Parthvi Shah

Pss434

Question 2.1:

In [1]:
```
import os
import numpy as np
import pickle
import random
from collections import Counter


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', 'willis', 'is', 'no
t', 'called', 'on',
    ...'to', 'carry', 'the', 'whole', 'movie', "he's", 'much', 'better', 'and', 'so', 'is',
'the', 'movie']
    '''
    f = open(file)
    lines = f.read().split(' ')
    symbols = '${}()[].,:;+-*/&|<>=~" '
    words = map(lambda Element: Element.translate(str.maketrans("", "", symbols)).strip(),
lines)
    words = filter(None, words)
    return list(words)
```

In [2]:
```
    pos_path = "/Users/parthvi/Downloads/data/pos"
    neg_path = "/Users/parthvi/Downloads/data/neg"

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

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

In [3]:
```
training_set = review[:1500]
validation_set = review[1500:]
```

Question 2.2:

In [4]:
```
x_train, y_train = training_set[0:-1], [item[-1] for item in training_set]
x_val, y_val = validation_set[:-1],[item[-1] for item in validation_set]

X_train, X_val = [], []
for i in range(len(x_train)):
    c_train = Counter()
    for word in x_train[i]:
        c_train[word] += 1
    X_train.append(c_train)

for i in range(len(x_val)):
    c_val = Counter()
    for word in x_val[i]:
        c_val[word] += 1
    X_val.append(c_val)
```

In [5]:
```
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
```

Question 3.4:

In [6]:
```
def pegasos(X_train, y_train, max_epoch, lam, watch_list=None, grad_checking=False):

    #Initialization
    weight = {}
    epoch = 0
    t = 0.

    #Loop
    # Use the util.increment and util.dotProduct functions in update
    while epoch < max_epoch:
        epoch+=1
        for j in range(len(X_train)):
            t+=1.
            eta=1.0/(lam*t)
            increment(weight, -eta*lam, weight)
            if y_train[j]*dotProduct(weight,X_train[j])<1:
                increment(weight,eta*y_train[j],X_train[j])
    return weight
```

Question 3.5:

In [7]:
```
def pegasos_fast(X_train,y_train, max_epoch, lam, watch_list=None, grad_checking=False, tfi
df= False):

    #Initialization
    weight = {}
    epoch = 0
    t = 1.
    #review_number = len(review_list)
    s = 1.
    W = {}

    #Loop

    while epoch < max_epoch:
        for j in range(len(X_train)):
            t+=1
            eta = 1/(lam*t)
            s += -eta*lam*s
            if y_train[j]*dotProduct(weight,X_train[j])*s<1:
                increment(weight,(1/s)*eta*y_train[j],X_train[j])
        epoch+=1
    W.update((X_train,s*y_train) for X_train,y_train in weight.copy().items())
    return W
```

Question 3.6:

In [8]:
```
import time
start_time=time.time()
w_=pegasos(X_train, y_train, 6, 1)
end_time=time.time()
print('Time for pegasos without S')
print(end_time-start_time)
```

```
Time for pegasos without S
141.25147700309753
```

In [19]:
```
import time
start_time=time.time()
w_fast=pegasos_fast(X_train, y_train, 6, 1)
end_time=time.time()
print('Time for pegasos with S')
print(end_time-start_time)
```

```
Time for pegasos with S
2.194347858428955
```

In [20]:
```
import time
start_time=time.time()
w_fast=pegasos_fast(X_train, y_train, 500, 1)
end_time=time.time()
print('Time for pegasos with S')
print(end_time-start_time)
```

```
Time for pegasos with S
175.7924451828003
```

With 6 epochs, it takes 2.19 second, with 500 it takes 175. I have kept 500 as a criteria for convergence.

Just printing the first 10 items so that it doesn't cover the whole page.

In [22]:
```
for i,j in enumerate(w_fast.items()):
    if(i<10):
        print(j)
#print(w_fast)
```

```
('capsule', -0.000652434086145336)
('in', 0.008022671083826042)
('2176', -0.0006671105175310139)
('on', -0.02297528622376838)
('the', 0.006909930740586103)
('planet', 0.000579051929216929)
('mars', -0.013362223666146355)
('police', -0.008676439391008463)
('taking', 0.00665642874392453)
('into', -0.00934488412957455)
```

In [23]:
```
for i,j in enumerate(w_.items()):
    if(i<10):
        print(j)
```

```
('capsule', -0.0027796308650211363)
('in', 0.005892817433844786)
('2176', -0.0006671114076050719)
('on', -0.03924838781409812)
('the', 0.00833889259506353)
('planet', 0.0007782966422059157)
('mars', -0.005892817433844753)
('police', -0.010117856348676901)
('taking', 0.0027796308650211276)
('into', -0.012341561040693803)
```

In [24]:
```
error_weights = sum(w_[value]*w_fast.get(value, 0) for value in w_)
```

In [25]:
```
error_weights
```

Out[25]:
```
0.3850636707530602
```

Since the error_weights is small, we can say that they are almost the same.

Question 3.7:

In [26]:
```
def error_percent(X_train,y_train, weight, tfidf=False):

    error = 0
    for i in range(len(X_train)):
        if dotProduct(weight, X_train[i])<0:
            prediction = -1
        else:
            prediction = 1
        if y_train[i] !=prediction:
            error+=1
    return error/len(X_train)
```

In [27]:
```
loss = error_percent(X_train,y_train, w_)
```

In [28]:
```
loss
accuracy = (1.0-loss)
print("Loss:",loss," Accuracy:",accuracy)
```

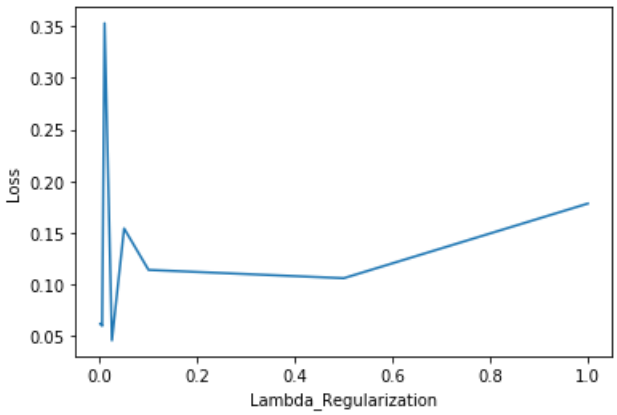
```
Loss: 0.09206137424949967  Accuracy: 0.9079386257505003
```

Question 3.8:

In [17]:
```
lambda_regs=[1e-3,2.5e-3,5e-3,1e-2,2.5e-2,5e-2, 1e-1, 5e-1,1]
w_l=[]
for lambda_reg in lambda_regs:
    w=pegasos_fast(X_train,y_train,lam=lambda_reg, max_epoch=20)
    w_l.append(w)
losses=[]
for w in w_l:
    loss=error_percent( X_val, y_val, w)
    losses.append(loss)
```

In [29]:
```
import matplotlib.pyplot as plt
plt.plot(lambda_regs,losses)
plt.xlabel('Lambda_Regularization')
plt.ylabel('Loss')
```

Out[29]:
```
Text(0, 0.5, 'Loss')
```



I believe the regularization parameter at around 0.1, gives the least loss.

In [ ]:
```
```

1. Perceptron.

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

1. $f_1 \dots f_m \rightarrow$ convex functions

$f(x) = \max f_i(x)$

$f_k(x) = f(x) \quad g \in \partial f_k(x)$

Show: $g \in \partial f(x)$

→ Since $g \in \partial f_k(x)$

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

$$\therefore f(z) \geq f_k(z) \geq f_k(x) + g^T(z-x)$$
$$= f(x) + g^T(z-x) \quad \forall z$$

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

2. $J(w) = \max\{0, 1 - yw^Tx\}$

By 2.1, we have

$$\partial J(w) = \begin{cases} 0 & 1 - yw^Tx \\ -yx & 0 \end{cases}$$

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

3 If $x | w^Tx = 0$; avg loss = 0

∴ any seperating hyperplane is an empirical risk minimizer for perceptron loss

→ ∵ $\{x \mid w^T x = 0\}$ is a seperating hyperplane

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

∴ By def^n, perceptron loss = 0 for all D_i's. ∴ Average perceptron loss = 0

$$\because \text{ERM} = \frac{1}{n} \sum_{i=1}^{n} \ell_{hf}(x)$$

$$= 0$$

∴ Any seperating hyperplane of D is an ERM ✓

4 $\quad x \longrightarrow w^T x$

→ If we use $\eta = 1$

$$w^{(k+1)} = w^k + y_i x_i \quad \text{if } y_i w^T x_i < 0$$

$$w^{(k+1)} = w^k \quad \text{if } y_i w^T x_i > 0$$

This exactly represents the perceptron algorithm.

5 Show that w is a linear comb. of i/p points

→ For perceptron algo,

we update w by either adding $y_i x_i$ or keeping it unchanged. ∴ w is a linear combination of x_i

If a pt is a support vector, $y_i w^T x_i > 0 \Rightarrow x_i$ is misclassified.

If a pt is misclassified in perceptron, it is a support vector, else it is not.

Q.3 SVM with via pegasos.

1. $J_i(w) = \frac{1}{2}\|w\|^2 + \max\{0, 1 - y_i w^T x_i\}$

→ $\frac{1}{2}\|w\|^2$ is convex. & the hinge loss is not differentiable w.r.t w, we calculate sub-gradient.

$\nabla J(w) = [\lambda w ; 0]$ if $\max(0, 1 - y_i w^T x_i) = 0$

$= [\lambda w ; 0 - N y_i x_i]$ o/w

$N \rightarrow$ no. in training set

Q.2

from 1;

$$\nabla J = \begin{cases} \lambda w - y_i x_i & 1 - y_i w x_i > 0 \\ \lambda w & \text{otherwise} \end{cases}$$

$$\therefore w = w - \frac{1}{\lambda t}(\lambda w - y_i x_i) \quad 1 - y_i w x_i > 0$$
$$w = w - \frac{1}{\lambda t}\lambda w \quad \text{otherwise}$$

Q.3 If $\eta_t = \frac{1}{\lambda t}$

$$\therefore w = w - \frac{1}{\lambda t}(\lambda w - y_i x_i)$$
$$= 1 - \eta_t(\lambda w - y_i x_i)$$
$$= (1 - \eta_{t-1})w + \eta_t y_i x_i$$

if $y_i w^T x_i < 1$

else

$$w = w - \frac{1}{\lambda t}\lambda w$$
$$= w - w/t$$
$$= (1 - \eta_{t-1})w$$

∴ It is the same as the pseudocode

Q.5 Kernels

1. $K(x,z)$ is a kernel by constructing $\phi(z), \phi(x)$ such that

$$K(x,z) = \phi(x) \cdot \phi(z)$$

Given the vocab V,

$\phi(x) \rightarrow x$

If a word appears in V (wk);
$\phi(x)_k = 1$ else $\phi(x)_k = 0$

Then, the unique words is given by $\phi(x) * \phi(z)$

Q.2

(a) $f(x)\, f(z)\, k_1(x,z)$

$$\rightarrow \quad k_1(x,z) = \frac{1}{\|x\|} \frac{1}{\|z\|} K(x,z)$$

$$= \frac{x}{\|x\|} \cdot \frac{z}{\|z\|}$$

(b) $k_2(x,z) = k_1(x,z) + k_2(x,z)$

$$\Rightarrow \quad \frac{x}{\|x\|} \frac{z}{\|z\|} + \frac{x}{\|x\|} \frac{z}{\|z\|}$$

$\rightarrow 1 + \frac{x}{\|x\|} \frac{z}{\|z\|}$ //

(c) Product.

$$k(x, z) = k_1(x, z)\, k_2(x, z)$$

$\rightarrow k_3(x, z) = k_2(x, z)\, k_2(x, z)$

$= 1 + \left(\frac{x}{\|x\|} \cdot \frac{z}{\|z\|}\right)^2$

Show: $\left(1 + \left(\frac{x}{\|x\|}\right)^T \left(\frac{z}{\|z\|}\right)\right)^3$ is a kernel

$\rightarrow$ $k_3(x, z) \cdot k_2(x, z)$ //

8.6 Kernel Pegasos.

1.

margin $= y_i w^T x_i$

$w = \sum_{i=1}^{n} \alpha_i^t x_i$

$\therefore y_j w^T x_j = y_j \sum_{i=1}^{n} \alpha_i^t x_i x_j$

$\therefore y_j w^T x_j = y_j K_j \alpha^t$

$K(x_i, x_j) = \langle x_i, x_j \rangle$

2 $w^t = \sum_{i=1} \alpha_i x_i$

If $t+1$ doesn't violate margin,

$$w^{t+1} = (1 - \eta^{(t)} \lambda) w^{(t)}$$

$$x^T \alpha^{(t+1)} = (1 - \eta^t \lambda) x^T \alpha^{(t)}$$

$$\boxed{\alpha^{t+1} = (1 - \eta^t \lambda) \alpha^t}$$

2.3 If $t+1$ violates margin,

$$w^{(t+1)} = (1 - \eta^t \lambda) w^t + \eta_t y_j x_j$$

$$x^T \alpha^{(t+1)} = (1 - \eta^t \lambda) x^T \alpha^t + \eta_t y_j x^T (0,0,\ldots,1,0)$$

$$\alpha^{t+1} = (1 - \eta^t \lambda) \alpha^t + \eta_t y_j (0,0,\ldots,1,0)$$

∴ Pseudo code

: $\alpha^1 = (0, 0 \ldots 0)$

$t = 0$

while:

$t = t+1$

$\eta^t = \frac{1}{t}$

for random (j)

if $y_j K_j^T K < 1$

$\alpha^{t+1} = (1 - \eta^t \lambda) \alpha^t$

$\alpha^{t+1}[j] = \alpha^{t+1}[j] + \eta^t y_j$

else

$\alpha^{t+1} = (1 - \eta^t \lambda) \alpha^t$

return α^t