_vect4 = TfidfVectorizer()
X_train_counts4 = count_vect4.fit_transform(X_train1) #use splited train dat
X_test_counts4 = count_vect4.transform(X_test)
X_val_counts1 = count_vect4.transform(X_val)
print (X_val_counts1.shape)
```

```
(445, 29873)
```

In [29]:
```
for deg in [1,2,3]:
    clf5 = SVC(C=10000, kernel='poly', degree = deg)
    clf5.fit(X_train_counts4, y_train1)
    pred_val1 = clf5.predict(X_val_counts1)
    accuracy_val1 = accuracy_score(y_val, pred_val1)
    print ('For kernel poly when the degree is %s' % deg)
    print ('we get validation accuracy = %s' %accuracy_val1)
```

```
For kernel poly when the degree is 1
we get validation accuracy = 0.838202247191
For kernel poly when the degree is 2
we get validation accuracy = 0.244943820225
For kernel poly when the degree is 3
we get validation accuracy = 0.244943820225
```

In [30]:
```
for ker in ('rbf','sigmoid'):
    clf6 = SVC(C=10000, kernel=ker)
    clf6.fit(X_train_counts4, y_train1)
    pred_val2 = clf6.predict(X_val_counts1)
    accuracy_val2 = accuracy_score(y_val, pred_val2)
    print ('when the kernel is %s' % ker)
    print ('we get validation accuracy = %s' %accuracy_val2)
```

```
when the kernel is rbf
we get validation accuracy = 0.860674157303
when the kernel is sigmoid
we get validation accuracy = 0.838202247191
```

In [31]:
```
clf7 = SVC(kernel='rbf', C=10000)
clf7.fit(X_train_counts, y_train) #use original train data
pred_test = clf7.predict(X_test_counts)
accuracy_test = accuracy_score(y_test, pred_test)
print ('when the kernel is "rbf"')
print ('we get test accuracy = %s' %accuracy_test)
```

```
when the kernel is "rbf"
we get test accuracy = 0.821380243572
```

Question 8: Use Cosine Similarity and Laplacian Kernel (exp−||x−y||1) measures, and report the test accuracies using these kernels with SVM. (15 marks)

In [32]:
```
from sklearn.metrics.pairwise import cosine_similarity
clf8 = SVC(kernel=cosine_similarity)
clf8.fit(X_train_counts, y_train)
pred = clf8.predict(X_test_counts)
accuracy = accuracy_score(y_test, pred)
print ('when the kernel is cosine similarity')
print ('we get test accuracy = %s' %accuracy)
```

```
when the kernel is cosine similarity
we get test accuracy = 0.822056833559
```

In [33]:
```
from sklearn.metrics.pairwise import laplacian_kernel
clf9 = SVC(kernel=laplacian_kernel)
clf9.fit(X_train_counts, y_train)
pred = clf9.predict(X_test_counts)
accuracy = accuracy_score(y_test, pred)
print ('when the kernel is laplacian_kernel')
print ('we get test accuracy = %s' %accuracy)
```

```
when the kernel is laplacian_kernel
we get test accuracy = 0.266576454668
```

Question 9: Another way to construct a kernel is use a linear combination of 2 kernels.

In this question, we simply sum two scaled kernels, and the sum of two kernel is still a valid kernel. The reseason is the the sum of two kernels is simply concatenation of the corresponding induced feature respresentations.

In [36]:
```
def gram(x,y,alpha):
    gram_kernel = alpha * cosine_similarity(x, y) + (1-alpha)*laplacian_kerr
    return gram_kernel
```

In [38]:
```
#train data split into train1 and validate data refered to qustion 6
for alpha in (0,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8,0.9,1):
    gram_train1 = gram(X_train_counts3, X_train_counts3,alpha)
    gram_val = gram(X_val_counts, X_train_counts3,alpha)
    clf10 = SVC(kernel='precomputed')
    clf10.fit(gram_train1, y_train1)
    pred = clf10.predict(gram_val)
    accuracy = accuracy_score(y_val, pred)
    print ('when alpha is %s' % alpha)
    print ('we get validataion accuracy = %s' %accuracy)
```

```
when alpha is 0
we get validataion accuracy = 0.244943820225
when alpha is 0.1
we get validataion accuracy = 0.74606741573
when alpha is 0.2
we get validataion accuracy = 0.791011235955
when alpha is 0.3
we get validataion accuracy = 0.811235955056
when alpha is 0.4
we get validataion accuracy = 0.822471910112
when alpha is 0.5
we get validataion accuracy = 0.833707865169
when alpha is 0.6
we get validataion accuracy = 0.833707865169
when alpha is 0.7
we get validataion accuracy = 0.833707865169
when alpha is 0.8
we get validataion accuracy = 0.83595505618
when alpha is 0.9
we get validataion accuracy = 0.829213483146
when alpha is 1
we get validataion accuracy = 0.829213483146
```

In [39]:
```
# when alpha is 0.8, we got the hightest accuracy of validation data predic
#train data is original data
gram_train = gram(X_train_counts, X_train_counts,0.8)
gram_test = gram(X_test_counts, X_train_counts,0.8)
clf10 = SVC(kernel='precomputed')
clf10.fit(gram_train, y_train)
pred = clf10.predict(gram_test)
accuracy = accuracy_score(y_test, pred)
print ('when alpha is 0.8')
print ('we get test accuracy = %s' %accuracy)
```

```
when alpha is 0.8
we get test accuracy = 0.822056833559
```

In [ ]: