Given dataset $\{(x_1, y_1) \dots (x_n, y_n)\}$ where $x_i \in R^d$ and $y_i \in \{-1, 1\}$. Parameterized by $\theta \in R^d$. Predicts +1 at a point x if $\theta \cdot x > 0$ and -1 otherwise.

$\|\theta^*\| = 1$ (1) where $\|x\| = \sqrt{\sum_i (x^{(i)})^2}$ and some scalar $\gamma > 0$ such that for all $i = 1, \dots, n$, $y_i(x_i \cdot \theta^*) \geq \gamma$ (2). This assumes that for all $i = 1, \dots n$ x_i is bounded by a constant $R > 0$ i $\|x_i\| \leq R$ (3). Find that $m_{update} \leq R^2/\gamma^2$ (4)

Proof: (a) $\theta^1 = 0$ (5) Starts with a parameter that classifies for x_i that classifies all points at -1 and has zero slope. This puts all points to one side of the linear classifier allowing it to sweep over all points.

(b) when Alg. 1 meets a point x_k that is misclassified we update $\theta^{t+1} = \theta^t + y_k x_k$. Taking the dot product of both sides by θ^* we get $\theta^{t+1} \cdot \theta^* = (\theta^t + y_k x_k) \cdot \theta^*$ (6) $= \theta^t \cdot \theta^* + y_k x_k \cdot \theta^*$ (7)

(c) Taking eq(2) we see that $\theta^{t+1} \cdot \theta^* = \theta^t \cdot \theta^* + y_k(x_k \cdot \theta^*) \geq \theta^t \cdot \theta^* + \gamma$. by the transitive property we get $\theta^{t+1} \cdot \theta^* \geq \theta^t \cdot \theta^* + \gamma$ (8)

(d) Tracking back from iteration t to Iteration 1 of Alg(1) we see from eq(8) that $\theta^{t+1} \cdot \theta^* \geq \theta^t \cdot \theta^* + \gamma \geq \theta^{t-1} \cdot \theta^* + 2\gamma \dots$ $\theta^1 \cdot \theta^* + t\gamma$. Combined with eq(5) and the transitive property we see that $\theta^{t+1} \cdot \theta^* \geq t\gamma$ (9)

(e) Applying Cauchy-Schwarz's inequality to eq(9) we see that $\|\theta^{t+1}\|\|\theta^*\| \geq \theta^{t+1} \cdot \theta^* \geq t\gamma$. applying eq(1) and the transitive property we see that $\|\theta^{t+1}\| \geq t\gamma$ (10)

(f) Following we have $\|\theta^{t+1}\|^2 = \|\theta^t + y_k x_k\|^2$ (11) $= \|\theta^t\|^2 + y_k^2\|x_k\|^2 + 2y_k x_k \cdot \theta^t$ (12)

Knowing that each iteration is a misclassification we observe that $x_k \theta^t$ and y_k have opposite parity and that y_k^2 is always equal to +1. Therefore

$$\Rightarrow \|\theta^t\|^2 = \|\theta^t\|^2 + \|x_k\|^2 - 2|y_k x_k \cdot \theta^t| \leq \|\theta^t\|^2 + R^2 - 2|y_k x_k \theta^t|$$

given the last negative term and the transitive property, we observe that $\|\theta^{t+1}\|^2 \leq \|\theta^1\|^2 + R^2$ (13). From the above we get $\|\theta^{t+1}\|^2 \leq tR^2$ (14)

(g) Together (10) and (14) yields

$$t^2\gamma^2 \leq \|\theta^{t+1}\|^2 \leq tR^2$$

implying that $t^2\gamma^2 \leq tR^2$ by induction. Therefore . . .

$$t \leq \frac{R^2}{\gamma^2}$$

QED

Programming Assignment 4: Sentiment analysis with SVM

In this programming assignment, we will revisit the problem of sensiment analysis, but using a different approach. Recall that the task is to predict the *sentiment* (positive or negative) of a single sentence taken from a review of a movie, restaurant, or product. The data set consists of 3000 labeled sentences, which we divide into a training set of size 2500 and a test set of size 500. Previously we found a logistic regression classifier. Today we will use a support vector machine.

Make sure the notebook is in the same folder that contains `full_set.txt` .

1. Load and preprocess data

In [1]:
```
%matplotlib inline
import string
import numpy as np
import matplotlib
import matplotlib.pyplot as plt
matplotlib.rc('xtick', labelsize=14)
matplotlib.rc('ytick', labelsize=14)
```

The data set consists of 3000 sentences, each labeled '1' (if it came from a positive review) or '0' (if it came from a negative review). To be consistent with our notation from lecture, we will change the negative review label to '-1'.

In [2]:
```
## Read in the data set.
with open("full_set.txt") as f:
    content = f.readlines()

## Remove leading and trailing white space
content = [x.strip() for x in content]

## Separate the sentences from the labels
sentences = [x.split("\t")[0] for x in content]
labels = [x.split("\t")[1] for x in content]

## Transform the labels from '0 v.s. 1' to '-1 v.s. 1'
y = np.array(labels, dtype='int8')
y = 2*y - 1
```

Preprocessing the text data

To transform this prediction problem into one amenable to linear classification, we will first need to preprocess the text data. We will do four transformations:

1. Remove punctuation and numbers.
2. Transform all words to lower-case.
3. Remove *stop words*.
4. Convert the sentences into vectors, using a bag-of-words representation.

We begin with first two steps.

In [3]:
```
## full_remove takes a string x and a list of characters removal_list
## returns x with all the characters in removal_list replaced by ' '
def full_remove(x, removal_list):
    for w in removal_list:
        x = x.replace(w, ' ')
    return x

## Remove digits
digits = [str(x) for x in range(10)]
digit_less = [full_remove(x, digits) for x in sentences]

## Remove punctuation
punc_less = [full_remove(x, list(string.punctuation)) for x in digit_less]

## Make everything lower-case
sents_lower = [x.lower() for x in punc_less]
```

Stop words

Stop words are words that are filtered out because they are believed to contain no useful information for the task at hand. These usually include articles such as 'a' and 'the', pronouns such as 'i' and 'they', and prepositions such 'to' and 'from'. We have put together a very small list of stop words, but these are by no means comprehensive. Feel free to use something different; for instance, larger lists can easily be found on the web.

In [4]:
```
## Define our stop words
stop_set = set(['the', 'a', 'an', 'i', 'he', 'she', 'they', 'to', 'of', 'it',
'from'])

## Remove stop words
sents_split = [x.split() for x in sents_lower]
sents_processed = [" ".join(list(filter(lambda a: a not in stop_set, x))) for
x in sents_split]
```

Bag of words

In order to use linear classifiers on our data set, we need to transform our textual data into numeric data. The classical way to do this is known as the *bag of words* representation. In this representation, each word is thought of as corresponding to a number in `{1, 2, ..., d}` where `d` is the size of our vocabulary. And each sentence is represented as a d-dimensional vector x, where x_i is the number of times that word i occurs in the sentence.

To do this transformation, we will make use of the `CountVectorizer` class in `scikit-learn` (Note that this is the only time you can call an external function from scikit-learn). We will cap the number of features at 4500, meaning a word will make it into our vocabulary only if it is one of the 4500 most common words in the corpus. This is often a useful step as it can weed out spelling mistakes and words which occur too infrequently to be useful.

Once we get the bag-of-words representation, append a '1' to the beginning of each vector to allow our linear classifier to learn a bias term.

In [42]:
```
from sklearn.feature_extraction.text import CountVectorizer

## Transform to bag of words representation.
vectorizer = CountVectorizer(analyzer = "word", tokenizer = None, preprocessor
= None, stop_words = None, max_features = 4500)
data_features = vectorizer.fit_transform(sents_processed)
data_mat = data_features.toarray()
print ('The original size: ',data_features.shape)
```

```
The original size:  (3000, 4500)
```

Training / test split

Finally, we split the data into a training set of 2500 sentences and a test set of 500 sentences (of which 250 are positive and 250 negative).

In [6]:
```
## Split the data into testing and training sets
np.random.seed(0)
test_inds = np.append(np.random.choice((np.where(y==-1))[0], 250, replace=Fals
e), np.random.choice((np.where(y==1))[0], 250, replace=False))
train_inds = list(set(range(len(labels))) - set(test_inds))

train_data = data_mat[train_inds,]
train_labels = y[train_inds]

test_data = data_mat[test_inds,]
test_labels = y[test_inds]

print("train data: ", train_data.shape)
print("test data: ", test_data.shape)
```

```
train data:  (2500, 4500)
test data:  (500, 4500)
```

2. Solving for soft-margin SVM

Recall that support vector machine (SVM) finds a linear decision boundary with the largest margin for a binary classification problem. Suppose we have a training dataset $\{(x_1, y_1), \ldots, (x_n, y_n)\}$ where $x_i \in \mathbb{R}^d$ are feature vectors and $y_i \in \{-1, +1\}$ are labels. The linear classifier is parametrized by $\theta \in \mathbb{R}^d$ and $\theta_0 \in \mathbb{R}$, and predicts +1 at a point x if $\theta \cdot x + \theta_0 > 0$ and -1 otherwise.

It turns out that the soft-margin SVM optimization is equivalent to the following unconstrained optimization:

$$\min_{\theta \in \mathbb{R}^d, \theta \in \mathbb{R}} \|\theta\|^2 + C \sum_{i=1}^{n} \ell_{\text{hinge}}(y_i(\theta \cdot x_i + \theta_0))$$

where $\ell_{\text{hinge}}(t) = \max(0, 1 - t)$ is called the ``hinge loss," which takes value $1 - t$ if $t < 1$ and 0 otherwise. For example, $\ell_{\text{hinge}}(-1) = 2$, and $\ell_{\text{hinge}}(2) = 0$.

It turns out that for gradient-based optimization, hinge loss may be difficult to deal with because it is not differentiable at point $t = 1$. One solution is to use the ``smoothed version" of hinge loss:

$$\ell_{\text{smooth-hinge}}(t) = \begin{cases} \frac{1}{2} - t & \text{if } t \leq 0, \\ \frac{1}{2}(1 - t)^2 & \text{if } 0 < t < 1, \\ 0 & \text{if } 1 \leq t \end{cases}$$

Thus, in the rest of the problem, we will consider the following optimization:

$$\min_{\theta \in \mathbb{R}^d, \theta \in \mathbb{R}} \|\theta\|^2 + C \sum_{i=1}^{n} \ell_{\text{smooth-hinge}}(y_i(\theta \cdot x_i + \theta_0))$$

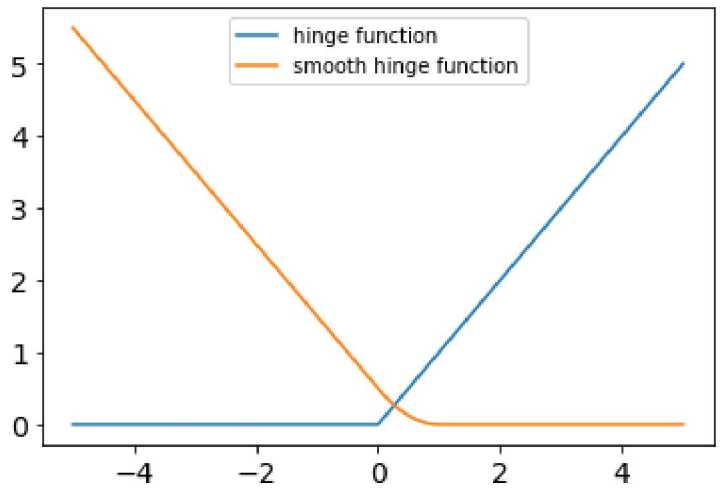
Task P2: Implement the hinge loss function and the smooth hinge loss function. Plot the function $\ell_{\text{hinge}}(t)$ and $\ell_{\text{smooth-hinge}}(t)$ for $t \in [-5, 5]$.

In [7]:
```
### STUDENT: Start of code ###
def hinge(t):
    return np.maximum(0, t)

def smooth_hinge(t):
    if t <= 0:
        return 0.5 - t
    elif 0 < t < 1:
        return 0.5 * (1. - t) ** 2.
    else:
        return 0.

def d_smooth_hinge(t):
    if t <= 0:
        return -1.
    elif 0 < t < 1:
        return t - 1.
    else:
        return 0.

x_list = np.linspace(-5, 5, 500)
hinge_list, smooth_list = [], []
for x in x_list:
    hinge_list.append(hinge(x))
    smooth_list.append(smooth_hinge(x))
plt.plot(x_list, hinge_list, label='hinge function')
plt.plot(x_list, smooth_list, label='smooth hinge function')
plt.legend(loc='best')
plt.show()
### End of code ###
```



Task P5: Let $f(\theta, \theta_0) = \|\theta\|^2 + C \sum_{i=1}^{n} \ell_{\text{smooth-hinge}}(y_i(\theta \cdot x_i + \theta_0))$ be the objective function of the optimization problem we want to solve. Implement the function that obtains the partial derivative $\frac{\partial}{\partial \theta} f(\theta, \theta_0)$ and $\frac{\partial}{\partial \theta_0} f(\theta, \theta_0)$. Also, print out the output of the code that calculates the derivatives at $\theta = 1$ and $\theta_0 = 1$ with $C = 1$.

Hint: you need to calculate the partial derivative of the smoothed hinge loss for each data point separately, and add them together to obtain the result.

In [14]:
```
def weight_derivative(theta, theta0, C, feature_matrix, labels):
    # Input:
    # theta: weight vector theta, a numpy vector of dimension d
    # theta0: intercept theta0, a numpy vector of dimension 1
    # C: constant C
    # feature_matrix: numpy array of size n by d, where n is the number of dat
a points, and d is the feature dimension
    # labels: true labels y, a numpy vector of dimension d, each with value -1
or +1
    # Output:
    # Derivative of the cost function with respect to the weight theta, grad_t
heta
    # Derivative of the cost function with respect to the weight theta0, grad_
theta0

    ## STUDENT: Start of code ###
    grad_theta, grad_theta0 = 0, 0
    for i in range(len(feature_matrix)):
        inside = labels[i] * (np.dot(feature_matrix[i], theta) + theta0)
        common_terms = d_smooth_hinge(inside) * labels[i]
        grad_theta += common_terms * feature_matrix[i]
        grad_theta0 += common_terms
    grad_theta *= C
    grad_theta0 *= C

    grad_theta += 2 * theta

    return grad_theta, grad_theta0
    # End of code ###
```

In [15]:
```
# STUDENT: PRINT THE OUTPUT AND COPY IT TO THE SOLUTION FILE
theta = np.ones(data_mat.shape[1]) # a weight of all 1s
theta0 = np.ones(1) # a number 1
C = 1
grad_theta, grad_theta0 = weight_derivative(theta, theta0, C, train_data, trai
n_labels)

print (grad_theta[:10])
print (grad_theta0)
```

```
[ 2.  3.  3.  4.  3.  3. 38.  5.  2.  3.]
1250.0
```

Task P6: For sentiment analysis data, choose a value for the trade-off parameter C. Report the training error at convergence and the testing error.

Note: you can just use the same gradient descent algorithm that we wrote in assignment 2, or use the adam_optimizer provided below.

Here is an article (https://towardsdatascience.com/adam-latest-trends-in-deep-learning-optimization-6be9a291375c#:~:text=Adam%20%5B1%5D%20is%20an%20adaptive,for%20training%20deep%20neural%20ne on the Adam optimizer.

In [16]:
```
def objective(feature_matrix, labels, theta,theta0, C):
    score = (feature_matrix.dot(theta)+theta0)*labels
    return np.sum(theta**2)+C*np.sum([smooth_hinge(t) for t in score])
```

In [17]:
```
def adam_optimizer(feature_matrix, labels, initial_theta,initial_theta0, C, st
ep_size=0.01, tolerence=0.01, b1=0.9, b2=0.999, eps=10**-8):
    # Gradient descent algorithm for logistic regression problem

    # Input:
    # feature_matrix: numpy array of size n by d, where n is the number of dat
a points, and d is the feature dimension
    # labels: true labels y, a numpy vector of dimension d
    # initial_theta: initial theta to start with, a numpy vector of dimension
 d
    # initial_theta0: initial theta0 to start with, a numpy vector of dimensio
n 1
    # step_size: step size of update
    # tolerance: tolerace epsilon for stopping condition
    # Parameters by Adam optimizer
    # Output:
    # Weights obtained after convergence

    converged = False
    m = np.zeros(len(initial_theta))
    v = np.zeros(len(initial_theta))
    m0 = np.zeros(1)
    v0 = np.zeros(1)
    theta = np.array(initial_theta) # current iterate
    theta0 = np.array(initial_theta0) # current iterate
    i = 0
    while not converged:
        # impelementation of what the gradient descent algorithm does in every
iteration
        # Refer back to the update rule listed above: update the weight
        i += 1
        grad_theta, grad_theta0  = weight_derivative(theta, theta0, C,feature_
matrix, labels)

        m = (1 - b1) * grad_theta      + b1 * m  # First  moment estimate.
        v = (1 - b2) * (grad_theta**2) + b2 * v  # Second moment estimate.
        mhat = m / (1 - b1**(i + 1))    # Bias correction.
        vhat = v / (1 - b2**(i + 1))
        theta = theta - step_size*mhat/(np.sqrt(vhat) + eps)

        m0 = (1 - b1) * grad_theta0      + b1 * m0  # First  moment estimate.
        v0 = (1 - b2) * (grad_theta0**2) + b2 * v0  # Second moment estimate.
        mhat0 = m0 / (1 - b1**(i + 1))    # Bias correction.
        vhat0 = v0 / (1 - b2**(i + 1))
        theta0 = theta0 - step_size*mhat0/(np.sqrt(vhat0) + eps)

        # Compute the gradient magnitude:

        gradient_magnitude = np.sqrt(np.sum(grad_theta**2))

        # Check the stopping condition to decide whether you want to stop the
 iterations

        if gradient_magnitude < tolerance:
            converged = True
```

```
        preds_train = model_predict(train_data,theta,theta0)

        ## Compute errors
        errs_train = np.sum((preds_train > 0.0) != (train_labels > 0.0))

        print ("Iteration: ",i,"objective: ",objective(feature_matrix, labels,
theta,theta0, C),"tr err: ",float(errs_train)/len(train_labels),"gradient_magn
itude: ", gradient_magnitude) # for us to check about convergence

    return(theta, theta0)

### End of code ###
```

In [18]:

```
def model_predict(feature_matrix,theta,theta0):
# Prediction made by SVM

    # Input:
    # feature_matrix: numpy array of size n by d+1, where n is the number of d
ata points, and d+1 is the feature dimension
    #                 note we have included the dummy feature as the first col
umn of the feature_matrix
    # theta: weight theta, a numpy vector of dimension d
    # theta0: weight theta0, a numpy vector of dimension 1
    # Output:
    # labels: predicted labels, a numpy vector of dimension n

    h =  feature_matrix.dot(theta)+theta0
    y_h = (h >= 0)*2-1

    return y_h
```

In [19]:
```
# Initialize the weights, step size and tolerance
# Start of code
theta = np.ones(4500) # a weight of all 1s
theta0 = np.ones(1) # a number 1

initial_theta =  np.ones(data_mat.shape[1]) ## STUDENT: initialize theta
initial_theta0 = np.ones(1)   ## STUDENT: initialize theta0
C =  1 ## STUDENT: choose the C
step_size =  0.01 ## STUDENT: choose the step_size
tolerance =  0.07 ## STUDENT: choose the tolerance

# end of code

theta, theta0 = adam_optimizer(train_data, train_labels, initial_theta, initia
l_theta0, C, step_size, tolerance)
print(theta)
print(theta0)
```

```
Iteration:  1 objective:  18062.780117925668 tr err:  0.5 gradient_magnitude:
881.9659857386791
Iteration:  2 objective:  17873.964729424068 tr err:  0.5 gradient_magnitude:
881.6146137011254
Iteration:  3 objective:  17674.63403909325 tr err:  0.5 gradient_magnitude:
881.2105241348464
Iteration:  4 objective:  17469.781250121036 tr err:  0.5 gradient_magnitude:
880.7838262127175
Iteration:  5 objective:  17261.772586408806 tr err:  0.5 gradient_magnitude:
880.3451985835258
Iteration:  6 objective:  17051.921865916556 tr err:  0.5 gradient_magnitude:
879.8997037290503
Iteration:  7 objective:  16841.0370627796 tr err:  0.5 gradient_magnitude:
879.4501545665358
Iteration:  8 objective:  16629.6521405062 tr err:  0.5 gradient_magnitude:
878.9982828953936
Iteration:  9 objective:  16418.139378458018 tr err:  0.5 gradient_magnitude:
878.5452349470058
Iteration:  10 objective:  16206.769269370263 tr err:  0.5 gradient_magnitud
e:  878.0918114530543
Iteration:  11 objective:  15995.744870214543 tr err:  0.5 gradient_magnitud
e:  877.6385956422866
Iteration:  12 objective:  15785.222665064954 tr err:  0.5 gradient_magnitud
e:  877.1860266343668
Iteration:  13 objective:  15575.325839267109 tr err:  0.5 gradient_magnitud
e:  876.7344440076195
Iteration:  14 objective:  15366.153057459334 tr err:  0.5 gradient_magnitud
e:  876.2841161501474
Iteration:  15 objective:  15157.784460881541 tr err:  0.5 gradient_magnitud
e:  875.8352590040168
Iteration:  16 objective:  14950.285881451862 tr err:  0.5 gradient_magnitud
e:  875.3880488686324
Iteration:  17 objective:  14743.71187626295 tr err:  0.5 gradient_magnitude:
874.942631394905
Iteration:  18 objective:  14538.107960565561 tr err:  0.5 gradient_magnitud
e:  874.499128060109
Iteration:  19 objective:  14333.512283181 tr err:  0.5 gradient_magnitude:
874.0576409312374
Iteration:  20 objective:  14129.956905900766 tr err:  0.5 gradient_magnitud
e:  873.6182562380314
Iteration:  21 objective:  13927.468796362016 tr err:  0.5 gradient_magnitud
e:  873.1810471008223
Iteration:  22 objective:  13726.070610132407 tr err:  0.5 gradient_magnitud
e:  872.7460756470628
Iteration:  23 objective:  13525.781315354885 tr err:  0.5 gradient_magnitud
e:  872.3133946782973
Iteration:  24 objective:  13326.616698157102 tr err:  0.5 gradient_magnitud
e:  871.8830490015049
Iteration:  25 objective:  13128.589776591998 tr err:  0.5 gradient_magnitud
e:  871.4550765063808
Iteration:  26 objective:  12931.711143562843 tr err:  0.5 gradient_magnitud
e:  871.0295090478298
Iteration:  27 objective:  12735.989253984644 tr err:  0.5 gradient_magnitud
e:  870.6063731773171
Iteration:  28 objective:  12541.430667683777 tr err:  0.5 gradient_magnitud
e:  870.185690755612
Iteration:  29 objective:  12348.040256800223 tr err:  0.5 gradient_magnitud
```

```
Iteration:  1198 objective:  291.89334720177476 tr err:  0.012 gradient_magni
tude:  0.08601069021293996
Iteration:  1199 objective:  291.8933368855992 tr err:  0.012 gradient_magnit
ude:  0.08543053966155899
Iteration:  1200 objective:  291.8933266855712 tr err:  0.012 gradient_magnit
ude:  0.08485419786226302
Iteration:  1201 objective:  291.89331660044684 tr err:  0.012 gradient_magni
tude:  0.08428164103079436
Iteration:  1202 objective:  291.89330662899465 tr err:  0.012 gradient_magni
tude:  0.08371284557039023
Iteration:  1203 objective:  291.8932967699958 tr err:  0.012 gradient_magnit
ude:  0.08314778800291042
Iteration:  1204 objective:  291.893287022244 tr err:  0.012 gradient_magnitu
de:  0.08258644490740594
Iteration:  1205 objective:  291.8932773845453 tr err:  0.012 gradient_magnit
ude:  0.08202879287023859
Iteration:  1206 objective:  291.89326785571774 tr err:  0.012 gradient_magni
tude:  0.08147480845108832
Iteration:  1207 objective:  291.89325843459187 tr err:  0.012 gradient_magni
tude:  0.08092446816567334
Iteration:  1208 objective:  291.89324912000995 tr err:  0.012 gradient_magni
tude:  0.08037774848624368
Iteration:  1209 objective:  291.8932399108261 tr err:  0.012 gradient_magnit
ude:  0.07983462585764485
Iteration:  1210 objective:  291.8932308059063 tr err:  0.012 gradient_magnit
ude:  0.0792950767263906
Iteration:  1211 objective:  291.89322180412813 tr err:  0.012 gradient_magni
tude:  0.07875907757902628
Iteration:  1212 objective:  291.8932129043807 tr err:  0.012 gradient_magnit
ude:  0.07822660498558068
Iteration:  1213 objective:  291.8932041055644 tr err:  0.012 gradient_magnit
ude:  0.07769763564347067
Iteration:  1214 objective:  291.89319540659113 tr err:  0.012 gradient_magni
tude:  0.07717214641841987
Iteration:  1215 objective:  291.8931868063838 tr err:  0.012 gradient_magnit
ude:  0.07665011437812415
Iteration:  1216 objective:  291.89317830387654 tr err:  0.012 gradient_magni
tude:  0.07613151681721875
Iteration:  1217 objective:  291.8931698980142 tr err:  0.012 gradient_magnit
ude:  0.07561633127113086
Iteration:  1218 objective:  291.89316158775273 tr err:  0.012 gradient_magni
tude:  0.07510453551941536
Iteration:  1219 objective:  291.8931533720587 tr err:  0.012 gradient_magnit
ude:  0.07459610757843944
Iteration:  1220 objective:  291.89314524990925 tr err:  0.012 gradient_magni
tude:  0.07409102568560408
Iteration:  1221 objective:  291.8931372202924 tr err:  0.012 gradient_magnit
ude:  0.07358926827704833
Iteration:  1222 objective:  291.8931292822061 tr err:  0.012 gradient_magnit
ude:  0.07309081396134945
Iteration:  1223 objective:  291.89312143465895 tr err:  0.012 gradient_magni
tude:  0.07259564149215121
Iteration:  1224 objective:  291.8931136766698 tr err:  0.012 gradient_magnit
ude:  0.07210372974244933
Iteration:  1225 objective:  291.89310600726753 tr err:  0.012 gradient_magni
tude:  0.0716150576821131
Iteration:  1226 objective:  291.89309842549085 tr err:  0.012 gradient_magni
```

```
tude:  0.07112960436119578
Iteration:  1227 objective:  291.89309093038884 tr err:  0.012 gradient_magni
tude:  0.07064734889928774
Iteration:  1228 objective:  291.89308352102 tr err:  0.012 gradient_magnitud
e:  0.07016827048193616
Iteration:  1229 objective:  291.8930761964527 tr err:  0.012 gradient_magnit
ude:  0.06969234836346697
[-2.05176335e-24  2.93130007e-22 -2.26276431e-01 ... -3.75922144e-28
 -4.64270739e-02 -5.19597434e-02]
[-0.09953247]
```

In [22]:
```
# STUDENT: copy the output of this section to the solution file

## Get predictions on training and test data
preds_train = model_predict(train_data,theta,theta0)
preds_test = model_predict(test_data,theta,theta0)

## Compute errors
errs_train = np.sum((preds_train > 0.0) != (train_labels > 0.0))
errs_test = np.sum((preds_test > 0.0) != (test_labels > 0.0))

print("Training error: ", float(errs_train)/len(train_labels))
print("Test error: ", float(errs_test)/len(test_labels))
```

```
Training error:  0.012
Test error:  0.136
```

Task P7: List 4 example sentences that are correctly classified by SVM, and 4 example sentences that are incorrectly classified by SVM. Explain what you have found.

In [55]:
```
# STUDENT: your code here
incorrect_sent = []
correct_sent = []
correct, incorrect = 0, 0
i = np.random.randint(500)
while (correct < 4) or (incorrect < 10):
    if ((preds_test[i] > 0.) != (test_labels[i] > 0.)): #error
        if incorrect < 10:
            incorrect_sent.append((test_inds[i], test_data[i], preds_test[i],
test_labels[i]))
            incorrect += 1
    else: #correct prediction
        if correct < 4:
            correct_sent.append((test_inds[i], test_data[i], preds_test[i], te
st_labels[i]))
            correct += 1
    i = np.random.randint(500)



print("_____________________________________")
print("correctly classified sentences:")
print("-----------------------------------")
for sent in correct_sent:
    print(sentences[sent[0]])
    print("labeled as:", sent[2], "actually:", sent[3])
    print(" ")

print('_____________________________________')
print("incorrectly classified sentences:")
print("-----------------------------------")
for sent in incorrect_sent:
    print(sentences[sent[0]])
    print("labeled as:", sent[2], "actually:", sent[3])
    print(" ")
```

```
----------------------------------
correctly classified sentences:
----------------------------------
Food was great and so was the serivce!
labeled as: 1 actually: 1

This is a GREAT place to eat!
labeled as: 1 actually: 1

Great food and service, huge portions and they give a military discount.
labeled as: 1 actually: 1

Damian is so talented and versatile in so many ways of writing and portraying
different Characters on screen.
labeled as: 1 actually: 1


----------------------------------
incorrectly classified sentences:
----------------------------------
This was like the final blow!
labeled as: 1 actually: -1

This is one of Peter Watkins most accessible films.
labeled as: -1 actually: 1

i'm glad i found this product on amazon it is hard to find, it wasn't high pr
iced.
labeled as: -1 actually: 1

Very Dissapointing Performance.
labeled as: 1 actually: -1

Still, it was the SETS that got a big "10" on my "oy-vey" scale.
labeled as: -1 actually: 1

Julian Fellowes has triumphed again.
labeled as: -1 actually: 1

WORTHWHILE.
labeled as: -1 actually: 1

Overall, I don't think that I would take my parents to this place again becau
se they made most of the similar complaints that I silently felt too.
labeled as: 1 actually: -1

Steer clear of this product and go with the genuine Palm replacementr pens, w
hich come in a three-pack.
labeled as: 1 actually: -1

A standout scene.
labeled as: -1 actually: 1
```

Observations

Of the incorrectly labelled sentences, there were a few observations to be made. Most of the false positives contain some simple words that have greater weight than the neagtive obscure words. Words like "much", "more", and "good" showed up in the false positives where a customer described the experience they expected. This is because only a bag of words was considered apposed to looking at sentence structure and context. Similarly the false negatives contained words like "almost" or "hard". This echoes the lack of context. Another factor for some of the mislabeled sentences were the size of those sentences. few words led to very binary results that could increase the error of the sentence.