Homework 3 - Raul G. Martinez (PID: A12461871)

DSE 220: Machine Learning

Due Date: May 14 11:59pm

1. Turn-in Instructions

The answers should be submitted on Gradescope. You should submit the PDF of the Jupyter Notebook. Explain your approach as clearly as possible whereever needed. Please make sure that the questions are clearly segmented and labeled. To secure full marks for a question both the answer and the code should be correct. Completely wrong (or missing) code with correct answer will result in zero marks. Please complete this homework individually.

In [1]:
```
# import libraries
import pandas as pd
import numpy as np
import matplotlib.pyplot as plt
import nltk
from nltk.corpus import stopwords

# import sklearn libraries
from sklearn.datasets import fetch_20newsgroups
from sklearn.feature_extraction.text import TfidfVectorizer
from sklearn.linear_model import Perceptron
from sklearn.metrics import accuracy_score
from sklearn.svm import SVC
from sklearn.model_selection import train_test_split
from sklearn.metrics.pairwise import cosine_similarity, laplacian_kernel
```

2. Logistic Regression

Question 1: This question was included in the previous homework and no submission is needed.

3. Perceptron & Support Vector Machines

3.1 Data

In this section, we will work with text data. Download the newsgroups train and test data 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 [2]:
```python
# function to get data
corpus_train = fetch_20newsgroups(subset = 'train',
                        remove = ('headers', 'footers', 'quotes'),
                        categories = ['alt.atheism', 'comp.graphics', 'sci.space', 'talk.politics.mideast'])
corpus_test = fetch_20newsgroups(subset = 'test',
                        remove = ('headers', 'footers', 'quotes'),
                        categories = ['alt.atheism', 'comp.graphics', 'sci.space', 'talk.politics.mideast'])

# split into features and labels
X_train = corpus_train.data
y_train = corpus_train.target

X_test = corpus_test.data
y_test = corpus_test.target

# show size of data
print(len(X_train))
print(len(X_test))
```

```
2221
1478
```

Next, we need to vectorize the documents to train our classifier. Use the TfidfVectorizer to get vectors of the documents (after smoothing). A common practice is to convert all the documents to lowercase and remove stopwords like a, and, the etc. Use the stopwords set provided by 'nltk.corpus.stopwords'. Take advantage of the parameters provided by the TfidfVectorizer to convert to lowercase and remove stopwords. (10 marks)

Note: Fit the TfidfVectorizer only on the train data and re-use the same on the test data. Do not fit on the test data again.

Note: You might have to run 'nltk.download('stopwords')' before using nltk's stopwords.

Smoothing the next data is the same as computing the idf values after adding a document with all the words in the vocabulary.

In [3]:
```python
# get stopwords from nltk library
nltk.download('stopwords')
stopW_list = stopwords.words('english')

print(stopW_list)
```

```
['i', 'me', 'my', 'myself', 'we', 'our', 'ours', 'ourselves', 'you', "you're", "you've", "you'll", "you'd", 'your', 'yours', 'yourself', 'yo
urselves', 'he', 'him', 'his', 'himself', 'she', "she's", 'her', 'hers', 'herself', 'it', "it's", 'its', 'itself', 'they', 'them', 'their',
'theirs', 'themselves', 'what', 'which', 'who', 'whom', 'this', 'that', "that'll", 'these', 'those', 'am', 'is', 'are', 'was', 'were', 'be',
'been', 'being', 'have', 'has', 'had', 'having', 'do', 'does', 'did', 'doing', 'a', 'an', 'the', 'and', 'but', 'if', 'or', 'because', 'as',
'until', 'while', 'of', 'at', 'by', 'for', 'with', 'about', 'against', 'between', 'into', 'through', 'during', 'before', 'after', 'above',
'below', 'to', 'from', 'up', 'down', 'in', 'out', 'on', 'off', 'over', 'under', 'again', 'further', 'then', 'once', 'here', 'there', 'when',
'where', 'why', 'how', 'all', 'any', 'both', 'each', 'few', 'more', 'most', 'other', 'some', 'such', 'no', 'nor', 'not', 'only', 'own', 'sam
e', 'so', 'than', 'too', 'very', 's', 't', 'can', 'will', 'just', 'don', "don't", 'should', "should've", 'now', 'd', 'll', 'm', 'o', 're',
've', 'y', 'ain', 'aren', "aren't", 'couldn', "couldn't", 'didn', "didn't", 'doesn', "doesn't", 'hadn', "hadn't", 'hasn', "hasn't", 'haven',
"haven't", 'isn', "isn't", 'ma', 'mightn', "mightn't", 'mustn', "mustn't", 'needn', "needn't", 'shan', "shan't", 'shouldn', "shouldn't", 'wa
sn', "wasn't", 'weren', "weren't", 'won', "won't", 'wouldn', "wouldn't"]

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

In [4]:
```python
# vectorize the documents with train data
vectorizer = TfidfVectorizer(stop_words = stopW_list, lowercase=True, smooth_idf=True)
vectorizer.fit(X_train)
```

Out[4]:
```
TfidfVectorizer(analyzer='word', binary=False, decode_error='strict',
                dtype=<class 'numpy.float64'>, encoding='utf-8',
                input='content', lowercase=True, max_df=1.0, max_features=None,
                min_df=1, ngram_range=(1, 1), norm='l2', preprocessor=None,
                smooth_idf=True,
                stop_words=['i', 'me', 'my', 'myself', 'we', 'our', 'ours',
                            'ourselves', 'you', "you're", "you've", "you'll",
                            "you'd", 'your', 'yours', 'yourself', 'yourselves',
                            'he', 'him', 'his', 'himself', 'she', "she's",
                            'her', 'hers', 'herself', 'it', "it's", 'its',
                            'itself', ...],
                strip_accents=None, sublinear_tf=False,
                token_pattern='(?u)\\b\\w\\w+\\b', tokenizer=None, use_idf=True,
                vocabulary=None)
```

In [5]:
```
# print size of vocabulary from vectorizer
print(len(vectorizer.vocabulary_))
```

```
30451
```

In [6]:
```
# obtain the tf-idf vectors for train and test data
X_train_vector = vectorizer.transform(X_train)
X_test_vector = vectorizer.transform(X_test)

print(X_train_vector.shape)
print(X_test_vector.shape)
```

```
(2221, 30451)
(1478, 30451)
```

3.2 Learning

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 [7]:
```
# model
clf = Perceptron(penalty=None)

# fit
clf.fit(X_train_vector, y_train)

# predict
pred = clf.predict(X_test_vector)

# evaluate on test vector
print('Perceptron Model')
print ('Test accuracy = ' + str(accuracy_score(y_test, pred)))
```

```
Perceptron Model
Test accuracy = 0.7949932341001353
```

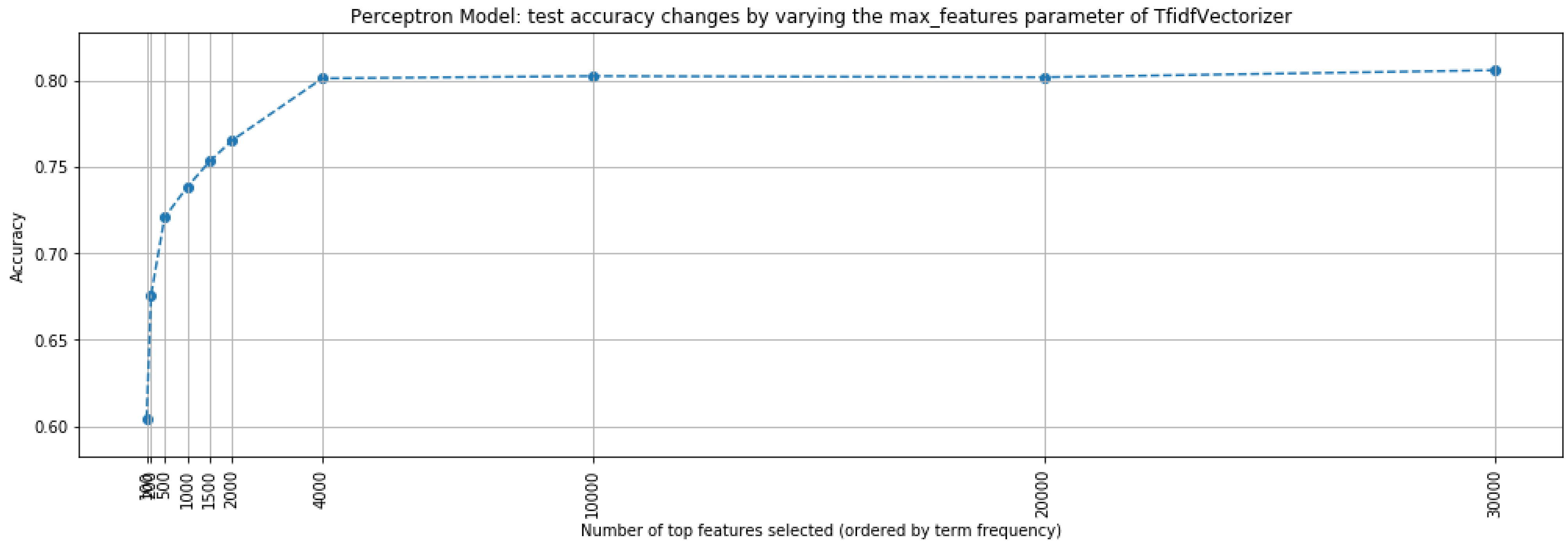
Question 3: Keeping all the above data processing steps same, observe how the test accuracy changes by varying the number of top features selected for 100, 200, 500, 1000, 1500, 2000, 4000, 10000, 20000, 30000 for a perceptron model. Report and plot the results. Provide a brief explanation of the observed results. (10 mark)

In [8]:
```python
# Perceptron Model: calculate test accuracy by varying the max_features parameters for the vectorizer

top_features_list = [100, 200, 500, 1000, 1500, 2000, 4000, 10000, 20000, 30000]
accuracy_list = []

for n in top_features_list:

    # vectorize the documents with train data
    vectorizer_temp = TfidfVectorizer(stop_words = stopW_list, lowercase=True, max_features = n)
    vectorizer_temp.fit(X_train)

    # obtain the tf-idf vectors for train and test data
    X_train_vector_temp = vectorizer_temp.transform(X_train)
    X_test_vector_temp = vectorizer_temp.transform(X_test)

    # model
    clf = Perceptron(penalty=None)

    # fit
    clf.fit(X_train_vector_temp, y_train)

    # predict
    pred = clf.predict(X_test_vector_temp)

    # calculate accuracy
    accuracy = accuracy_score(y_test, pred)
    accuracy_list.append(accuracy)

    # evaluate on test vector and report results
    print('Perceptron Model, max_features = {}'.format(n))
    print ('\tTest accuracy = ' + str(accuracy))
```

```
Perceptron Model, max_features = 100
	Test accuracy = 0.6041948579161028
Perceptron Model, max_features = 200
	Test accuracy = 0.6759133964817321
Perceptron Model, max_features = 500
	Test accuracy = 0.7205683355886333
Perceptron Model, max_features = 1000
	Test accuracy = 0.7381596752368065
Perceptron Model, max_features = 1500
	Test accuracy = 0.7530446549391069
Perceptron Model, max_features = 2000
	Test accuracy = 0.7652232746955345
```

```
Perceptron Model, max_features = 4000
        Test accuracy = 0.8010825439783491
Perceptron Model, max_features = 10000
        Test accuracy = 0.8024357239512855
Perceptron Model, max_features = 20000
        Test accuracy = 0.8017591339648173
Perceptron Model, max_features = 30000
        Test accuracy = 0.8058186738836265
```

In [9]:
```
# plot the results
plt.figure(figsize = (17,5))
plt.scatter(top_features_list, accuracy_list)
plt.plot(top_features_list, accuracy_list, '--')
plt.title('Perceptron Model: test accuracy changes by varying the max_features parameter of TfidfVectorizer')
plt.xlabel('Number of top features selected (ordered by term frequency)')
plt.ylabel('Accuracy')
plt.xticks(top_features_list, rotation = 'vertical')
plt.grid()
plt.show()
```



Explanation (Q3)

The plot above shows how we can significantly reduce the amount of features required to predict in the perceptron model. For instance, we can use 4,000 features to obtain an accuracy of 80.10% on the test data, as opposed to an accuracy of 80.24% for 10,000 features. This leads to a reduction of at least 2.5-fold to the number of features needed with only ~0.1% difference in accuracy. In other words, accuracy has already plateau at 4,000 features and having additional features does not make our predictions better.

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 [10]:

```
# use SVM model
clf = SVC(kernel = 'linear')

# fit
clf.fit(X_train_vector, y_train)

# predict
pred = clf.predict(X_test_vector)

# evaluate on test accuracy
print('SVM Model')
print ('Test accuracy = ' + str(accuracy_score(y_test, pred)))
```

```
SVM Model
Test accuracy = 0.8335588633288228
```

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, 4000, 10000, 20000, 30000 for a linear SVM model. Report and plot the results. Provide a brief explanation of the observed results. (10 mark)

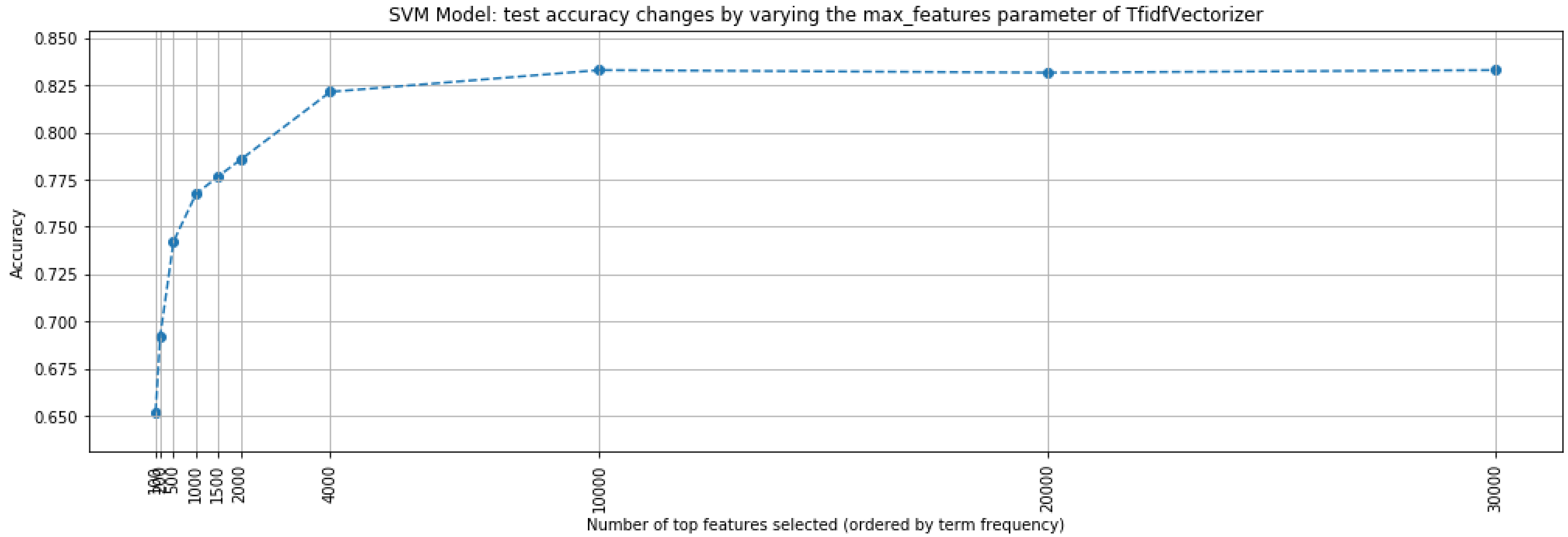
In [11]:
```python
# SVM Model: calculate test accuracy by varying the max_features parameters for the vectorizer

top_features_list = [100, 200, 500, 1000, 1500, 2000, 4000, 10000, 20000, 30000]
accuracy_list = []

for n in top_features_list:

    # vectorize the documents with train data
    vectorizer_temp = TfidfVectorizer(stop_words = stopW_list, lowercase=True, max_features = n)
    vectorizer_temp.fit(X_train)

    # obtain the tf-idf vectors for train and test data
    X_train_vector_temp = vectorizer_temp.transform(X_train)
    X_test_vector_temp = vectorizer_temp.transform(X_test)

    # model
    clf = SVC(kernel = 'linear')

    # fit
    clf.fit(X_train_vector_temp, y_train)


    # predict
    pred = clf.predict(X_test_vector_temp)

    # calculate accuracy
    accuracy = accuracy_score(y_test, pred)
    accuracy_list.append(accuracy)

    # evaluate on test vector and report results
    print('SVM Model, max_features = {}'.format(n))
    print ('\tTest accuracy = ' + str(accuracy))
```

```
SVM Model, max_features = 100
        Test accuracy = 0.652232746955345
SVM Model, max_features = 200
        Test accuracy = 0.6921515561569689
SVM Model, max_features = 500
        Test accuracy = 0.7422192151556157
SVM Model, max_features = 1000
        Test accuracy = 0.7672530446549392
SVM Model, max_features = 1500
        Test accuracy = 0.7767253044654939
```

```
SVM Model, max_features = 2000
        Test accuracy = 0.7855209742895806
SVM Model, max_features = 4000
        Test accuracy = 0.8213802435723951
SVM Model, max_features = 10000
        Test accuracy = 0.8328822733423545
SVM Model, max_features = 20000
        Test accuracy = 0.8315290933694182
SVM Model, max_features = 30000
        Test accuracy = 0.8328822733423545
```

In [12]:

```
# plot the results
plt.figure(figsize = (17,5))
plt.scatter(top_features_list, accuracy_list)
plt.plot(top_features_list, accuracy_list, '--')
plt.title('SVM Model: test accuracy changes by varying the max_features parameter of TfidfVectorizer')
plt.xlabel('Number of top features selected (ordered by term frequency)')
plt.ylabel('Accuracy')
plt.xticks(top_features_list, rotation = 'vertical')
plt.grid()
plt.show()
```



Explanation (Q5)

Similar observations to the Perceptron Model. The plot above shows how we can significantly reduce the amount of features required to predict in the SVM model. For instance, we can use 4,000 features to obtain an accuracy of 82.13% on the test data, as opposed to an accuracy of ~83.0% for >=10,000 features. This leads to a reduction of at least 2.5-fold to the number of features needed with only ~1.0% difference in accuracy. In other words, accuracy has already likely plateau at 4,000 features and having additional features does not make our predictions better.

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 regularization parameter 'C' for values 0.01,0.1,1,10,100. Select the best 'C' and compute the accuracy for the test data. Report the validation and test accuracies. Use feature vectors of 2000 dimensions. (10 marks)

Note: Retrain on train + validation data while reporting accuracy on test data

In [13]:
```
# split training data into train + validation
X_train2, X_val, y_train2, y_val = train_test_split(X_train, y_train, test_size=0.2, random_state=10)
```

In [14]:
```
# vectorize the documents with train data
vectorizer_temp = TfidfVectorizer(stop_words = stopW_list, lowercase=True, max_features = 2000)
vectorizer_temp.fit(X_train2, y_train2)

# obtain the tf-idf vectors for train and test data
X_train_vector_temp = vectorizer_temp.transform(X_train2)
X_val_vector_temp = vectorizer_temp.transform(X_val)

# get original training and test data vectors with only 2000 features
# this becomes train + val for finding test accuracy
vectorizer2000 = TfidfVectorizer(stop_words = stopW_list, lowercase=True, max_features = 2000)
vectorizer2000.fit(X_train, y_train)
X_train2000_vector = vectorizer2000.transform(X_train)
X_test2000_vector = vectorizer2000.transform(X_test)
```

In [15]:
```python
# use validation data to tune the regularization parameter 'C', use SVM Model

C_list = [0.01, 0.1, 1, 10, 100]

for c in C_list:

    # model
    clf_SVC = SVC(kernel = 'linear', C = c)

    # fit
    clf_SVC.fit(X_train_vector_temp, y_train2)

    # predict
    pred_SVC = clf_SVC.predict(X_val_vector_temp)

    # evaluate on test vector and report results
    print('C-value = {}'.format(c))
    print ('\tSVC Model -> Validation accuracy = ' + str(accuracy_score(y_val, pred_SVC)))
```

```
C-value = 0.01
	SVC Model -> Validation accuracy = 0.2449438202247191
C-value = 0.1
	SVC Model -> Validation accuracy = 0.7595505617977528
C-value = 1
	SVC Model -> Validation accuracy = 0.8426966292134831
C-value = 10
	SVC Model -> Validation accuracy = 0.8202247191011236
C-value = 100
	SVC Model -> Validation accuracy = 0.797752808988764
```

In [16]:
```python
# use best C-value (C = 1, val accuracy of 84.3%) from validation to report test accuracy

# model
clf_SVC = SVC(kernel = 'linear', C = 1)

# fit, use original training data that contains train + validation
clf_SVC.fit(X_train2000_vector, y_train)

# predict
pred_SVC = clf_SVC.predict(X_test2000_vector)

# evaluate on test vector and report results
print('C-value = {}'.format(1))
print ('\tSVC Model -> Test accuracy = ' + str(accuracy_score(y_test, pred_SVC)))
```

```
C-value = 1
	SVC Model -> Test accuracy = 0.7855209742895806
```

Question 7: Use the same train and validation split as the previous question. 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. Report the test accuracy for the selected kernel. (10 marks)

In [17]:
```
# use validation data to tune the kernel values

kernel_values = ['poly', 'rbf', 'sigmoid']
poly_values = [1, 2, 3]
c = 10000

for k in kernel_values:
    for p in poly_values:

        # prevent from running same non-poly kernel multiple times
        if k != 'poly' and p != 1: continue

        # model
        clf_SVC = SVC(kernel = k, C = c, degree = p)

        # fit
        clf_SVC.fit(X_train_vector_temp, y_train2)

        # predict
        pred_SVC = clf_SVC.predict(X_val_vector_temp)

        # change poly degree to NA for non-poly kernel
        if k != 'poly': p = 'NA'

        # evaluate on test vector and report results
        print('Kernel = {}, Poly Degree = {}, C-value = {}'.format(k, p, c))
        print ('\tSVC Model -> Validation accuracy = ' + str(accuracy_score(y_val, pred_SVC)))
```

```
Kernel = poly, Poly Degree = 1, C-value = 10000
	SVC Model -> Validation accuracy = 0.7955056179775281
Kernel = poly, Poly Degree = 2, C-value = 10000
	SVC Model -> Validation accuracy = 0.6269662921348315
Kernel = poly, Poly Degree = 3, C-value = 10000
	SVC Model -> Validation accuracy = 0.3325842696629214
Kernel = rbf, Poly Degree = NA, C-value = 10000
	SVC Model -> Validation accuracy = 0.8539325842696629
Kernel = sigmoid, Poly Degree = NA, C-value = 10000
	SVC Model -> Validation accuracy = 0.7528089887640449
```

In [18]:
```
# use best kernel ('rbf', val accuracy of 82.0%) from validation to report test accuracy

# model
clf_SVC = SVC(kernel = 'rbf', C = 10000)

# fit, use original training data that contains train + validation
clf_SVC.fit(X_train2000_vector, y_train)

# predict
pred_SVC = clf_SVC.predict(X_test2000_vector)

# evaluate on test vector and report results
print('Kernel = {}, C-value = {}'.format('rbf', 10000))
print ('\tSVC Model -> Test accuracy = ' + str(accuracy_score(y_test, pred_SVC)))
```

```
Kernel = rbf, C-value = 10000
	SVC Model -> Test accuracy = 0.7895805142083897
```

3.3 Custom Kernels

Now we introduce the concept of custom kernels in Support Vector Machines. In class we discussed how kernel functions are a form of similarity measure for our data. There are good chances that we need some other form of similarity measure for our data which works better with the SVM classifier.

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 [19]:
```python
# report test accuracies with both custom kernels

kernel_list = [cosine_similarity, laplacian_kernel]
k_name = ['cosine_similarity', 'laplacian_kernel']

for k, k_name in zip(kernel_list, k_name):

    # model
    clf_SVC = SVC(kernel = k)

    # fit, use original training data that contains train + validation
    clf_SVC.fit(X_train2000_vector, y_train)

    # predict
    pred_SVC = clf_SVC.predict(X_test2000_vector)

    # evaluate on test vector and report results
    print('Kernel = {}'.format(k_name))
    print ('\tSVC Model -> Test accuracy = ' + str(accuracy_score(y_test, pred_SVC)))
```

```
Kernel = cosine_similarity
	SVC Model -> Test accuracy = 0.7855209742895806
Kernel = laplacian_kernel
	SVC Model -> Test accuracy = 0.2665764546684709
```

Question 9: Another way to construct a kernel is use a linear combination of 2 kernels. Let K be a kernel represented as:

$$K(x,y) = \alpha K1(x,y) + (1-\alpha)K2(x,y) \quad (0 \leq \alpha \leq 1)$$

Provide a brief explanation of why K is a valid kernel. Does your reasoning hold true for other values of α as well? Let K1 be the Cosine Similarity and K2 be the Laplacian Kernel. Using K as kernel, train a SVM model to tune the value of α (upto one decimal) and report the accuracy on the test data using the selected parameter. (15 marks)

In [20]:
```
def two_kernels_linear(X, Y):
    """
    linear combination of two kernels, as:
    K(x,y) = αK1(x,y) + (1-α)K2(x,y)

    where,
    K1: cosine similarity
    K2: laplacian kernel
    """

    global alpha # variable defined before func is called
    K1_term = alpha * cosine_similarity(X, Y)
    K2_term = (1 - alpha) * laplacian_kernel(X, Y)

    return K1_term + K2_term
```

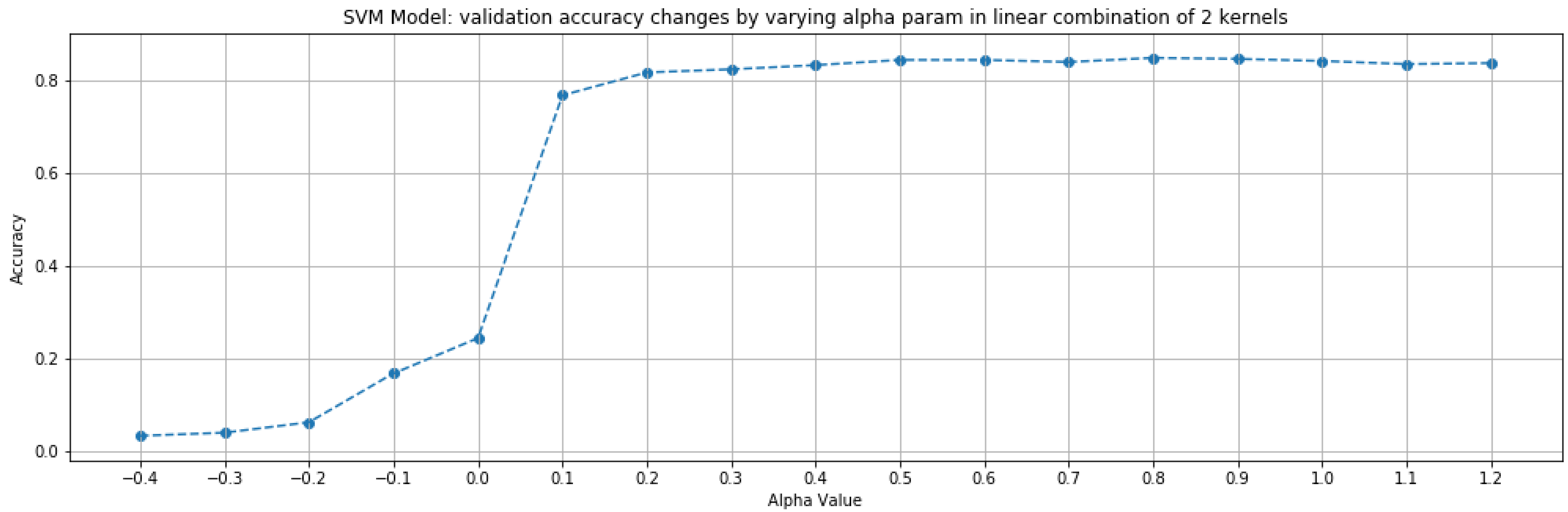
In [21]:
```python
# tune the value for alpha up to one decimal, evaluate on validation data

alpha_list = np.arange(-0.4,1.3,0.1)
val_accuracy_list = []

for alpha in alpha_list:

    # model
    clf_SVC = SVC(kernel = two_kernels_linear)

    # fit
    clf_SVC.fit(X_train_vector_temp, y_train2)

    # predict
    pred_SVC = clf_SVC.predict(X_val_vector_temp)

    # evaluate on val vector and report results
    print('alpha value = {}'.format(np.round(alpha,1)))
    val_accuracy = accuracy_score(y_val, pred_SVC)
    val_accuracy_list.append(val_accuracy)
    print ('\tSVC Model -> Validation accuracy = ' + str(val_accuracy))
```

```
alpha value = -0.4
	SVC Model -> Validation accuracy = 0.033707865168539325
alpha value = -0.3
	SVC Model -> Validation accuracy = 0.04044943820224719
alpha value = -0.2
	SVC Model -> Validation accuracy = 0.06292134831460675
alpha value = -0.1
	SVC Model -> Validation accuracy = 0.16853932584269662
alpha value = -0.0
	SVC Model -> Validation accuracy = 0.2449438202247191
alpha value = 0.1
	SVC Model -> Validation accuracy = 0.7685393258426966
alpha value = 0.2
	SVC Model -> Validation accuracy = 0.8179775280898877
alpha value = 0.3
	SVC Model -> Validation accuracy = 0.8247191011235955
alpha value = 0.4
	SVC Model -> Validation accuracy = 0.8337078651685393
alpha value = 0.5
	SVC Model -> Validation accuracy = 0.8449438202247191
alpha value = 0.6
	SVC Model -> Validation accuracy = 0.8449438202247191
```

```
alpha value = 0.7
        SVC Model -> Validation accuracy = 0.8404494382022472
alpha value = 0.8
        SVC Model -> Validation accuracy = 0.849438202247191
alpha value = 0.9
        SVC Model -> Validation accuracy = 0.8471910112359551
alpha value = 1.0
        SVC Model -> Validation accuracy = 0.8426966292134831
alpha value = 1.1
        SVC Model -> Validation accuracy = 0.8359550561797753
alpha value = 1.2
        SVC Model -> Validation accuracy = 0.8382022471910112
```

In [22]:

```
# plot the results
plt.figure(figsize = (17,5))
plt.scatter(alpha_list, val_accuracy_list)
plt.plot(alpha_list, val_accuracy_list, '--')
plt.title('SVM Model: validation accuracy changes by varying alpha param in linear combination of 2 kernels')
plt.xlabel('Alpha Value')
plt.ylabel('Accuracy')
plt.xticks(alpha_list)
plt.grid()
plt.show()
```



In [23]:
```
# report accuracy on test data, selected alpha value is 0.8 with val accuracy of 0.8494

# define alpha
alpha = 0.8

# model
clf_SVC = SVC(kernel = two_kernels_linear)

# fit
clf_SVC.fit(X_train2000_vector, y_train)

# predict
pred_SVC = clf_SVC.predict(X_test2000_vector)

# evaluate on test vector and report results
print('alpha value = {}, Kernel = {}'.format(0.0, 'Linear Combination of 2', 10000))
print ('\tSVC Model -> Test accuracy = ' + str(accuracy_score(y_test, pred_SVC)))
```

```
alpha value = 0.0, Kernel = Linear Combination of 2
        SVC Model -> Test accuracy = 0.7922868741542625
```

Explanation (Q9)

K(x,y) is a valid kernel for the linear combination of K1(x,y) and K2(x,y) for alpha values from [0,1]. However, when alpha = 0 the accuracy is significantly reduced (about 3 times) as the K1(x,y) term is elimiated and K2(x,y) is the only one considered (see scatter plot above). Otherwise, any value from (0,1] (zero not included) yields an improvement in accuracy for the linear combination of 2 kernels as compared to their individual effects.

Considering values for alpha outside the range [0,1]:

The kernel is not valid for alpha values less than zero, and is not practically valid (with no added benefit) for values greater than 1.

Below are the kernel properties, for any space X of samples and kernels:

- (1) k(x, y) = k1(x, y) + k2(x, y)
- (2) k(x, y) = ak1(x, y) where a > 0

When alpha < 0: this will create a subtraction between K1(x,y) and K2(x,y), hence, making the accuracy of the linear combination of 2 kernels even worst than the lowest individual kernel accuracy between the two. Accuracy approximates to zero with more negative values for alpha (see scatter plot above).

When alpha > 1: this will not affect the performance of accuracy but it will also not add any additional benefit as accuracy has already plateau between 0 and 1 (see scatter plot above), fundamentally, the linear combination of kernels will just have extra information (larger numbers) that is likely not very useful.

```
In [ ]:
```