Machine Learning and Computational Statistics, Sprint 2015 Homework 3: SVM and Sentiment Analysis

Charlie Guthrie

Due Monday, Feb 23, 2015 at 4pm.

1 Introduction

In [26]:
```
#Prep
import os
import numpy as np
import pandas as pd
import pickle
import random
import collections
from datetime import datetime
import matplotlib.pyplot as plt
%matplotlib inline
#pd.options.display.mpl_style = 'default'
```

2 The Data

1. Load all the data and randomly split it into 1500 training examples and 500 test examples.

In [55]:
```
def folder_list(path,label):
    '''
    Take path to a directory, apply the label the contained documents,
    and return the list of documents
    PARAMETER PATH IS THE PATH OF YOUR LOCAL FOLDER
    Label is positive or negative
    '''
    filelist = os.listdir(path)
    review = []
    for infile in filelist:
        myfile = os.path.join(path,infile)
        r = read_data(myfile)
        r.append(label)
        r.append(infile) #add file id for later reference
        review.append(r)
    return review
```

```
def read_data(file_path):
    '''
    Read each file into a list of strings.
    Example:
    ["it's", 'a', 'curious', 'thing', "i've", 'found', 'that', 'when', '
willis', 'is', 'not', 'called', 'on',
    ...'to', 'carry', 'the', 'whole', 'movie', "he's", 'much', 'better',
 'and', 'so', 'is', 'the', 'movie']
    '''
    f = open(file_path)
    lines = f.read().split(' ')
    symbols = '${}()[].,:;+-*/&|<>=~"_`\' '
    words = map(lambda Element: Element.translate(None, symbols).strip()
, lines)
    words = filter(None, words)
    return words

def train_test_split(data,pivot_index):
    '''
    split list of documents into training and test sets
    '''
    random.seed(1)
    random.shuffle(data)
    train = data[:pivot_index]
    test = data[pivot_index:]
    assert len(train)==pivot_index, "Training set is not proper length"
    return train,test

def load_data():
    '''
    pos_path is where you save positive review data.
    neg_path is where you save negative review data.
    '''
    home_dir = os.getcwd()
    pos_path = os.path.join(home_dir,"data/pos")
    neg_path = os.path.join(home_dir,"data/neg")

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

    review = pos_review + neg_review
    return train_test_split(review,1500)

train,test = load_data()
```

3 Sparse Representations

1. Write a function that converts an example (e.g. list of words) into a sparse bag-of-words representation.

In [262]:
```
def lists2bags(lists):
    '''Converts list of words into sparse bag-of-words representation'''

    bags = []
    labels = []
    ids = []
    for review in lists:
        #don't include the review's label or id
        bag = collections.Counter(review[:-2])
        bags.append(bag)
        labels.append(review[-2])
        ids.append(review[-1])
    return bags,labels,ids
```

---

2. Write a version of `generic_gradient_checker` from Homework 1 that works with sparse vectors represented as dict types. Since we'll be using it for stochastic methods, it should take a single (x, y) pair, rather than the entire dataset. Be sure to use the dotProduct and increment primitives we provide, or make your own.

In [263]:
```
def dotProduct(d1, d2):
    """
    @param dict d1: a feature vector represented by a mapping from a fea
ture (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):
    """
    MODIFIED: Does not add keys to d1
    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.
    """
    for f, v in d2.items():
        current_val = d1.get(f)
        if current_val is not None:
            d1[f] = (current_val + v * scale)

def grad_checker(x, y, w, Lambda, obj_func,grad_func,epsilon=0.01, toler
ance=1e-4):
    """Implement Gradient Checker
    Check that the function compute_square_loss_gradient returns the
    correct gradient for the given x, y, and theta.

    Let d be the number of features. Here we numerically estimate the
```

```
    gradient by approximating the directional derivative in each of
    the d coordinate directions:
    (e_1 = (1,0,0,...,0), e_2 = (0,1,0,...,0), ..., e_d = (0,...,0,1)

    The approximation for the directional derivative of J at the point
    theta in the direction e_i is given by:
    ( J(theta + epsilon * e_i) - J(theta - epsilon * e_i) ) / (2*epsilon
).

    We then look at the Euclidean distance between the gradient
    computed using this approximation and the gradient computed by
    compute_square_loss_gradient(X, y, theta).  If the Euclidean
    distance exceeds tolerance, we say the gradient is incorrect.

    Args:
        x - sparse feature dict
        y - the label vector, 1D numpy array of size (num_instances)
        w - sparse parameter dict
        Lambda - regularization hyperparameter
        epsilon - the epsilon used in approximation
        tolerance - the tolerance error

    Return:
        A boolean value indicate whether the gradient is correct or not

    """

    true_gradient = grad_func(x, y, w,Lambda) #the true gradient
    approx_grad = {} #Initialize the gradient we approximate
    dist = 0
    for key in w:
        w_plus = w.copy()
        w_minus = w.copy()
        w_plus[key]+= -epsilon
        w_minus[key]+= epsilon

        approx_grad[key] = (obj_func(x,y,w_plus,Lambda) - obj_func(x,y,w
_minus))*1.0 / (2.0*epsilon)
        dist+=(approx_grad[key] - true_gradient[key])**2

    correct_grad = dist<tolerance
    assert correct_grad, "Gradient bad: dist %s is greater than toleranc
e %s" % (dist,tolerance)
    return correct_grad
```

4 Support Vector Machine via Pegasos

1. [Written] Compute a subgradient for the "stochastic" SVM objective, which assumes a single training point. Show that if your step size rule is $\eta_t = 1/(\lambda t)$, then the corresponding SGD update is the same as given in the pseudocode.

The objective function $J(w)$ for example j is as follows:

$$J(w) = \begin{cases} 1 - y_j w^T x_j + \frac{\lambda}{2}\|w\|^2 & \text{if } y_j w^T x_j < 1; \\ \frac{\lambda}{2}\|w\|^2 & \text{otherwise.} \end{cases}$$

Taking the gradients of both parts with respect to w, then:

$$\nabla J(w) = \begin{cases} \lambda w - x_j y_j & \text{if } y_j w^T x_j < 1; \\ \lambda w & \text{otherwise.} \end{cases}$$

Updating w according to the equation $w_{t+1} = w_t + \eta_t \nabla J(w_t)$, where η_t is step size, we have:

$$w_{t+1} = \begin{cases} (1 - \eta_t \lambda) w_t + \eta_t y_j x_j & \text{if } y_j w_t^T x_j < 1; \\ (1 - \eta_t \lambda) w_t & \text{otherwise.} \end{cases}$$

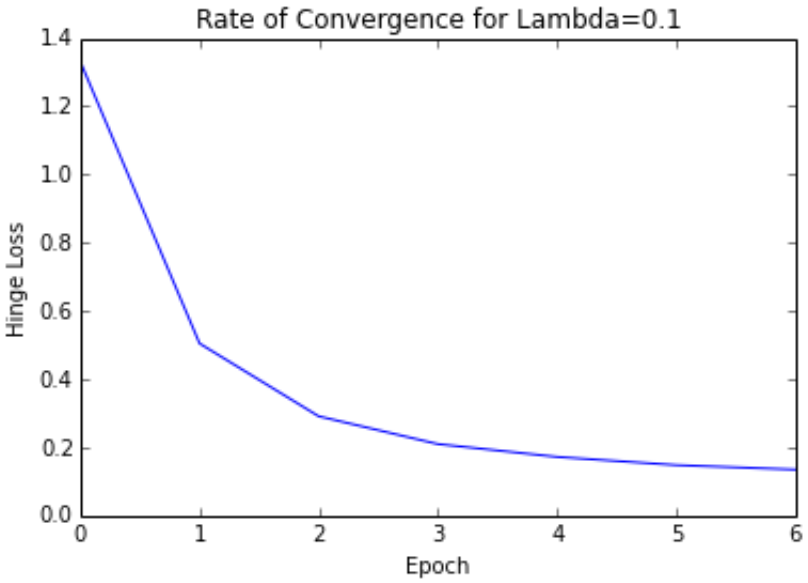
2. Implement the Pegasos algorithm to run on a sparse data representation. The output should be a sparse weight vector w. [As should be your habit, please check your gradient using generic gradient checker while you are in the testing phase. That will be our first question if you ask for help debugging. Once you're convinced it works, take it out so it doesn't slow down your code.]

In [253]:
```
def count_corpus_words(X):
    '''count words in entire corpus'''
    c = collections.Counter()
    for review in X:
        c.update(review)
    return c

def initialize_w(min_occurrences):
    '''
    initialize w to a dictionary where keys are all words and values are
```

```
 zeros.
    remove single instances
    Stopwords list from http://www.ranks.nl/stopwords

    '''
    full_dictionary = count_corpus_words(X)

    w={}
    for k, v in full_dictionary.items():
        if v>min_occurrences:
            w[k]=0
    return w

def remove_stopwords(w):
    '''
    remove stopwords
    Stopwords list from http://www.ranks.nl/stopwords
    '''
    f = open('stopwords.txt')
    stopwords = f.read().split(',')

    for k in stopwords:
        w.pop(k, None)
    return w

def compute_svm_cost(X,y,w,Lambda):
    """
    Given a set of X, y, theta, compute the square loss for predicting y
 with X*theta

    Args:
        x - sparse feature dict
        y - the label vector, 1D numpy array of size (num_instances)
        w - sparse parameter dict
        Lambda - regularization hyperparameter

    Returns:
        loss - the square loss, scalar
    """
    m = len(X)
    loss = 0
    for i in range(m):
        loss+=max(0,1-y[i]*dotProduct(w,X[i]))

    return Lambda/2.0*dotProduct(w,w) + 1.0/m*loss

def pegasos(X,y,Lambda,w_init,max_epochs=30,delta=0.1, plot=False, remov
e_stops=False):
    t1 = datetime.now()
    W=w_init.copy()
    W=remove_stopwords(W) if remove_stops==True else W
    t=s=1
    epoch=0
    epochs=[]
    losses=[]
    m=len(X)
```

```
    convergence=False
    loss=10000
    while epoch<max_epochs and convergence==False:
        prev_loss=loss
        epochs.append(epoch)
        for j in range(1,m):
            t+=1
            eta = 1.0/(t*Lambda)
            ywx=y[j]*s*dotProduct(W,X[j])
            s=(1-eta*Lambda)*s
            if ywx<1:
                increment(W,eta*y[j]/s,X[j])

        #Scale w:
        w={}
        for f,v in W.items():
            w[f]=v*s

        loss = compute_svm_cost(X,y,w,Lambda)
        losses.append(loss)
        if len(losses)>=4:
            loss_diff = max(losses[-4:])-min(losses[-4:])
            convergence=(loss_diff<delta)
        epoch+=1

    t2 = datetime.now()
    print "Time taken: %i seconds" %(t2-t1).seconds

    if plot:
        plot_convergence(epochs,losses,Lambda)

    return w

def plot_convergence(epochs,losses,Lambda):
    plt.plot(epochs,losses)
    plt.xlabel('Epoch')
    plt.ylabel('Hinge Loss')
    plt.title('Rate of Convergence for Lambda=%s' %Lambda)
    plt.show()

#X,y,ids=lists2bags(train)
X_test,y_test,ids_test=lists2bags(test)
w=initialize_w(1)
w = pegasos(X,y,0.1,w,30, delta=0.1, plot=True, remove_stops=True)
```

```
Time taken: 18 seconds
```



---

3. Write a function that takes the sparse weight vector w and a collection of (x, y) pairs, and returns the percent error when predicting y using sign$(w^T x)$ (that is, report the 0-1 loss).

In [216]:
```
def get_pct_error(X,y,w):
    num_correct=0
    for i in range(len(X)):
        yhat = np.sign(dotProduct(w,X[i]))
        num_correct+=max(y[i]*yhat,0)
    return 1-float(num_correct)/len(y)

get_pct_error(X_test,y_test,w)
```

Out[216]:
```
0.15200000000000002
```

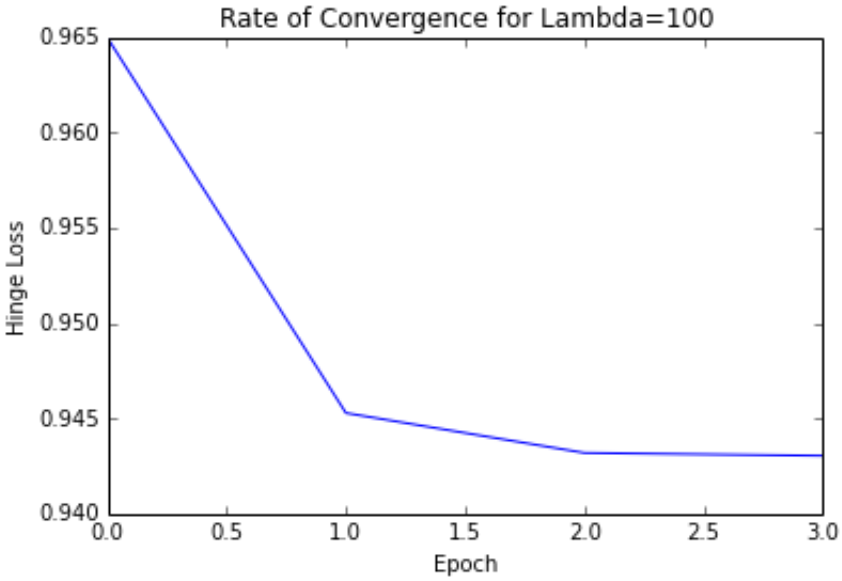
---

4. Using the bag-of-words feature representation described above, search for the regularization parameter that gives the minimal percent error on your test set. A good search strategy is to start with a set of lambdas spanning a broad range of orders of magnitude. Then, continue to zoom in until you're convinced that additional search will not significantly improve your test performance. Once you have a sense of the general range of regularization parameters that give good results, you do not have to search over orders of magnitude every time you change something (such as adding new feature).

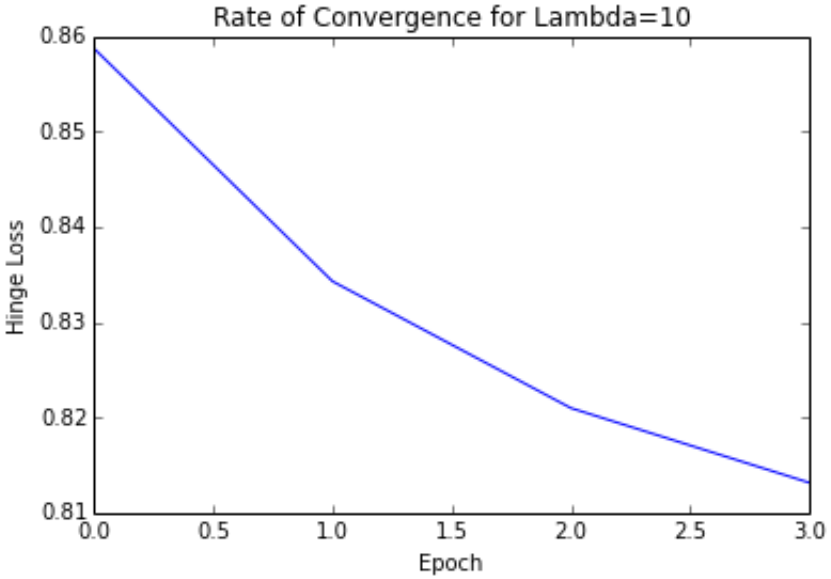
In [255]:
```
def Lambda_search(opt_func,plot_bool):
    """
    Runs an optimization function on a training set with a variety of
    regularization hyperparameters Lambda.
    Selects the lambda that minimizes square error on the validation set
.
    Plots validation error vs. Lambda
    Prints selected lambda along with corresponding test error.
    """
    #loop through array of lambdas
    t=0
    Lambdas=[]
    loss_hist=[]
    print "Starting Loop"
    w=initialize_w(1)
    for i in range(2,-4,-1):
        Lambda = 10**i
        print "Lambda",Lambda
        Lambdas.append(Lambda)
        w=opt_func(X,y,Lambda,w,plot=plot_bool)
        loss=get_pct_error(X_test,y_test,w)
        loss_hist.append(loss)

        if t==0 or loss<=loss_opt:
            loss_opt = loss
            lambda_opt = Lambda
            w_opt = w.copy()
        t=t+1

    print "Best Lambda:",lambda_opt
    print "Pct Error on Test Data:", loss_opt

    return w_opt,Lambdas,loss_hist
```

In [144]:
```
w_opt,Lambdas,loss_hist=Lambda_search(pegasos,True)
plt.plot(Lambdas,loss_hist)
plt.xlabel('Lambda')
plt.ylabel('Pct. Error')
plt.xscale('log')
plt.title('Pct. Error vs. Lambda')
plt.show()
```

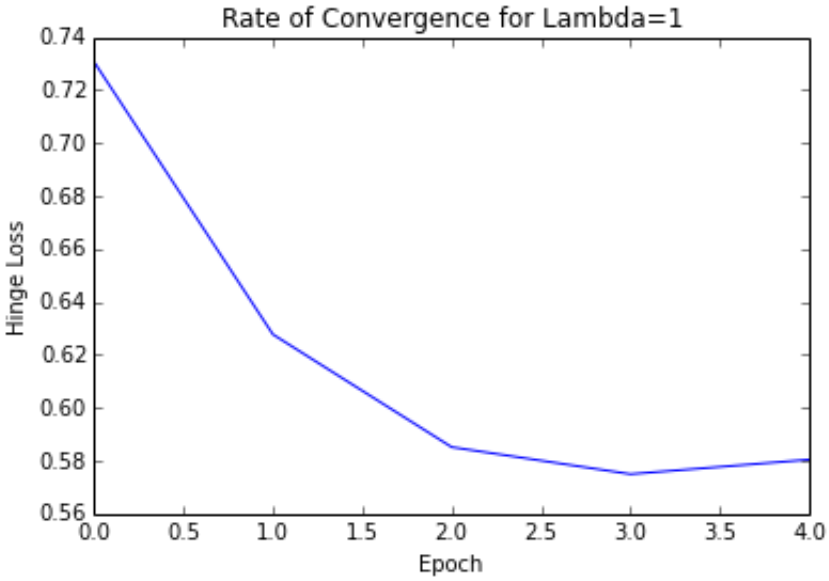
```
Starting Loop
Lambda 100
Time taken: 3 seconds
```



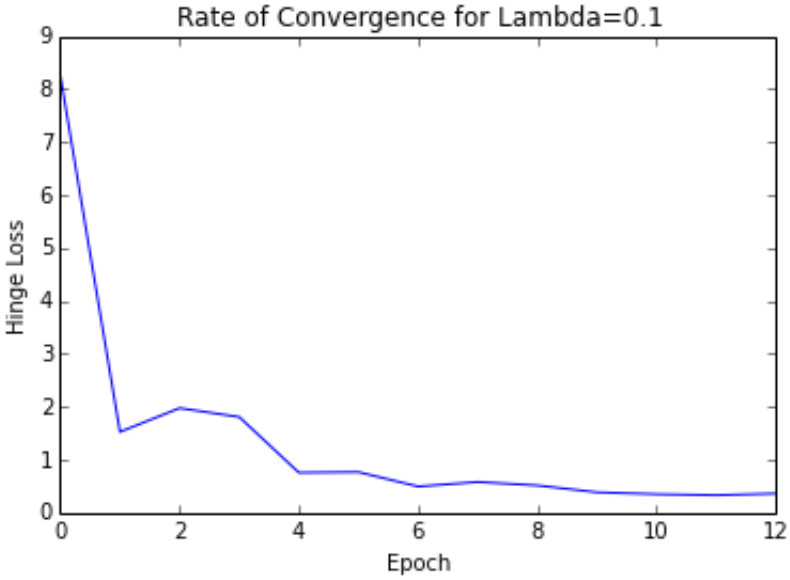

```
Lambda 10
Time taken: 3 seconds
```



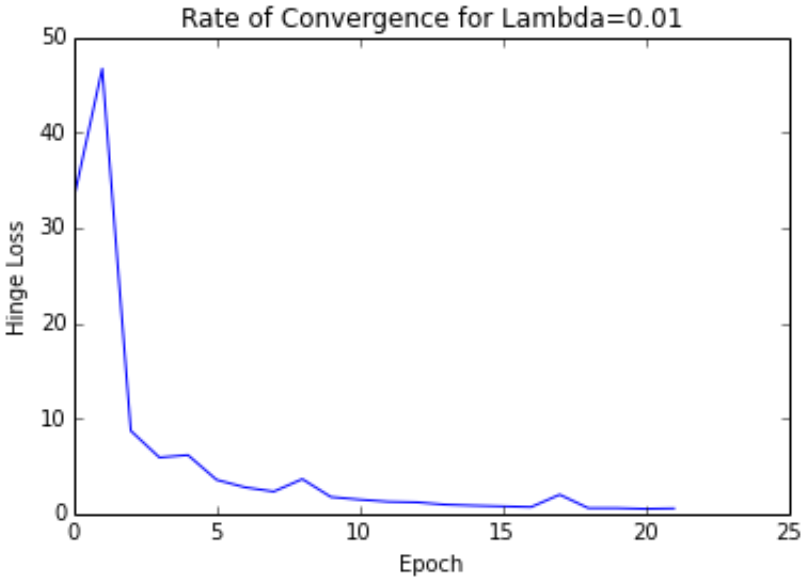

```
Lambda 1
Time taken: 3 seconds
```



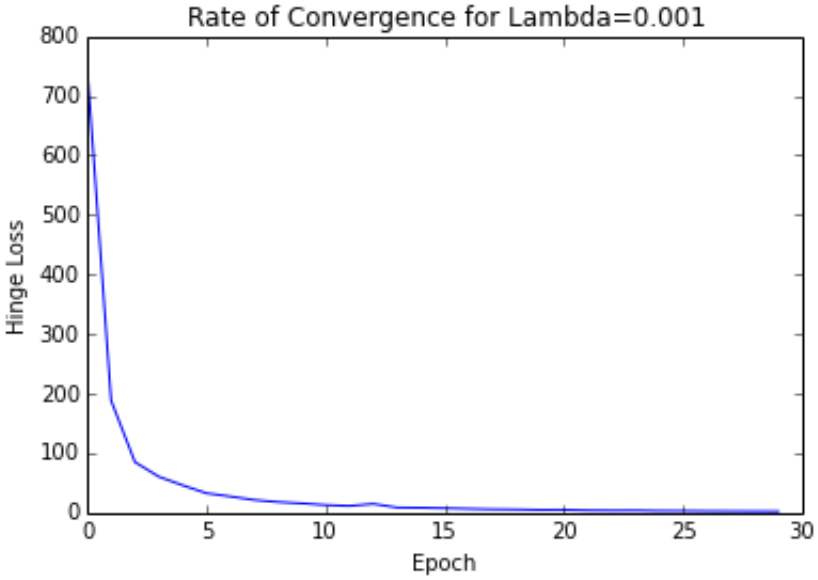
Lambda 0.1
Time taken: 8 seconds



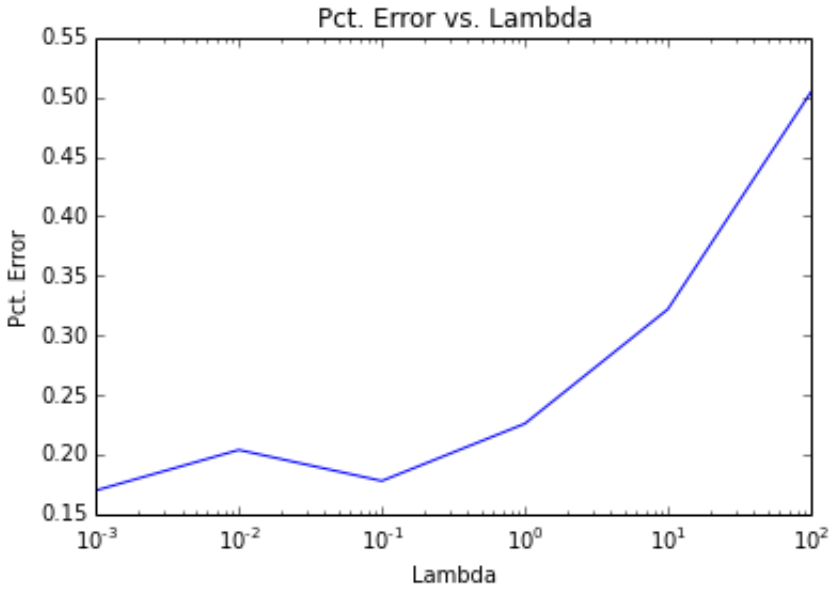
Lambda 0.01
Time taken: 14 seconds




```
Lambda 0.001
Time taken: 20 seconds
```




```
Best Lambda: 0.001
Square Loss on Test Data: 0.17
```

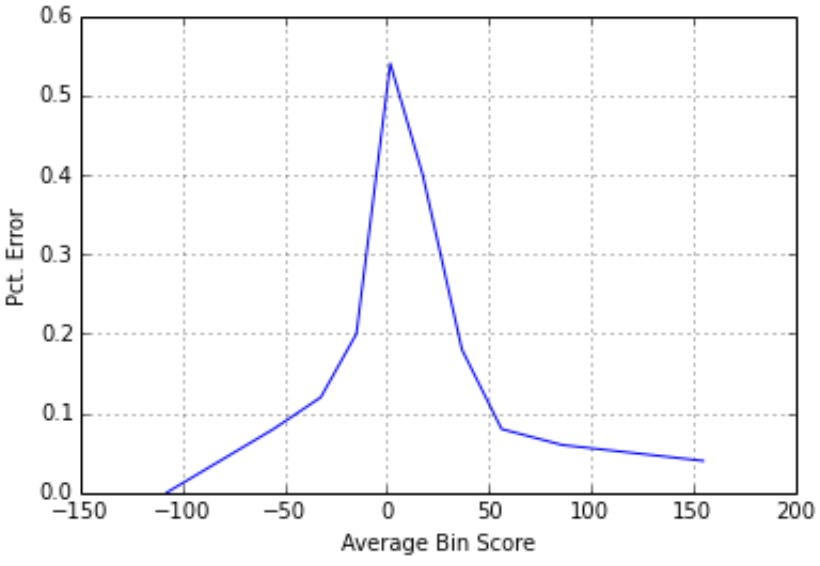


5. Recall that the "score" is the value of the prediction $f(x) = w^T x$. We like to think that the magnitude of the score represents the confidence of the prediction. This is something we can directly verify or refute. Break the predictions into groups based on the score (you can play with the size of the groups to get a result you think is informative). For each group, examine the percentage error. You can make a table or graph. Summarize the results. Is there a correlation between higher magnitude scores and accuracy?

> See below for graph. I find that higher magnitude scores give considerably better accuracy.

In [145]:
```
def discretize_score(X,y,w,bins=10):
    scores=[]
    for i in range(len(X)):
        score = dotProduct(w,X[i])
        scores.append(score)
    score_df=pd.DataFrame(zip(scores,y))
    score_df.columns=('score','label')
    score_df['prediction']=np.sign(score_df['score'])
    score_df = score_df.sort('score')
    #sorted_scores = np.sort(score_array,axis=0)

    index=0
    score_df['bin']=0
    summary = pd.DataFrame(columns=('avg_score','pct_error'))
    chunk = score_df.shape[0]/bins

    #for each bin...
    for i in range(bins):
        score_df['bin'].iloc[index:index+chunk]=i
        temp = score_df.iloc[index:index+chunk]
        assert temp.shape[0]==score_df.shape[0]/bins,"Temporary DF is wr
ong size"
        #record avg score
        summary.loc[i,'avg_score']=temp['score'].mean()

        #get pct error
        pct_error = float(temp[temp.prediction != temp.label].shape[0])/
temp.shape[0]
        summary.loc[i,'pct_error']=pct_error
        index+=chunk
    return score_df,summary
score_df,score_summary=discretize_score(X_test,y_test,w_opt)

score_summary.plot(x='avg_score',y='pct_error')
plt.xlabel('Average Bin Score')
plt.ylabel('Pct. Error')
```

Out[145]: <matplotlib.text.Text at 0x107a92290>



5 Error Analysis

The natural language processing domain is particularly nice in that one can often interpret why a model has performed well or poorly on a specific example, and sometimes it is not very difficult to come up with ideas for new features that might help fix a problem. The first step in this process is to look closely at the errors that our model makes.

1. Choose some examples that the model got wrong. List the features that contributed most heavily to the decision (e.g. rank them by $|w_i x_i|$), along with x_i, w_i, xw_i. Do you understand why the model was incorrect? Can you think of a new feature that might be able to fix the issue? Include a short analysis for at least 3 incorrect examples.

   The model incorrectly classified *cv082_11080* as negative. It had several instances of "no" and "problems", which the model interprets as indicating negative sentiment. But in two cases the review was actually saying there were in fact "no problems". Using bigrams or trigrams, rather than just bags of words, might help.

   The next two reviews I sampled, *cv507_9220* and *cv118_28980*, were heavily influenced by stopwords like 'and' and 'then'. Removing stopwords might help, or at least make the model more interpretable.

   The one notable thing I found in the 3rd review I sampled, *cv118_28980*, was frequent mention of 'apes', which counted against it. Perhaps movies about apes tend to fare poorly? I know for example that no film about apes has ever won an Oscar - they tend to favor historical dramas and the like.

In [208]:
```
def load_file(ID,label):
    '''
    Display a single file from its ID
    '''
    home_dir = os.getcwd()
    label_dir = "data/pos" if label==1 else "data/neg"
    myfile = os.path.join(home_dir,label_dir,ID)

    #Make sure file is available
    assert os.path.isfile(myfile), "File not found"

    f = open(myfile)
    lines = f.read()
    return lines

score_df['abs_score']=abs(score_df['score'])
wrongs = score_df[score_df['label']!=score_df['prediction']]
wrongs.sort('abs_score',ascending=False)
i=wrongs.index[1]

def example_analysis(i):
    '''
    Print the review and a list of the most contributing words

    Parameters: takes i as the index of the review
    '''
    example=X_test[i]
    feature_dict = {}
    for key in example:
        w=w_opt[key] if (key in w_opt.keys()) else np.nan
        x=example[key]
        feature_dict[key]={'abs_xw':abs(x*w),'x':x,'w':w,'xw':x*w}
    print "Review file name:",ids_test[i]
    print "True sentiment: ",y_test[i], ".  Predicted sentiment: ",wrong
s.loc[i,'prediction']
    print load_file(ids_test[i],y_test[i])

    feature_df = pd.DataFrame.from_dict(feature_dict,'index')
    return feature_df.sort('abs_xw',ascending=False)[:10]
```

In [209]:
```
wrongs.head()
```

Out[209]:

	score	label	prediction	bin	abs_score
72	-59.816471	1	-1	1	59.816471
224	-55.836129	1	-1	1	55.836129
174	-47.363851	1	-1	1	47.363851
13	-46.029737	1	-1	1	46.029737
392	-31.998445	1	-1	2	31.998445

In [199]:
```
wrongs[(wrongs['label']==-1) & (wrongs['prediction']==1)].head()
```

Out[199]:

	score	label	prediction	bin	abs_score
15	1.267489	-1	1	4	1.267489
63	1.556621	-1	1	4	1.556621
450	1.556641	-1	1	4	1.556641
321	2.245933	-1	1	4	2.245933
454	2.401583	-1	1	4	2.401583

In [204]:
```
example_analysis(72)
```

```
Review file name: cv082_11080.txt
True sentiment:  1 .  Predicted sentiment:  -1.0
a big surprise to me .
the good trailer had hinted that they pulled the impossible off , but mak
ing a blues brothers movie without jake blues ( john belushi ) is such a
dumb idea i really had no hope .
they replaced him just fine .
not with john goodman , he didn't do much of anything , but with the bril
liant actor joe morton , who can really sing .
the fourth blues brother is j evan bonifant , who's ten .
this was another of my fears for the film , but he's a really good dancer
 , and plays a mean harmonica ( although he may have been dubbed ) .
things that intellectually had bothered me before , like no mission from
god , everyone being richer , it not being filmed in chicago -- gave me n
o problems at all .
i'm quite pleased that there were less car pile-ups , because they meant
less music , and john landis seems to have lost interest in the whole thi
ng .
there's a few early crashes , and then one huge pile-up , but after that
it all stops .
it's just the music .
one of my problems with the first is that cab calloway's song is so good
the actually blues brothers look dull after him , but there's no problems
 with this .
the music is all as good as ever , tons of great musicians showing up --
with the exception of johnny lang , who can't sing , all the musicians do
 a great job .
the only real problems i had was the special effects .
these were superfluous and a waste of money .
since the film isn't doing very well , they could mean we have no possibi
lity of another sequel , which i want to see .
the bluegrass version of riders in the sky is even better than rawhide .
-- http : //www . geocities . com/hollywood/academy/8034/
remove no spam to reply .
 " drive carefully but recklessly " , mama , child's toy " the only excer
cise i take is walking behind the coffins of friends who took exercise .
"
peter o'toole
```

Out[204]:

	x	w	abs_xw	xw
no	6	-2.534977	15.209861	-15.209861
good	4	2.979705	11.918818	11.918818
problems	4	-2.579447	10.317789	-10.317789
and	4	2.512735	10.050940	10.050940
only	2	-4.825348	9.650695	-9.650695
all	4	-2.045769	8.183077	-8.183077
great	2	3.780226	7.560452	7.560452
but	6	-1.223012	7.338072	-7.338072
have	3	-2.268136	6.804409	-6.804409
had	4	-1.623273	6.493092	-6.493092

In [205]:
```
example_analysis(224)
```

```
Review file name: cv507_9220.txt
True sentiment:  1 .  Predicted sentiment:  -1.0
capsule : side-splitting comedy that follows its own merciless logic almo
st through to the end . . .
but not without providing a good deal of genuine laughs .
most comedies these days have one flaw .
they're not funny .
they think they're funny , but they are devoid of anything really penetra
ting or dastardly .
occasionally a good funny movie sneaks past the deadening hollywood preco
nceptions of humor and we get a real gem : ruthless people , for instance
 , which established a microcosm of a setup and played it out to the bitt
er end .
liar liar is built the same way and is just about as funny .
this is one of the few movies i've seen where i was laughing consistently
 almost all the way through : instead of a couple of set-pieces that insp
ired a laugh ( think of the dismal fatal instinct ) , the whole movie wor
ks like clockwork .
jim carrey playes a high-powered lawyer , to whom lying is as natural as
breathing .
there is one thing he takes seriously , though : his son , and we can sen
se the affection that they have for each other right away .
but his wife is divorced and seeing another man , and now it looks like t
hey may move away together .
the son goes with them , of course .
the movie sets up this early material with good timing and a remarkable b
alance of jim carrey's over-the-top persona with reality .
then the plot springs into action : after being snubbed ( not deliberatel
y ) by his father at his birthday , the kid makes a wish as he blows out
the birthday candles : that for just one day , dad can't lie .
he gets the wish .
what happens next is sidesplitting .
everything turns into a confrontation : when cornered by a bum for some c
hange , he shouts , " no !
```

```
i'm not giving you any money because i know you'll spend it on booze !
all i want to do is to get to the office without having to step over the
debris of our decaying society ! "
he can't even get into an elevator without earning a black eye .
and what's worse , he's now gotten himself into an expensive divorce sett
lement that requires him to twist the truth like abstract wire sculpture
.
carrey , who i used to find unfunny , has gotten better at his schtick ,
even if it's a limited one .
he uses it to great effect in this movie .
there is a scene where he tries to test his ability to lie and nearly dem
olishes his office in the process ( there's a grin breaking out across my
 face right now , just remembering the scene ) .
he can't even write the lie ; his fingers twitch , his body buckles like
someone in the throes of cyanide poisoning , and when he tries to talk it
's like he's speaking in tongues .
equally funny is a scene where he beats himself to a pulp ( don't ask why
 ) , tries to drink water to keep from having outbursts in the courtroom
( it fails , with semi-predictable results ) , and winds up biting the bu
llet when he gets called into the boardroom to have everyone ask what the
y think of them .
this scene alone may force people to stop the tape for minutes on end .
the movie sustains its laughs and also its flashes of insight until almos
t the end .
a shame , too , because the movie insists on having a big , ridiculous cl
imax that involves carrey's character flagging down a plane using a set o
f motorized stairs , then breaking his leg , etc . a simple reconciliatio
n would do the trick .
why is this stupid pent-up climax always obligatory ?
it's not even part of the movie's real agenda .
thankfully , liar liar survives it , and so does carrey .
maybe they were being merciful , on reflection .
if i'd laughed any more , i might have needed an iron lung .
```

Out[205]:

	x	w	abs_xw	xw
and	13	2.512735	32.665556	32.665556
he	11	2.379319	26.172509	26.172509
to	18	-0.756045	13.608805	-13.608805
tries	3	-3.713517	11.140550	-11.140550
then	2	-5.403499	10.806999	-10.806999
funny	5	-1.934586	9.672929	-9.672929
have	4	-2.268136	9.072545	-9.072545
is	11	0.822754	9.050289	9.050289
good	3	2.979705	8.939114	8.939114
even	4	-2.223664	8.894656	-8.894656

In [206]:
```
example_analysis(174)
```

```
Review file name: cv118_28980.txt
True sentiment:  1 .  Predicted sentiment:  -1.0
plot : a human space astronaut accidentally falls upon a planet ruled by
apes .
he is taken prisoner along with some other humans , and tries his best to
 escape his simian captors .
oh yeah , and did i mention that the apes can talk ?
kewl !
critique : despite this film not looking like a " tim burton movie " or i
ts lead human character showing any signs of being a human being himself
, i did find myself generally entertained throughout most of this film an
d do recommend it as fun , summer fare .
although the thing that sets this movie apart from all others is definite
ly its incredible ape make-up jobs and characterizations .
i mean , i was completely convinced that every single one of these apes w
as for real !
i also have to " give it up " to all of the actors inside the costumes ,
not only for waking up at three in the morning every day to get make-up s
lobbered all over them for 2-4 hours , but for coming through their costu
med exteriors and selling us their characters on the screen as well .
i especially enjoyed bonham carter's performance , who i believed to be t
he most developed character in the entire film ( human or ape ) and tim r
oth's bad guy , who chewed up scenery and oozed evil in every scene in wh
ich he took part .
plot-wise , i liked the beginning of the film , the whole build-up once w
ahlberg got into the village , the escape and most of the fight scenes ,
but once things got out into the deserted area ( and the final battle seq
uence ) , i kinda lost a bit of my interest .
i was much more intrigued by the interactions between the characters and
the whole idea of " human beings as animals " then i was about the end ba
ttle sequence , which just seemed a bit too typical for this kind of film
 .
i also hated the " kid " character who just got on my nerves and didn't r
eally care for the way that the battle sequence was resolved ( i won't ru
in it here but c'mon guys . . . you
coulda thought of something a little more plausible than that , no ? ) .
but the thing that held this movie back more than anything else was its l
ead character , his cold demeanor and the zero chemistry that he shared w
ith either one of his inter-species love interests .
this dude didn't seem to give a rat's ass about anyone but himself and i
ultimately didn't really care about what happened to him either .
now i'm not sure if this was the fault of the actor or the script , but t
he film suffered because of it .
visually , the movie was also nice to look at ( especially the apes ! ) ,
 but i definitely expected greater coolness from tim burton .
the " surprise ending " that highlighted the original planet of the apes
film back in 1968 , felt a little tacked on here , and despite being inte
resting , didn't really make all that much sense to me ( unless you look
at it from a sequel point of view ) .
so overall , i loved the whole " feel " of the planet , the apes that rul
ed it , the manner in which they treated the humans , screamed , shouted
and reverted the entire evolution chain , but didn't appreciate the film'
s lack of human character development ( why was kris kristofferson even i
n this movie ? ) and the fact that its lead male and female humans were i
```

```
ust plain boring .
but i would still recommend this film as a fun , summer movie with laughs
 , creepiness and a really cool premise .
ps : for anyone who has seen the original , i cannot believe that they di
dn't at least bring back the scene in which the lead human character ( in
 this case , mark wahlberg ) goes nuts at the world in which he's suddenl
y been thrust ( " this is a mad house ! " -
type of thing ) .
the guy in this film barely seems to be " put off " by the fact that he's
 surrounded by apes who can speak english ! !
where's joblo coming from ?
the arrival ( 8/10 ) - battlefield earth ( 7/10 ) - godzilla ( 4/10 ) - i
nstinct ( 6/10 ) - mission to mars ( 3/10 ) - pitch black ( 7/10 ) - ( 19
68 ) planet of the apes ( 9/10 ) - sleepy hollow ( 8/10 )
```

Out[206]:

	x	w	abs_xw	xw
and	18	2.512735	45.229231	45.229231
this	12	-1.689983	20.279794	-20.279794
apes	8	-1.534326	12.274611	-12.274611
the	50	0.222368	11.118381	11.118381
but	9	-1.223012	11.007107	-11.007107
at	5	-2.001298	10.006492	-10.006492
!	6	-1.667746	10.006478	-10.006478
also	3	3.335492	10.006475	10.006475
most	3	3.024182	9.072546	9.072546
bad	1	-8.694517	8.694517	-8.694517

In [207]:
```
example_analysis(15)
```

Review file name: cv165_2389.txt
True sentiment: -1 . Predicted sentiment: 1.0
what are we going to do with jim carrey ?
viewers of television's " in living color " know this one-man cartoon fro
m such characters as fire marshall bill .
viewers also know that " in living color " is a skit-show and that a litt
le of jim carrey goes a long way .
unfortunately , this fact was forgotten by the makers of the carrey comed
y ace ventura : pet detective .
three writers , including carrey , worked on the slapstick story , which
sends a self-styled " pet detective " on the trail of a stolen dolphin .
the missing mammal belongs to the miami dolphins , who need their mascot
for the upcoming superbowl .
for plot porpoises , this story works as well as any three stooges short
.
carrey gets to do his " official " schtick as he snoops around greater mi
ami .

he leers and sneers , craning his neck to funny effect .
he even does his captain kirk impersonation .
again .
all of this is pretty harmless stuff up until the point that you realize
that the writers have absolutely no intention of focusing on anyone * oth
er * than carrey .
 (suggested alternate title--jim carrey : will do anything for a laugh .
)
export it to france and you may have a hit .
as it stands , ace ventura isn't even good kid's stuff .
the profanity count , alone , is too high .
which is ironic , since children are , probably , carrey's best audience
.
the film doesn't even have the goofball charm of chris elliott's recent n
o-brainer cabin boy .
sure , carrey has his moments .
but what can you say about a film whose high-points include watching carr
ey slink around to the theme from " mission impossible ? "
ace ventura has one glaring incongruity .
amid the butt-jokes and double-takes , the script takes great pains to se
t-up an elaborate and rather funny " crying game " gag .
and , for * this * intended audience , that takes (ahem) cojones .

Out[207]:

	x	w	abs_xw	xw
and	6	2.512735	15.076410	15.076410
you	3	3.157599	9.472797	9.472797
this	5	-1.689983	8.449914	-8.449914
do	3	-2.557211	7.671634	-7.671634
he	3	2.379319	7.137957	7.137957
have	3	-2.268136	6.804409	-6.804409
even	3	-2.223664	6.670992	-6.670992
to	7	-0.756045	5.292313	-5.292313
unfortunately	1	-4.892058	4.892058	-4.892058
as	5	0.956174	4.780872	4.780872

6 Features

For a problem like this, the features you use are far more important than the learning model you choose. Whenever you enter a new problem domain, one of your first orders of business is to beg, borrow, or steal the best features you can find. This means looking at any relevant published work and seeing what they've used. Maybe it means asking a colleague what features they use. But eventually you'll need to engineer new features that help in your particular situation. To get ideas for this dataset, you might check the discussion board on this Kaggle competition, which is using a very similar dataset https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews (https://www.kaggle.com/c/sentiment-analysis-on-movie-reviews). There are also a very large number of academic research papers on sentiment analysis that you can look at for ideas.

1. Based on your error analysis, or on some idea you have, find a new feature (or group of features) that improve your test performance. Describe the features and what kind of improvement they give. At this point, it's important to consider the standard errors ($\sqrt{p(1-p)/n}$) on your performance estimates, to know whether the improvement is statistically significant.

> From the beginning, I removed singleton words - words which only appear once in the corpus - to improve efficiency. Removing stop words improved model performance by ~3 percentage points. Adding in a review's word count did not improve model performance.

In [258]:
```
def evaluate_model(X,y,remove_stops=False, Lambda=0.1):
    w = initialize_w(1)
    w_opt = pegasos(X,y,Lambda,w,30, delta=0.1, plot=False, remove_stops
=remove_stops)
    pct_err = get_pct_error(X_test,y_test,w_opt)
    std_err = np.sqrt(pct_err*(1-pct_err)/len(X_test))

    print "Percent Error: %f +/- %f" %(pct_err,std_err)
```

In [229]:
```
X,y,ids=lists2bags(train)
X_test,y_test,ids_test=lists2bags(test)

print "Model with stopwords included"
evaluate_model(X,y)

print ""
print "Model with stopwords removed"
evaluate_model(X,y,remove_stops=True)


#Add a word count feature
for review in X:
    total_words=0
    for key in review:
        total_words+=review[key]
    review['word_count']=total_words

print ""
print "With word counts"
evaluate_model(X,y)
```

```
Model with stopwords included
Time taken: 8 seconds
Percent Error: 0.178000 +/- 0.017106

Model with stopwords removed
Time taken: 3 seconds
Percent Error: 0.152000 +/- 0.016056

With word counts
Time taken: 22 seconds
Percent Error: 0.504000 +/- 0.022360
```

2. [Optional] Try to get the best performance possible by generating lots of new features, changing the pre-processing, or any other method you want, so long as you are using the same core SVM model. Describe what you tried, and how much improvement each thing brought to the model. To get you thinking on features, here are some basic ideas of varying quality: 1) how many words are in the review? 2) How many “negative” words are there? (You’d have to construct or find a list of negative words.) 3) Word n-gram features: Instead of single-word features, you can make every pair of consecutive words a feature. 4) Character n-gram features: Ignore word boundaries and make every sequence of n characters into a feature (this will be a lot). 5) Adding an extra feature whenever a word is preceded by “not”. For example “not amazing” becomes its own feature. 6) Do we really need to eliminate those funny characters in the data loading phase? Might there be useful signal there? 7) Use tf-idf instead of raw word counts. The tf-idf is calculated as

$$\text{tfidf}(f_i) = \frac{FF_i}{log(DF_i)}$$

where FF_i is the feature frequency of feature f_i and DF_i is the number of document containing f_i. In this way we increase the weight of rare words. Sometimes this scheme helps, sometimes it makes things worse. You could try using both! [Extra credit points will be awarded in proportion to how much improvement you achieve.]

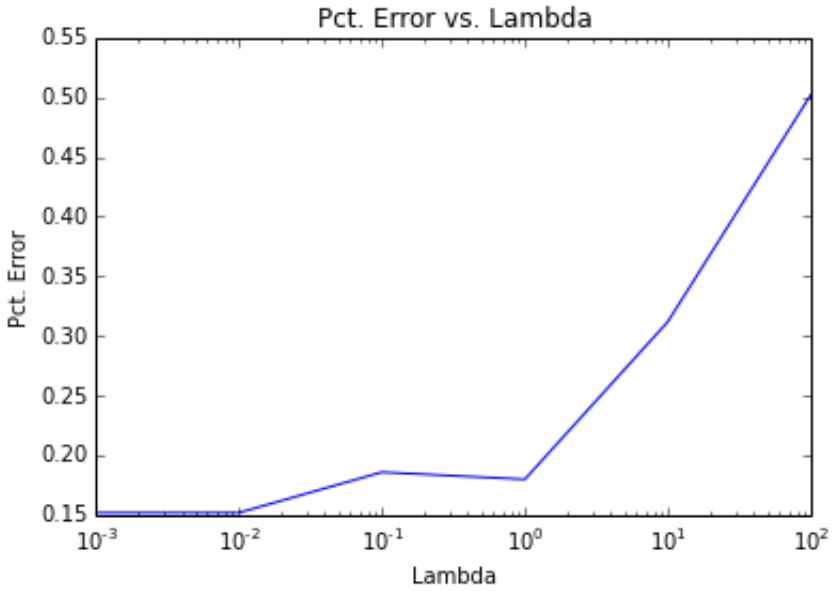
I was able to get an accuracy improvement of 2 percentage points by adding bigrams to the model, in addition to the unigrams. The best model, which removed stop words and singleton words, and used bigrams, had percent error of %13.2. I suspect bigrams addressed issues like the above "no problems", so that positive words that get negated become negative words. Additional code for this is below.

In [250]:
```
def append_bigrams(List):
    prev_word = 'start_of_review'
    bigrams=[]
    for word in List:
        bigram = prev_word + ' ' + str(word)
        bigrams.append(bigram)
        prev_word=str(word)

    return bigrams + List
```

In [251]:
```
def lists2bigrams(lists):
    '''Converts list of words into sparse bag-of-words representation'''
    bags = []
    labels = []
    ids = []
    for review in lists:
        #append bigrams
        review = append_bigrams(review)
        #don't include the review's label or id
        bag = collections.Counter(review[:-2])
        bags.append(bag)
        labels.append(review[-2])
        ids.append(review[-1])
    return bags,labels,ids
```

In [256]:
```
w_opt,Lambdas,loss_hist=Lambda_search(pegasos,False)
plt.plot(Lambdas,loss_hist)
plt.xlabel('Lambda')
plt.ylabel('Pct. Error')
plt.xscale('log')
plt.title('Pct. Error vs. Lambda')
plt.show()
```

```
Starting Loop
Lambda 100
Time taken: 14 seconds
Lambda 10
Time taken: 15 seconds
Lambda 1
Time taken: 15 seconds
Lambda 0.1
Time taken: 34 seconds
Lambda 0.01
Time taken: 67 seconds
Lambda 0.001
Time taken: 85 seconds
Best Lambda: 0.001
Square Loss on Test Data: 0.152
```



In [259]:
```
X,y,ids=lists2bigrams(train)
X_test,y_test,ids_test=lists2bigrams(test)
evaluate_model(X,y,remove_stops=True,Lambda=0.001)
```

```
Time taken: 57 seconds
Percent Error: 0.132000 +/- 0.015138
```

7 Feedback (not graded)

1. Approximately how long did it take to complete this assignment?

   > 22 hours

2. Did you find the Python programming challenging (in particular, converting your code to use sparse representations)?

   > The python was fairly straightforward. There are always snags and frustrations - for example, I had a tougher time than I expected discretizing the scores into bins in part 5.

3. Any other feedback?

   > For the first time I felt comfortable with this assignment, and confident that I had enough time to thoroughly understand what I was doing, rather than just following the steps.