Data Preparation

In [44]:
```
import os
import numpy as np
import pickle
import random
import pdb
from collections import Counter
'''
Note:  This code is just a hint for people who are not familiar with text proc
essing in python. There is no obligation to use this code, though you may if y
ou like.
'''


def folder_list(path,label):
    '''
    PARAMETER PATH IS THE PATH OF YOUR LOCAL FOLDER
    '''
    filelist = os.listdir(path)
    review = []
    for infile in filelist:
        file = os.path.join(path,infile)
        r = read_data(file)
        r.append(label)
        review.append(r)
    return review

def read_data(file):
    '''
    Read each file into a list of strings.
    Example:
    ["it's", 'a', 'curious', 'thing', "i've", 'found', 'that', 'when', 'willi
s', 'is', 'not', 'called', 'on',
    ...'to', 'carry', 'the', 'whole', 'movie', "he's", 'much', 'better', 'an
d', 'so', 'is', 'the', 'movie']
    '''
    f = open(file)
    lines = f.read().split(' ')
    symbols = '${}()[].,:;+-*/&|<>=~" '
    words = list(map(lambda Element: Element.translate(str.maketrans({char: No
ne for char in symbols})).strip(), lines))
    words = list(filter(lambda x:x!='', words))
    return words

################################################
######## YOUR CODE STARTS FROM HERE. ##########
################################################

def shuffle_data():
    '''
    pos_path is where you save positive review data.
```

```
    neg_path is where you save negative review data.
    '''
    pos_path = 'C:\\Users\\kumar\\Downloads\\MLCS\\Assignments\\A3\\hw3\\hw3-s
entiment\\data\\data\\neg'
    neg_path = 'C:\\Users\\kumar\\Downloads\\MLCS\\Assignments\\A3\\hw3\\hw3-s
entiment\\data\\data\\pos'

    pos_review = folder_list(pos_path,1)
    neg_review = folder_list(neg_path,-1)

    review = pos_review + neg_review
    random.shuffle(review)
    return review


def bag_of_words(reviews):
    '''
    Accepts a review (a list of words) and convert it into a bag of words
    Return a list of tuples containing bag of words and correxponding result
    '''
    return [(Counter(review[:-1]), review[-1]) for review in reviews]

'''
Now you have read all the files into list 'review' and it has been shuffled.
Save your shuffled result by pickle.
*Pickle is a useful module to serialize a python object structure.
*Check it out. https://wiki.python.org/moin/UsingPickle
'''
print("Pickling the file")
pickle.dump(shuffle_data(), open("review.p","wb"))
reviews = pickle.load( open( "review.p", "rb" ) )
train_data = reviews[:1500]
validation_data = reviews[1500:2000]
train_data = bag_of_words(train_data)
validation_dataset = bag_of_words(validation_data)
#print(train_data[1])
# Split into 1500 training and 500 validation examples
```

```
Pickling the file
```

Utilities

In [45]:
```
def dotProduct(d1, d2):
    """
    @param dict d1: a feature vector represented by a mapping from a feature
 (string) to a weight (float).
    @param dict d2: same as d1
    @return float: the dot product between d1 and d2
    """
    if len(d1) < len(d2):
        return dotProduct(d2, d1)
    else:
        return sum(d1.get(f, 0) * v for f, v in d2.items())

def increment(d1, scale, d2):
    """
    Implements d1 += scale * d2 for sparse vectors.
    @param dict d1: the feature vector which is mutated.
    @param float scale
    @param dict d2: a feature vector.

    NOTE: This function does not return anything, but rather
    increments d1 in place. We do this because it is much faster to
    change elements of d1 in place than to build a new dictionary and
    return it.
    """
    for f, v in d2.items():
        d1[f] = d1.get(f, 0) + v * scale
```

Support Vector Machine via Pegasos

In [50]:
```
# 6.2 Normal Pagasos Implementation
from timeit import default_timer
def pegasos(Lambda, data, max_iters):
    '''
    Args
        Lambda - Regularization parameter
    Returns:
        w - a asparse weight vector w
    '''
    iters = 0
    t=1
    w=Counter()
    # a simple termination condition for now
    while(iters<max_iters):
        print("Running ", iters)
        iters = iters+1
        for data in train_data:
            t=t+1
            eta = 1.0/(t*Lambda)
            y = data[1]
            x = data[0]
            if (y*dotProduct(w,x)<1):
                increment(w, -1.0*(Lambda*eta), w)
                increment(w, eta*y ,x)
            else:
                increment(w, -1.0*(Lambda*eta), w)
    return w
print("Time for Normal Pegasos")
start = default_timer()
w_opt1 = pegasos(0.1, train_data, 1)
print(default_timer()-start)
i =0
for k,v in w_opt2.most_common():
    print(k,":", v)
    i=i+1
    if (i>10): break;
```

```
Time for Normal Pegasos
Running  0
18.07491800529533
bad : 1.0792804796802142
this : 0.7261825449700204
have : 0.712858094603598
at : 0.6595602931379082
even : 0.6129247168554302
movie : 0.5929380413057963
no : 0.5596269153897407
plot : 0.5596269153897406
her : 0.5396402398401078
only : 0.532978014656896
off : 0.5129913391072626
```

In [51]:
```
# 6.3 Optimized pagasos implementation
def pegasos_optimized(Lambda, data, max_iters, print_kinks = False):
    '''
    Args
        Lambda - Regularization parameter
    Returns:
        w - a asparse weight vector w
    '''
    iters = 0
    t=1
    w=Counter()
    s=1
    kinks=0
    # a simple termination condition for now
    while(iters<max_iters):
        iters = iters+1
        for data in train_data:
            t=t+1
            eta = 1.0/(t*Lambda)
            y = data[1]
            x = data[0]
            # if s==0 reset the parameter
            if (s==0):
                s=1
                w=Counter()
            if print_kinks and dotProduct(x,w)==0:
                print('Feature Vector:')
                print(x)
                print("Parameter Vector:")
                print(w)
                kinks=kinks+1
            if(y*dotProduct(x,w)<1/s):
                s = (1-eta*Lambda)*s
                increment(w, ((1/s)*eta*y) ,x)
            else: s = (1-eta*Lambda)*s
    if print_kinks: print("Number of kinks in the plot {0}".format(kinks))
    k = Counter()
    increment(k, s, w)
    return k
print("Time for optimized pegasos")
start = default_timer()
w_opt2 = pegasos_optimized(0.1, train_data, 1)
print(default_timer()-start)
i =0
for k,v in w_opt2.most_common():
    print(k,":", v)
    i=i+1
    if (i>10): break;
```

```
Time for optimized pegasos
0.49815487921296153
bad : 1.0792804796802142
this : 0.7261825449700204
have : 0.712858094603598
at : 0.6595602931379082
even : 0.6129247168554302
movie : 0.5929380413057963
no : 0.5596269153897407
plot : 0.5596269153897406
her : 0.5396402398401078
only : 0.532978014656896
off : 0.5129913391072626
```

In [52]:
```
# 6.5 0-1 loss computation
def compute_loss(w, data):
    '''
    Return:
    percent error using 0-1 loss
    '''
    loss = 0
    for data_point in data:
        x = data_point[0]
        y = data_point[1]
        if np.sign(dotProduct(x,w)) != np.sign(y): loss=loss+1
    return (loss/len(data))*100

#compute loss on validation dataset
print("Loss with normal pegasos")
loss1 = compute_loss(w_opt1,validation_dataset)
loss2 = compute_loss(w_opt2, validation_dataset)
print(loss1)
print("Loss with optimized pegasos")
print(loss2)
```

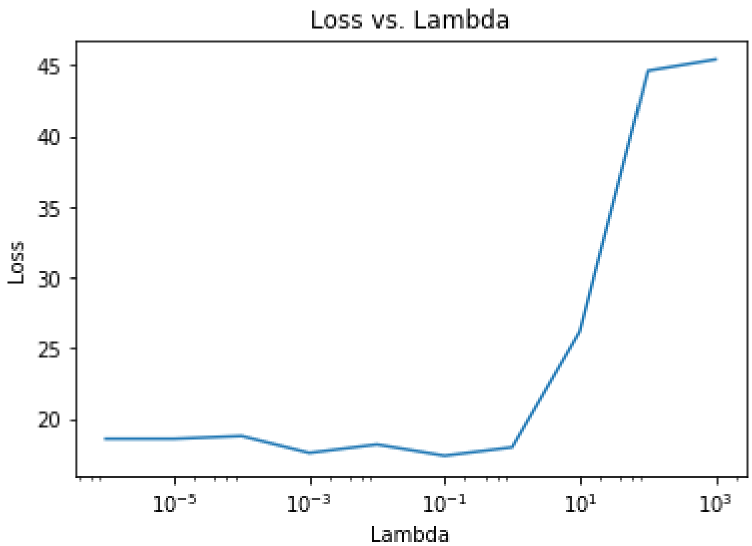
```
Loss with normal pegasos
26.0
Loss with optimized pegasos
26.0
```

In [49]:
```
# 6.6 Search for optimal Lambda
# Write a module for searching best Lambda
def get_opt_lambda(data, lambda_range):
    '''
    Search for an optimal lambda.
    '''
    loss_hist = list()
    loss_min = np.inf
    lambda_opt = None
    # for a range of lambda calculate the loss and keep track of the minimum
    for Lambda in lambda_range:
        w = pegasos_optimized(Lambda, data, 30)
        loss = compute_loss(w, validation_dataset)
        loss_hist.append(loss)
        print("Lambda is {0}, loss is {1}".format(Lambda,loss))
        # update the minimum
        if loss<loss_min:
            lambda_opt = Lambda
            loss_min = loss
            w_opt = w
    return lambda_opt, loss_hist, w_opt

# snippet to test lambda search
print("Running")
Lambdas=list(map(lambda x:10**x,np.linspace(-6,3,10)))
lambda_opt, loss_hist, w_opt = get_opt_lambda(train_data, Lambdas)
print(lambda_opt)
```

```
Running
Lambda is 1e-06, loss is 18.6
Lambda is 1e-05, loss is 18.6
Lambda is 0.0001, loss is 18.8
Lambda is 0.001, loss is 17.599999999999998
Lambda is 0.01, loss is 18.2
Lambda is 0.1, loss is 17.4
Lambda is 1.0, loss is 18.0
Lambda is 10.0, loss is 26.200000000000003
Lambda is 100.0, loss is 44.6
Lambda is 1000.0, loss is 45.4
0.1
```

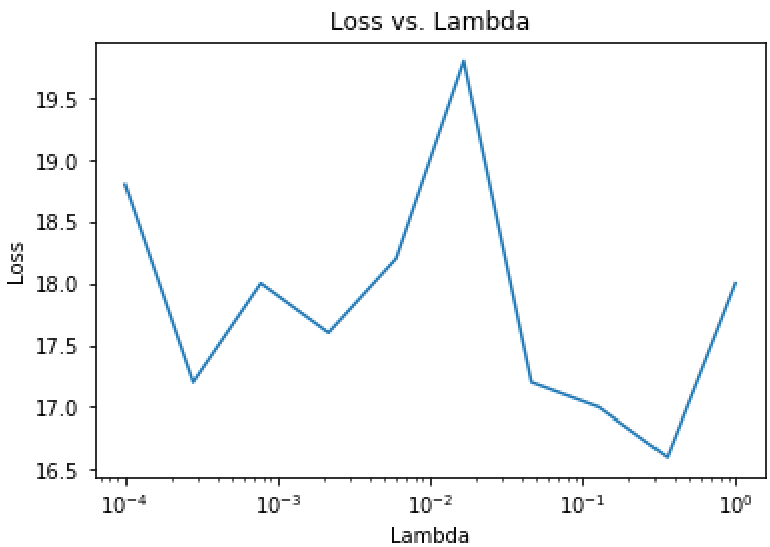
In [53]:
```
# Loss against Lambda
import matplotlib.pyplot as plt
%matplotlib inline
plt.plot(Lambdas,loss_hist)
plt.xlabel('Lambda')
plt.ylabel('Loss')
plt.xscale('log')
plt.title('Loss vs. Lambda')
plt.show()
```



In [54]:
```
# Zoomed in Lambda search
Lambdas=list(map(lambda x:10**x,np.linspace(-4,0,10)))
lambda_opt, loss_hist,w_opt = get_opt_lambda(train_data, Lambdas)
print(lambda_opt)
```

```
Lambda is 0.0001, loss is 18.8
Lambda is 0.0002782559402207126, loss is 17.2
Lambda is 0.000774263682681127, loss is 18.0
Lambda is 0.002154434690031882, loss is 17.599999999999998
Lambda is 0.005994842503189409, loss is 18.2
Lambda is 0.016681005372000592, loss is 19.8
Lambda is 0.046415888336127774, loss is 17.2
Lambda is 0.12915496650148828, loss is 17.0
Lambda is 0.3593813663804626, loss is 16.6
Lambda is 1.0, loss is 18.0
0.35938136638
```

In [55]:
```
plt.plot(Lambdas,loss_hist)
plt.xlabel('Lambda')
plt.ylabel('Loss')
plt.xscale('log')
plt.title('Loss vs. Lambda')
plt.show()
```



In [56]:

```
# 6.7 Plotting score against pecent error
import pandas as pd
validation_scores = list()
error = list()
for data in validation_dataset:
    x= data[0]
    y= data[1]
    score = dotProduct(x,w_opt)
    validation_scores.append(score)
    error.append(np.sign(y)!=np.sign(score))
data = pd.DataFrame(validation_scores)
data['error'] = error
data.columns = ['score', 'error']
groups = data.groupby(pd.cut(data.score,10))
groups.sum()
```

Out[56]:

	score	error
score		
(-5.0433, -3.956]	-26.885914	0.0
(-3.956, -2.88]	-24.388529	0.0
(-2.88, -1.804]	-94.225385	2.0
(-1.804, -0.728]	-144.289194	12.0
(-0.728, 0.348]	-37.586410	61.0
(0.348, 1.425]	91.737028	8.0
(1.425, 2.501]	37.990429	0.0
(2.501, 3.577]	8.354171	0.0
(3.577, 4.653]	3.647421	0.0
(4.653, 5.73]	5.729533	0.0

As is clear from above, the error is highest in the range with smallest absolute values of scores. Hence, higher the magnitude of score, higher are the chances of the prediction being correct; We can therefore think of magnitude of score as the confidence of our prediction.

In [57]:
```
#6.8 Investigating the kinks
# On Training Data
w_temp = pegasos_optimized(lambda_opt, train_data, 31, print_kinks=True)

# On validation dataset
validation_kinks = 0
for data in validation_dataset:
    x= data[0]
    y= data[1]
    if (dotProduct(x,w_temp)==0): validation_kinks+=1
print(validation_kinks)
```

```
Feature Vector:
Counter({'the': 48, 'to': 24, 'of': 22, 'a': 18, 'and': 14, 'her': 14, 'was':
12, 'as': 11, 'she': 10, 'in': 9, 'he': 6, 'would': 6, 'his': 6, 'joan': 5,
 'by': 5, "joan's": 5, 'are': 5, 'is': 4, 'an': 4, 'god': 4, 'on': 4, 'have':
4, 'which': 4, 'revelation': 4, 'holy': 4, 'spirit': 4, 'spiritual': 3, 'it':
3, 'there': 3, 'man': 3, 'or': 3, 'being': 3, 'were': 3, 'message': 3, 'but':
3, 'at': 3, 'john': 3, 'who': 3, 'scriptures': 3, '1': 3, 'they': 3, 'for':
 2, 'with': 2, 'such': 2, 'messenger': 2, 'little': 2, 'be': 2, 'from': 2, 'w
oman': 2, 'strong': 2, 'leadership': 2, 'that': 2, 'visions': 2, 'see': 2, 'd
auphin': 2, 'deliver': 2, 'him': 2, 'if': 2, 'army': 2, 'command': 2, 'crow
n': 2, 'does': 2, 'throne': 2, 'fifth': 2, 'element': 2, 'cast': 2, 'battle':
2, 'end': 2, 'jovovich': 2, 'role': 2, 'looks': 2, 'character': 2, 'been': 2,
'since': 2, 'like': 2, 'malkovich': 2, 'death': 2, 'than': 2, 'one': 2, 'camp
aign': 2, 'while': 2, 'men': 2, 'word': 2, 'spirits': 2, 'movie': 1, 'deep':
 1, 'religious': 1, 'undertones': 1, 'surprising': 1, 'find': 1, 'story': 1,
 'arc': 1, 'ungodly': 1, 'mess': 1, 'early': 1, 'mid': 1, "1400's": 1, 'way':
1, 'light': 1, 'found': 1, 'shining': 1, 'heart': 1, 'church': 1, 'dismally':
1, 'dark': 1, 'oppressive': 1, 'place': 1, 'france': 1, 'involved': 1, 'hundr
ed': 1, 'years': 1, 'war': 1, 'against': 1, 'england': 1, 'no': 1, 'politica
l': 1, 'country': 1, 'morale': 1, 'low': 1, 'hope': 1, 'future': 1, 'within':
1, 'this': 1, 'setting': 1, 'young': 1, 'french': 1, 'girl': 1, 'began': 1,
 'hearing': 1, 'voices': 1, 'seeing': 1, 'convinced': 1, 'these': 1, 'message
s': 1, 'brazenly': 1, 'demanded': 1, 'order': 1, 'directly': 1, 'give': 1, 't
hen': 1, 'once': 1, 'seated': 1, 'abandons': 1, 'english': 1, 'captors': 1,
 'director': 1, 'luc': 1, 'besson': 1, 'may': 1, 'cowrote': 1, 'script': 1,
 'never': 1, 'appeared': 1, 'proper': 1, 'handle': 1, 'material': 1, 'inconsi
stencies': 1, 'confusing': 1, 'blur': 1, 'violent': 1, 'scenes': 1, 'inapprop
riate': 1, 'musical': 1, 'score': 1, 'lack': 1, 'vibrant': 1, 'life': 1, 'for
ce': 1, 'center': 1, 'film': 1, 'adds': 1, 'up': 1, 'largely': 1, 'disappoint
ing': 1, 'product': 1, 'oftentimes': 1, 'unintentionally': 1, 'laughable': 1,
'biggest': 1, 'miscue': 1, 'wife': 1, 'milla': 1, 'title': 1, 'ms': 1, 'spect
acular': 1, 'clad': 1, 'armor': 1, 'astride': 1, 'similarly': 1, 'protected':
1, 'horse': 1, 'enough': 1, 'fully': 1, 'convey': 1, 'brilliant': 1, "isn't":
1, 'tried': 1, 'failed': 1, 'act': 1, 'part': 1, 'unbalanced': 1, 'inspirin
g': 1, 'troops': 1, 'merely': 1, 'screaming': 1, 'stridently': 1, 'waving':
 1, 'banner': 1, 'sword': 1, 'over': 1, 'head': 1, 'possessed': 1, 'fares':
 1, 'bit': 1, 'better': 1, 'charles': 1, 'vii': 1, 'easily': 1, 'manipulate
d': 1, 'weakness': 1, 'foreshadows': 1, 'betrayal': 1, 'lead': 1, 'faye': 1,
 'dunaway': 1, 'thomas': 1, 'affair': 1, 'gives': 1, 'performance': 1, 'minim
al': 1, 'screen': 1, 'time': 1, "dauphin's": 1, 'motherinlaw': 1, 'chief': 1,
'advisor': 1, 'under': 1, 'comprised': 1, 'comical': 1, 'figures': 1, 'more':
1, 'stooges': 1, 'soldiers': 1, 'exception': 1, 'tcheky': 1, 'karyo': 1, 'l
a': 1, 'femme': 1, 'nikita': 1, 'dunois': 1, 'leading': 1, 'attack': 1, 'prio
r': 1, 'arrival': 1, 'trying': 1, 'plan': 1, 'systematic': 1, 'sees': 1, 'aut
hority': 1, 'negated': 1, 'insistence': 1, 'following': 1, 'dustin': 1, 'hoff
man': 1, 'sphere': 1, 'has': 1, 'small': 1, 'inhuman': 1, 'conscience': 1, 'b
egins': 1, 'speaking': 1, 'awaiting': 1, 'trial': 1, 'dressed': 1, 'cloaked':
1, 'monk': 1, 'leads': 1, 'doubt': 1, 'herself': 1, 'revelations': 1, 'well':
1, 'should': 1, 'do': 1, 'speak': 1, 'via': 1, 'gift': 1, 'able': 1, 'communi
cate': 1, 'three': 1, 'nine': 1, 'manifestations': 1, 'listed': 1, 'corinthia
ns': 1, '12': 1, 'deal': 1, 'receiving': 1, 'knowledge': 1, 'wisdom': 1, 'dis
cerning': 1, 'even': 1, 'themselves': 1, 'result': 1, 'giving': 1, 'spake':
 1, 'moved': 1, 'i': 1, 'e': 1, 'also': 1, 'caution': 1, 'us': 1, 'beloved':
 1, 'believe': 1, 'not': 1, 'every': 1, 'try': 1, 'whether': 1, 'because': 1,
'many': 1, 'false': 1, 'prophets': 1, 'gone': 1, 'out': 1, 'into': 1, 'worl
d': 1, '4': 1, 'kjv': 1, 'burned': 1, 'stake': 1, 'age': 1, '19': 1, 'frenz
y': 1, 'mob': 1, 'rule': 1, 'blood': 1, 'lust': 1, 'inspiration': 1, 'wrough
```

```
t': 1, 'pain': 1, 'suffering': 1, 'followed': 1, 'all': 1, 'point': 1, 'devil
ish': 1, 'influence': 1, 'rather': 1, 'godly': 1, 'conviction': 1, 'intense':
1, 'believing': 1, 'remains': 1, 'admirable': 1, 'quality': 1, 'others': 1,
 'before': 1, 'misled': 1, 'master': 1, 'deception': 1, 'quite': 1, 'effectiv
e': 1, 'just': 1, 'confused': 1, 'whose': 1, 'carrying': 1})
Parameter Vector:
Counter()
Number of kinks in the plot 1
0
```

Findings:

1. There was only one point during the gradient descent when the dot product was zero
2. That was in begining when the feature vector was empty.
3. At any other point in the algorithm its highly improbable to have a zero dot product.
4. Hence, we can skip update when the product goes to zero only if we can find a different way to initialize the parameter vector(other then empty initialization, may be randomly), not otherwise.

Error analysis

7. Detailed analysis can be found below

In [58]:
```
# List of wrongly predicted reviews
wrong_predictions = list()
for data in validation_dataset:
    x=data[0]
    y=data[1]
    if (y*dotProduct(x,w_opt)<0):
        wrong_predictions.append((x,y,dotProduct(x,w_opt)))
i=0
for review in wrong_predictions[:5]:
    features = review[0]  # a dict
    target = review[1]
    score = review[2]
    # get sorted w_i*x_ifor this review
    contributions = Counter()
    print("Wrong prediction {0} has true_value {1} and score of
{2}".format(i,target,score))
    print("Following are the 10 greatest contributors to the predicted value")
    for word,count in features.most_common():
        contributions[word] = abs(count*w_opt[word])
    for word,contribution in contributions.most_common(10):
        print(word, "   ", contribution, "   ", features[word], "   ", w_opt[w
ord])
    i=i+1
```

```
Wrong prediction 0 has true_value -1 and score of 0.2504247812035112
Following are the 10 greatest contributors to the predicted value
and     0.585499321782     17     -0.0344411365754
as     0.520450711948     19     -0.0273921427341
is     0.381820499736     19     -0.0200958157756
the     0.352449692064     50     -0.00704899384128
to     0.316029890551     19     0.0166331521343
only     0.308671730313     3     0.102890576771
have     0.2596997731     4     0.064924943275
he     0.238738291414     9     -0.0265264768238
an     0.225073136687     5     0.0450146273373
planet     0.222908971911     7     0.0318441388444
Wrong prediction 1 has true_value 1 and score of -0.4867515747244316
Following are the 10 greatest contributors to the predicted value
and     0.861028414385     25     -0.0344411365754
to     0.432461955491     26     0.0166331521343
the     0.408841642794     58     -0.00704899384128
is     0.281341420858     14     -0.0200958157756
have     0.2596997731     4     0.064924943275
from     0.235955793845     8     -0.0294944742306
at     0.228968633283     7     0.0327098047547
an     0.180058509349     4     0.0450146273373
i     0.175297346842     9     -0.0194774829825
this     0.165218522315     4     0.0413046305788
Wrong prediction 2 has true_value 1 and score of -0.38107850038448715
Following are the 10 greatest contributors to the predicted value
and     0.103323409726     3     -0.0344411365754
most     0.0702426052956     1     -0.0702426052956
at     0.0654196095095     2     0.0327098047547
though     0.0654196095095     2     -0.0327098047547
have     0.064924943275     1     0.064924943275
from     0.0589889484613     2     -0.0294944742306
very     0.057133950082     1     -0.057133950082
the     0.0563919507303     8     -0.00704899384128
as     0.0547842854682     2     -0.0273921427341
he     0.0530529536476     2     -0.0265264768238
Wrong prediction 3 has true_value 1 and score of -0.5957018128677966
Following are the 10 greatest contributors to the predicted value
joe     0.628473450902     21     -0.0299273071858
and     0.378852502329     11     -0.0344411365754
to     0.365929346954     22     0.0166331521343
is     0.341628868185     17     -0.0200958157756
this     0.33043704463     8     0.0413046305788
also     0.294326409513     4     -0.0735816023783
the     0.27491075981     39     -0.00704899384128
he     0.238738291414     9     -0.0265264768238
they     0.227608301138     9     -0.0252898112376
one     0.200401658242     7     -0.0286288083203
Wrong prediction 4 has true_value -1 and score of 0.7082383812110735
Following are the 10 greatest contributors to the predicted value
and     0.516617048631     15     -0.0344411365754
have     0.454474602925     7     0.064924943275
to     0.365929346954     22     0.0166331521343
i     0.331117210702     17     -0.0194774829825
only     0.308671730313     3     0.102890576771
nothing     0.294202742955     3     0.0980675809849
the     0.232616796762     33     -0.00704899384128
```

```
very     0.228535800328     4     -0.057133950082
script     0.209058317345     3     0.0696861057818
even     0.206523152894     4     0.0516307882234
```

As is evident from the results above, the first file has huge count of the word "the" and "and"; while the second file has high counts for the words 'and', 'the', 'as' etc; Words like these do not usually indicate anything about sentiment but contribute largely in our prediction model due to their high count in the review text. Removing these words from our dataset can help improve the accuracy.

Features

In [59]:
```
import nltk
```

In [60]:
```
# Download stopwords from nltk
```

In [61]:
```
from nltk.corpus import stopwords
```

In [62]:
```
s_words = set(stopwords.words('english'))
def remove_s_words(bw):
    '''
    Accepts a bag of words representation of input
    Returns a bag of words with stop words removed.
    '''
    s_words = set(stopwords.words('english'))
    for key,value in bw.most_common():
        if key in s_words:
            del bw[key]
    return bw
```

In [64]:
```
# Generate a filtered dataset and use it for training model and test for impro
vement in accuracy
def new_pegasos_optimized(Lambda, data, max_iters, remove_stopwords=False):
    '''
    Args
        Lambda - Regularization parameter
    Returns:
        w - a asparse weight vector w
    '''
    iters = 0
    t=1
    w=Counter()
    s=1
    # a simple termination condition for now
    while(iters<max_iters):
        iters = iters+1
        for data in train_data:
            t=t+1
            eta = 1.0/(t*Lambda)
            y = data[1]
            x = data[0]
            if remove_stopwords:
                x = remove_s_words(x)
            # if s==0 reset the parameter
            if (s==0):
                s=1
                w=Counter()
            if(y*dotProduct(x,w)<1/s):
                s = (1-eta*Lambda)*s
                increment(w, ((1/s)*eta*y) ,x)
            else: s = (1-eta*Lambda)*s
    k = Counter()
    increment(k, s, w)
    return k
```

In [68]:
```
# Get optimal predictor
w_opt_5 = new_pegasos_optimized(lambda_opt, train_data, 31)
print("Loss with no optimization")
print(compute_loss(w_opt_5, validation_dataset))
w_opt_6 = new_pegasos_optimized(lambda_opt, train_data, 31, remove_stopwords=T
rue)
print("Loss with stop words removed")
print(compute_loss(w_opt_6, validation_dataset))
```

```
Loss with no optimization
15.8
Loss with stop words removed
13.8
```

As indicated in the analysis above, I removed the words which do not contribute towards sentiment, but get weighted higly due to high count. Addition of this feature lead to a 6% improvement in the error as can be seen above. Standard Error without the feature - percent_error - 24.8 numer of wrong classification = 124 p = 0.248 standard error = 0.0193130008 Standard Error with the feature : percent_error - 18.8 numer of wrong classification = 94 p = 0.188 standard error = 0.01747317944 Improvement in standard error = 0.002

Adding Features to improve performance

In [69]:
```
# Adding a n-grams to our featureset
def add_ngram(review):
    '''
    Accpets a review and adds all consequtive words as a feature
    '''
    prev = ' '
    ngrams = list()
    for word in review[:-1]:
        ngram = prev + " " + word
        ngrams.append(ngram)
        pre = word
    return ngrams + review

def not_features(review):
    '''
    Accpets a review and combines not and following word to form a
    feature that is added to original list of features
    '''
    prev = ' '
    ngrams = list()
    for word in review[:-1]:
        if prev == 'not':
            ngram = prev + " " + word
            ngrams.append(ngram)
            pre = word
    return ngrams + review
```

In [72]:
```
# Create updated datasets
reviews
updated_reviews = list(map(lambda x:add_ngram(x), reviews))
updated_reviews2 = list(map(lambda x:not_features(x), updated_reviews))
# split into train and test set
updated_train = bag_of_words(updated_reviews2[:1500])
updated_validation = bag_of_words(updated_reviews2[1500:2000])

#
w_opt_7 = new_pegasos_optimized(lambda_opt, updated_train, 31, remove_stopword
s=True)
print(len(w_opt_7))
print(compute_loss(w_opt_7, updated_validation))
```

```
38903
14.5
```

There is slight improvement with additione of new features

In [ ]:

Assignment-3 solutions

2.1) given, $g \in \partial f_K(x)$

$$\Rightarrow f_K(z) \geq f_K(x) + g^T(z-x)$$

$$\geq f(x) + g^T(z-x) \quad [\because f_K(x) = f(x)].$$

$$\Rightarrow f_K(z) \geq f(x) + g^T(z-x) \quad — (1)$$

also, $f(z) \geq f_K(z)$ $\because$ f is max of all f_{is}

$$\Rightarrow f(z) \geq f(x) + g^T(z-x)$$

$$\Rightarrow \quad g \in \partial f(x).$$

2.2) Using above:

Let $f_1 = 0$

$f_2 = 1 - y w^T x$

$$J(w) = \begin{cases} f_2 & \text{if } y w^T x < 1 \\ 0 & \text{if } y w^T x \geq 1 \end{cases}$$

$\Rightarrow$

Sub gradient = gradient if gradient exist $\Rightarrow$

$$-yx \in \partial f_2(w)$$

$\Rightarrow$ subgradient of $J(w) = \begin{cases} -yx & \text{if } yw^Tx \leq 1 \\ 0 & yw^Tx > 1 \end{cases}$

3.1) $\{x \mid w^Tx = 0\} \rightarrow$ a perfectly separating hyperplane

$\Rightarrow y_i w^T x_i > 0 \quad \forall \; i \in \{1, \dots n\}$

$\Rightarrow$ Perceptron loss $= \max\{0, -\hat{y}y\}$

$= \max\{0, -y_i w^T x_i\}$

$= 0$

$\Rightarrow$ perceptron loss is a point wise zero function.

$\Rightarrow$ average Loss $= 0$.

3.2) Empirical Risk for perceptron loss :

$$J(w) = \frac{1}{n}\sum_{i=1}^{n} \max\{0, -y_i w^T x_i\}$$

which is minimized when loss for a point x_i, y_i

$$l(w) = \max\{0, -y_i w^T x_i\}$$ is minimized.

⇒ The update for w for SSGD mentioned would be :

$$w_{t+1} = w_t - \eta \nabla_w l(w).$$

$$\Rightarrow w_{t+1} = w_t - (1)\,\partial\big(l(w)\big).$$

⇒ a sub gradient of $l(w)$ as calculated in previous problem is :

$$g = \begin{cases} -y_i x_i & \text{if } y_i w^T x_i \leq 0 \\ 0 & \text{if } y_i w^T x_i > 0 \end{cases}$$

⇒ The update becomes :

$$w_{t+1} = w_t - \begin{cases} -y_i x_i & \text{if } y_i w^T x_i \leq 0 \\ 0 & \text{if } y_i w^T x_i > 0 \end{cases}$$

which is implemented by the given algo.

3.3) w is linear combination of x_i :

since w is initialized with $w = 0$ & updated as

$$w_{K+1} = w_K + (y_i) x_i$$

$\Rightarrow$ we are appending either $+x_i$ or $-x_i$ $[y_i = \pm 1]$

$\Rightarrow$ w is a linear combination of x_i.

(ii) Characterization points:

a point (x_i) contributes towards w $(\alpha_i \neq 0)$

iff

x_i was ever misclassified during complete run of algorithm. $\left(y_i x_i^T w \leq 0 \right)$

$\rightarrow$ it doesn't contribute $(\alpha_i = 0)$.

iff

it was never misclassified

6.1) Subgradient of "Stochastic" SVM objective :

i.e of $\frac{\lambda}{2} \|w\|^2 + \max\{0, 1 - y_i w^T x_i\}$.

(since this part is differentiable).

$$= 2 \cdot \frac{\lambda}{2} \cdot w + \partial\left(\max\{0, 1 - y_i w^T x_i\}\right)$$

$$= \lambda w + \begin{cases} -y_i x_i & \text{if } y_i w^T x_i < 1 \\ 0 & \text{if } y_i w^T x_i \geq 1 \end{cases}$$

→ update :

$$w_{t+1} = w_t - \eta g$$

$$= w_t - \left(\frac{1}{\lambda t}\right) \cdot \begin{cases} (\lambda w - y_i x_i) & \text{if } y_i w^T x_i < 1 \\ \lambda w & , y_i w^T x_i \geq 1 \end{cases}$$

$$\Rightarrow w_{t+1} = \begin{cases} w_t - \eta(\lambda w - y_i x_i) & , y_i w^T x_i < 1 \\ w_t - \eta \lambda w & , y_i w^T x_i \geq 1 \end{cases}$$

similar to Pegasos update.

6.3 $W_{t+1} = w_t + \frac{1}{s_{t+1}} \eta_t y_j x_j$

$\Rightarrow s_{t+1} W_{t+1} = s_{t+1} W_t + \eta_t y_j x_j$

$\Rightarrow w_{t+1} = (1 - \eta_t \lambda) s_t W_t + \eta_t y_j x_j$

$$\Rightarrow \boxed{W_{t+1} = (1 - \eta_t \lambda) w_t + \eta_t y_j x_j}$$