Homework 4: Support Vector Machines and Kernels

Student Name: Kuan-Lin Liu

NetID: kll482

1. Perceptron

1.1

Given $g \in \partial f_k(x)$, we have $f_k(z) \geq f_k(x) + g^T(z - x), \forall z$

By definition,

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

Therefore,

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

which shows,

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

1.2

$$f(x) = \begin{cases} 0 & 1 - y_i {x_i}^T w^{(k)} \leq 0 \\ -y_i x_i & else \end{cases}$$

1.3

Given $\{x | w^T x = 0\}$ is a separating hyperplane, we know

$$y_i \hat{y_i} = y_i w^T x_i > 0, \forall i \in \{1, \ldots, n\}$$

Then,

$$R_{emp} = \frac{1}{n} \sum_{i=1}^{n} max\{0, -y_i \hat{y_i}\} = \frac{1}{n} * n * 0 = 0$$

Therefore, we know a separating hyperplane of $\mathcal{D}$ is an empirical risk minimizer for perceptron loss.

1.4

By the answer of 1.2, the subgradient of perceptron loss is

$$\partial\ell(\hat{y}, y) = \begin{cases} -y_i x_i & y_i {x_i}^T w^{(k)} \leq 0 \\ 0 & else \end{cases}$$

Since we have the step size 1 for SSGD, we would update w by either $w^{(k+1)} = w^{(k)} + y_i x_i$ if $y_i {x_i}^T w^{(k)} \leq 0$ or $w^{(k+1)} = w^{(k)}$ if $y_i {x_i}^T w^{(k)} > 0$. This is the same step as the one in Perceptron Algorithm.

1.5

Assume α is an indicator vector,

$$\alpha_i = \begin{cases} 1 & y_i {x_i}^T w^{(k)} \leq 0 \\ 0 & else \end{cases}$$

From the pseudocode, we know

$$w^{(k)} = (\alpha_i y_i x_i)^{(k-1)} + (\alpha_i y_i x_i)^{(k-2)} + (\alpha_i y_i x_i)^{(k-3)} + \ldots + (\alpha_i y_i x_i)^{(1)} + w^{(0)}$$

$\because \alpha \in \{0, 1\}; y \in \{1, -1\}$

$\therefore$ w is a linear combination of the inputs, x.

2. Sparse Representations

2.1

In [1]:

```
import os
import numpy as np
import pickle
import random
from load import read_data, folder_list
```

In [2]:

```
def shuffle_data_with_seed():
    pos_path = "data/pos"
    neg_path = "data/neg"

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

    review = pos_review + neg_review
    random.seed(123)
    random.shuffle(review)

    return review
```

In [3]:

```
shuffled = shuffle_data_with_seed() # read and shuffle
train = shuffled[:1500]
val = shuffled[1500:]
```

2.2

In [4]:

```
from collections import Counter
```

In [5]:

```
def SparseBOW(word_list):
    return Counter(word_list)
```

In [6]:

```
X_train = [row[:-1] for row in train]
y_train = [row[-1] for row in train]
X_val = [row[:-1] for row in val]
y_val = [row[-1] for row in val]
```

In [7]:

```
X_train_dict = [SparseBOW(row) for row in X_train]
X_val_dict = [SparseBOW(row) for row in X_val]
```

3. SVM with via Pegasos

3.1

We know hinge loss and $||w||^2$ is convex, so

$$\partial J_i(w) = \begin{cases} \lambda w - y_i x_i & 1 - y_i w^T {x_i}^T > 0 \\ \lambda w & else \end{cases}$$

3.2

3.3

In the stochastic subgradient descent, the weight is updated in the following rule:

$$\begin{cases} w^{(k+1)} = w^{(k)} - \eta_t(\lambda w^{(k)} - y_i x_i) = w^{(k)}(1 - \eta_t \lambda) + \eta_t y_i x_i & 1 - y_i w^T {x_i}^T > 0 \\ w^{(k+1)} = w^{(k)} - \eta_t \lambda w^{(k)} = w^{(k)}(1 - \eta_t \lambda) & else \end{cases}$$

This is the same as the update rule in the pseudocode.

3.4

In [8]:

```
from collections import defaultdict
from tqdm import tqdm_notebook
from util import dotProduct, increment
```

In [9]:

```
def svm_pegasus(X, y, lambda_reg=0.1, max_epoch=6):
    # X is a list of dict.
    t = 1
    w = defaultdict(float)
    for _ in tqdm_notebook(range(max_epoch)):
        for ind in range(len(X)):
            # x_i is a dictonary, y_i is a scaler
            t += 1
            eta = 1/(t*lambda_reg)
            #x_i, y_i = X[ind], y[ind]

            if y[ind]*dotProduct(w, X[ind]) < 1:
                increment(w, eta*y[ind], X[ind])

            # both conditions need to update the following line
            increment(w, -eta*lambda_reg, w)

    return w
```

3.5

$$w_{t+1} = s_{t+1} W_{t+1}$$

$$= (1 - \eta_t \lambda) s_t [W_t + \frac{1}{(1-\eta_t \lambda) s_t} \eta_t y_j x_j]$$

$$= (1 - \eta_t \lambda) s_t W_t + \eta_t y_j x_j$$

$$= (1 - \eta_t \lambda) w_t + \eta_t y_j x_j$$

In [10]:

```
def svm_pegasus_faster(X, y, lambda_reg=0.1, max_epoch=6):
    # X is a list of dict.
    t = 1
    s_t = 1
    W = defaultdict(float)
    for _ in tqdm_notebook(range(max_epoch)):
        for ind in range(len(X)):
            # x_i is a dictonary, y_i is a scaler
            t += 1
            eta = 1/(t*lambda_reg)

            if s_t*y[ind]*dotProduct(W, X[ind]) < 1:
                increment(W, eta*y[ind]/s_t, X[ind])

            s_t = (1-eta*lambda_reg)*s_t # when t=1, (1-eta*lambda_reg)=0. then,
s_t=0.

    W.update((k, v*s_t) for k, v in W.items())

    return W
```

3.6

In [11]:

```
from time import time
```

In [12]:

```
train_pegasus = svm_pegasus(X_train_dict, y_train)
```

In [13]:

```
train_pegasus_faster = svm_pegasus_faster(X_train_dict, y_train)
```

In [14]:

```
print(list(train_pegasus_faster.items())[:10])
print(list(train_pegasus.items())[:10])
```

```
[('plot', -0.22125439373241765), ('a', -0.036574315879128974), ('downandout', 0.0016676047332736604), ('girl', -0.046663
437773035776), ('moves', 0.012310508550439319), ('in', -0.016934847640683034), ('with', 0.024205969772543825), ('some',
-0.01205747816693706), ('overthetop', 0.009471747364872597), ('models', 0.00610359157062008)]
[('plot', -0.2212543937324168), ('a', -0.03657431587912881), ('downandout', 0.0016676047332736539), ('girl', -0.04666343
777303576), ('moves', 0.012310508550439364), ('in', -0.016934847640682787), ('with', 0.024205969772543866), ('some', -0.
012057478166937106), ('overthetop', 0.009471747364872619), ('models', 0.006103591570620092)]
```

3.7

In [15]:

```
def lossfunction(w, X, y):
    match = []
    for ind in range(len(X)):
        w_x = dotProduct(w, X[ind])
        if w_x < 0:
            y_hat = -1
        else:
            y_hat = 1
        match.append(y_hat == y[ind])
    return sum(match)/len(X)
```

3.8

In [16]:

```
def reg_tuning(reg_list, train_X, train_y, val_X, val_y):
    loss = {}
    for reg in reg_list:
        w = svm_pegasus_faster(train_X, train_y, reg, 50)
        loss[reg] = lossfunction(w, val_X, val_y)
    return loss
```

In [17]:

```
# reg = np.logspace(-3, 3, 7)
# {0.001: 0.856, 0.01: 0.848, 0.1: 0.798, 1.0: 0.816, 10.0: 0.712, 100.0: 0.508,
1000.0: 0.508}# reg = np.logspace(1, 5, 5)
# reg = np.logspace(-6, -1, 6)
# {1e-06: 0.836, 1e-05: 0.844, 0.0001: 0.836, 0.001: 0.856, 0.01: 0.848, 0.1: 0.
798}
reg = np.concatenate((np.linspace(0.0006, 0.001, 3), np.linspace(0.002, 0.008, 4
)))
# {0.0006: 0.838, 0.0008: 0.836, 0.001: 0.856, 0.002: 0.772, 0.004: 0.844, 0.006
: 0.852, 0.008: 0.842}
result = reg_tuning(reg, X_train_dict, y_train, X_val_dict, y_val)
print(result)
```

```
{0.0006: 0.838, 0.0007999999999999999: 0.836, 0.001: 0.856, 0.002: 0.772, 0.004: 0.844, 0.006: 0.852, 0.008: 0.842}
```

5. Kernels

5.1

We represent the two documents x and z in word vectors, $\phi(x)$ and $\phi(z)$, denoting whether the document contains the word. If x contains a word, "apple" and z doesn't, it will be 1 in the "apple" feature of x and 0 in the "apple" feature of z. When you multiply 1 by 0, the result is 0. From the conclusion, we know when the element of the inner product of $\phi(x)$ and $\phi(z)$ is 1, it means both x and z contain the word. This is how I can show $k(x, z) = \phi(x)^T \phi(z)$.

5.2.a

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

$= f(x)f(z)\phi(x)^T\phi(z)$

$= (f(x)\phi(x))^T(f(z)\phi(z))$

$= k(f(x)\phi(x), f(z)\phi(z))$, for any function $f(x) \in \mathcal{R}$

5.2.b

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

$= \phi(x_1)^T \phi(z_1) + \phi(x_2)^T \phi(z_2)$

$= \phi(X)^T \phi(Z) = k(\phi(x), \phi(Z))$, where $\phi(x) = (\phi(x1), \phi(x2))$; $\phi(z) = (\phi(z1), \phi(z2))$

5.2.c

$k_1(x, z)k_2(x, z)$

$$= \left(\sum_{n=1}^{N} \phi_n(x)\phi_n(z) \right)\left(\sum_{m=1}^{M} \phi_m(x)\phi_m(z) \right)$$

$$= \sum_{n=1}^{N} \sum_{m=1}^{M} \left[\phi_n(x)^T \phi_m(x) \right] \left[\phi_n(z)^T \phi_m(z) \right]$$

$$= \sum_{n=1}^{N} \sum_{m=1}^{M} f(x)_{mn} f(z)_{mn} = f(x)^T f(z)$$

5.2

$$\left(1 + \left(\frac{x}{||X||_2} \right)^T \left(\frac{z}{||z||_2} \right) \right)^3$$

$= \left(1 + \left(f_1(x) \right)^T \left(f_1(z) \right) \right)^3$, from the conclusion of 5.2.a, where $f_1(p) = \frac{p}{||p||_2}$

$$= \left(1 + k_1(x, z) \right)^3$$

$= \left(k_2(x, z) \right)^3$, from the conclusion of 5.2.b

$= k_3(x, z)$, from the conclusion of 5.2.c

6. Kernel Pegasos

6.1

$y_j\langle w_{(t)}, x_j\rangle = y_j\langle\sum_{i=1}^{n} \alpha_i^{(t)} x_i, x_j\rangle = y_j \sum_{i=1}^{n} \alpha_i^{(t)}(x_i \cdot x_j) = y_j \sum_{i=1}^{n} \alpha_i^{(t)} k(x_i, x_j) = y_j K_j \alpha^{(t)}$

where K_j is the jth row of the kernel matrix K correspoinding to kernel k.

6.2

$w^{(t+1)} = (1-\eta^{(t)}\lambda)w^{(t)} = (1-\eta^{(t)}\lambda)\sum_{i=1}^{n} \alpha_i^{(t)} x_i = \sum_{i=1}^{n}(1-\eta^{(t)}\lambda)\alpha_i^{(t)} x_i = \sum_{i=1}^{n} \alpha_i^{(t+1)} x_i$

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

6.3

Input: $\lambda > 0$. Choose $w_1 = 0, t = 0$

While termination condition not met

For $j = 1, \ldots, m$ (assumes data is randomly permuted)

$t = t + 1$

$\eta^{(t)} = \frac{1}{(t\lambda)}$

If $y_j K_j \alpha^{(t)} < 1$

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

Else

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

In [ ]: