1 Calculating subgradient(2.1)

Given,

$$f(x) = \max_{i=1,\ldots,m} f_i(x).$$

Suppose $f_k(x) = f(x)$, and choose $g \in \partial f_k(x)$.
To show: $g \in \partial f(x)$
We know, for all z,

$$f_k(z) \geq f_k(x) + g^T(z - x).$$

But note: $f(z) \geq f_k(z)$,
So,

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

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

2 Calculating subgradient(2.2)

To find subgradient of

$$J(w) = \max\left\{0, 1 - yw^T x\right\}.$$

1. When $J(w) = 0$, It is obvious that $0 \in \partial J(w)$
2. When $J(w) = 1 - yw^T x$,

$$J'(w) = \frac{d}{dw}(1 - yw^T x) = -yx$$

$$\Rightarrow -yx \in \partial J(w)$$

3 Perceptron(3.1)

If $\left\{x \mid w^T x = 0\right\}$ is a separating hyperplane, and we know, $\hat{y}_i = w^T x_i$, we know,

$$y_i w^T x_i > 0 \Rightarrow y_i \hat{y}_i > 0 \Rightarrow -y_i \hat{y}_i < 0$$

Then, for all i

$$l(\hat{y}_i, yi) = \max\{0, -\hat{y}_i y_i\} = 0$$

So, the average Perceptron loss is 0 since all losses are 0.

4 Perceptron(3.2)

Assuming the step-size of 1, the SSGD implements the following

if $(y_i x_i^T w^{(k)} \leq 0)$: $w^{(k+1)} = w^{(k)} - \nabla_w l$
else: $w^{(k+1)} = w^{(k)} - \nabla_w l$

But $\nabla_w l = -y_i x_i$ or 0, so, we get:
if $(y_i x_i^T w^{(k)} \leq 0)$: $w^{(k+1)} = w^{(k)} + y_i x_i$
else: $w^{(k+1)} = w^{(k)}$

Which is exactly Perceptron.

5 Perceptron(3.3)

The perceptron algorithm updates by two methods: wither scales w, or scales w and adds a vector x, evident by:

if $(y_i x_i^T w^{(k)} \leq 0)$: $w^{(k+1)} = w^{(k)} - y_i x_i$
else: $w^{(k+1)} = w^{(k)}$

So, $w = \sum_{i=1}^{n} \alpha_i x_i$ indeed.

The characterization of the support vector is at the surface, that they have non-zero coefficient when updating w, which is equivalent to saying that they were never miscategorized and add little value to classification.

6 The Data(4)

```
    #loading the shuffled data
    with open('data.pickle', 'rb') as f:
        review = pickle.load(f)


    #Splitting into training and test sets
    train, test = split(review)

def split(review):
    train = []
    test = []
    for i in range(len(review)):
        if i%4 == 0:
            test.append(review[i])
        else:
            train.append(review[i])
    return train, test
```

7 Sparse representation (5)

```
    #Splitting x and y values and getting ready for training
    x_train = []
    x_test = []
    y_train = []
    y_test = []

    for i in train:
        y_train.append(i.pop())
        x_train.append(bag_of_words(i))

    for i in test:
        y_test.append(i.pop())
        x_test.append(bag_of_words(i))


def bag_of_words(list):

    cnt = Counter()
    for word in list:
        cnt[word] += 1

    return cnt
```

8 SVM visa Pegasos (6.1)

The derivative is undefined at:

$$1 - y_i w^T x_i = 0$$
$$\Rightarrow y_i w^T x_i = 1$$

When the derivative is defined, it is either λw, when

$$\max(1 - y_i w^T x_i) = 0$$

and in other case, the derivative will be

$$\lambda w - y_i x_i$$

9 SVM visa Pegasos (6.2)

Using the given facts in the question and the fact that gradients are also subgradients, and from the gradient values from the last answer, we get:

$$g = \begin{cases} \lambda w - y_i x_i & \text{for } y_i w^T x_i < 1 \\ \lambda w & \text{for } y_i w^T x_i \geq 1. \end{cases}$$

10 SVM visa Pegasos (6.3)

In SGD form where size rule is $\eta_t = 1/\,(\lambda t)$:

If $y_j w_t^T x_j < 1$: $w_{t+1} = w_t + g$

Else: $w_{t+1} = w_t + g$

But substituting the value of gradient from the last question, we get:

If $y_j w_t^T x_j < 1$: $w_{t+1} = (1 - \eta_t \lambda) w_t + \eta_t y_j x_j$

Else: $w_{t+1} = (1 - \eta_t \lambda) w_t$

which is exactly pegasos.

11 SVM visa Pegasos (6.4)

```
def pegasos(x, y, l):
w = dict()
t = 2
temp_loss = 0
flag = True
while flag:
    for j in range(len(x)):
        t = t + 1
        n = 1/(l*t)
        if y[j]*(dotProduct(w, x[j])) < 1:
            cnt = cnt +1
            temp = x[j].copy()
            increment(temp, (n*y[j]-1), temp)
            increment(w,-n*l,w)
            increment(w,1,temp)
        else:
            increment(w,-n*l,w)
        loss_real = loss(x,y,l,w)
        if abs(temp_loss - loss_real) < 10**-2:
        flag = False
        temp_loss = loss_real
return w
```

12 SVM visa Pegasos (6.5)

We know:

$$\begin{aligned} s_{t+1} &= (1 - \eta_t \lambda)\, s_t \\ W_{t+1} &= W_t + \frac{1}{s_{t+1}} \eta_t y_j x_j. \end{aligned}$$

substituting by $w = W_{t+1} s_{t+1}$ on both sides:

$$\frac{w}{s_{t+1}} = \frac{w(1 - \eta_t) + \eta_t y_j x_j}{s_{t+1}}$$

$$\Rightarrow w = (1 - \eta_t) + \eta_t y_j x_j$$

So, the two updates are equivalent.
The code looks as follows:

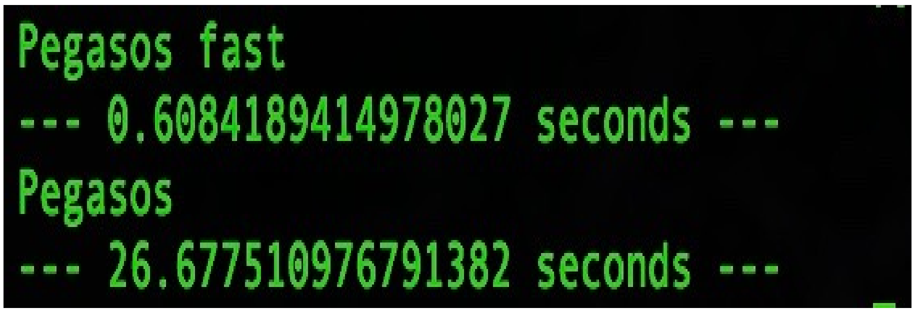
```
def pegasos_fast(x, y, l):

w = dict()
temp_w = dict()
t = 2
s = 1
temp_loss = 0
flag = True
while flag:
    for j in range(len(x)):
        t = t + 1
        n = 1/(l*t)
        s = (1-n*l)*s
        if y[j]*(dotProduct(w, x[j])) < s:
            cnt = cnt +1
            temp = x[j].copy()
            increment(temp, (n*y[j]-1), temp)
            increment(w,(1/s), temp)
    temp_w = w.copy()
    increment(temp_w, s-1, temp_w)
    loss_real = loss(x,y,l,temp_w)
    if abs(temp_loss - loss_real) < 10**-2:
        flag = False
    temp_loss = loss_real

increment(w, s-1, w)
return w
```

13 SVM visa Pegasos (6.6)

The implementations are essentially the same, and have the same value for objective function and the error function. On running the two algorithms for two epochs, we get the the following difference, which is a pretty large difference.



time

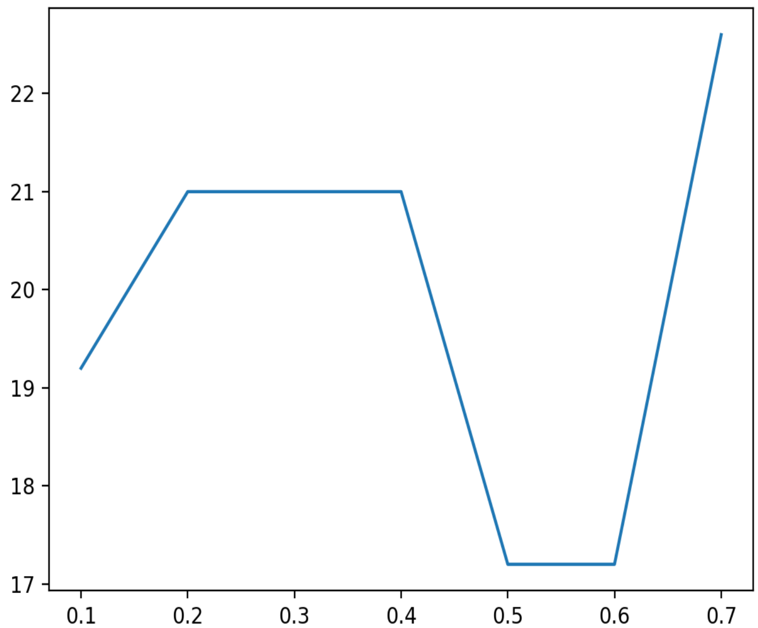
14 SVM visa Pegasos (6.7)

The error reporting function follows:

```
def per_loss(x,y,w):
cnt = 0
total = len(y)
for i in range(total):
    if np.sign(dotProduct(w, x[i])) != np.sign(y[i]):
        cnt = cnt + 1
error = (cnt/total)*100.0
return error
```

15 SVM visa Pegasos (6.8)

After trying out different values for the lambda, validation loss seems to be minimum at $\lambda = 0.5 or 0.6$. Searching deeper didn't seem to be useful as the minimum stabilized at 17.2 percentage.



λ

16 Error Analysis (7)

The first example of that the model got wrong was due to the negative prediction to a positively rated movie. I have included the first page of the table sorted by absolute value of the weight as listing the whole table wold take 15 pages and not particularly useful. What we see immediately is that the top of the table is filled with words like "the", "he', "and", and so on that lack any positive or negative connotation or signalling power, but our model has learned to have some value for them due to their abundance in the training data. While not visible in the table, their seems to be words like "hope" and "serious" with negative weight in the model, but their signalling power is entirely dependent in the context.

(1).png

feature	abs(w*x)	w*x	x	w
the	0.312819373	0.312819373	32	0.009775605
to	0.211508554	-0.211508554	17	-0.01244168
is	0.151521884	0.151521884	22	0.006887358
he	0.150855365	0.150855365	7	0.021550766
of	0.138635859	0.138635859	12	0.011552988
on	0.129971118	-0.129971118	5	-0.025994224
arnold	0.106642968	-0.106642968	6	-0.017773828
and	0.10131082	0.10131082	19	0.005332148
about	0.099977783	-0.099977783	6	-0.016662964
or	0.085980893	-0.085980893	3	-0.028660298
a	0.083981337	-0.083981337	18	-0.00466563
if	0.065318818	-0.065318818	2	-0.032659409
so	0.063319262	-0.063319262	3	-0.021106421
we	0.06309709	0.06309709	4	0.015774272
film	0.062208398	-0.062208398	8	-0.00777605
never	0.053988003	-0.053988003	3	-0.017996001
any	0.049322373	-0.049322373	2	-0.024661186
then	0.046656299	-0.046656299	2	-0.023328149
4	0.046211953	-0.046211953	4	-0.011552988
obvious	0.045767607	-0.045767607	2	-0.022883804
know	0.043101533	-0.043101533	2	-0.021550766
world	0.039324595	0.039324595	1	0.039324595
could	0.039102422	-0.039102422	2	-0.019551211
war	0.037325039	0.037325039	4	0.00933126
his	0.035547656	-0.035547656	8	-0.004443457
very	0.033992446	0.033992446	1	0.033992446
script	0.032215063	-0.032215063	1	-0.032215063
great	0.031326372	0.031326372	1	0.031326372
as	0.031104199	0.031104199	2	0.0155521
more	0.030659853	0.030659853	2	0.015329927
see	0.027993779	0.027993779	1	0.027993779
play	0.027993779	-0.027993779	2	-0.01399689
best	0.027993779	0.027993779	1	0.027993779
at	0.027549433	-0.027549433	2	-0.013774717
all	0.027105088	-0.027105088	1	-0.027105088
looks	0.026216396	-0.026216396	1	-0.026216396
especially	0.025772051	0.025772051	1	0.025772051
what	0.025327705	-0.025327705	3	-0.008442568
no	0.025105532	-0.025105532	1	-0.025105532
films	0.024883359	0.024883359	2	0.01244168
make	0.024439014	-0.024439014	1	-0.024439014

In example 2 too, you see exactly the same issue as the first example. When the text lacks strongly suggestive words, filler words such as "to", and "on" get more decision power on predicting the sentiment, and this is exactly the case with the second example. This example predicted positive sentiment for a negatively reviewed movie.

(-1).png

Feature	abs(w*x)	w*x	x	w
the	0.146634081	0.146634081	15	0.009775605
to	0.136858476	-0.136858476	11	-0.01244168
of	0.092423906	0.092423906	8	0.011552988
world	0.078649189	0.078649189	2	0.039324595
on	0.077982671	-0.077982671	3	-0.025994224
is	0.068873584	0.068873584	10	0.006887358
a	0.055987558	-0.055987558	12	-0.00466563
seen	0.04287936	0.04287936	1	0.04287936
some	0.042212842	-0.042212842	2	-0.021106421
john	0.035769829	-0.035769829	7	-0.005109976
have	0.033992446	-0.033992446	1	-0.033992446
reason	0.032881582	-0.032881582	2	-0.016440791
script	0.032215063	-0.032215063	1	-0.032215063
more	0.030659853	0.030659853	2	0.015329927
see	0.027993779	0.027993779	1	0.027993779
though	0.025549878	0.025549878	1	0.025549878
result	0.024439014	-0.024439014	2	-0.012219507
but	0.023994668	0.023994668	3	0.007998223
life	0.023550322	0.023550322	1	0.023550322
4	0.023105976	-0.023105976	2	-0.011552988
good	0.022439458	0.022439458	1	0.022439458
he	0.021550766	0.021550766	1	0.021550766
and	0.021328594	0.021328594	4	0.005332148
be	0.021328594	-0.021328594	3	-0.007109531
poor	0.020884248	-0.020884248	1	-0.020884248
things	0.017551655	0.017551655	1	0.017551655
as	0.0155521	0.0155521	1	0.0155521
middle	0.014885581	-0.014885581	1	-0.014885581
movie	0.014663408	-0.014663408	1	-0.014663408
hope	0.014219062	-0.014219062	2	-0.007109531
at	0.013774717	-0.013774717	1	-0.013774717
does	0.013330371	0.013330371	2	0.006665186
an	0.01244168	-0.01244168	1	-0.01244168
this	0.011997334	-0.011997334	2	-0.005998667
family	0.01088647	0.01088647	1	0.01088647
serious	0.010664297	-0.010664297	1	-0.010664297
like	0.010664297	-0.010664297	2	-0.005332148
one	0.010219951	0.010219951	2	0.005109976
off	0.010219951	-0.010219951	1	-0.010219951
sent	0.009775605	-0.009775605	2	-0.004887803
must	0.009775605	0.009775605	2	0.004887803

One definitive improvement strategy would be to get rid of the filler words mentioned above. That would result in the weight being distributed more among words with actual predictive value. Furthermore, taking words separately rids them of context, which can be important to understand the sentiment. So, somehow (perhaps through hard coding famous phrases) incorporating phrases and idioms migh be helpful in increasing the predicting power of the model.