DS-GA 1003 HW3

Yiyan Chen

February 2018

1 Introduction

2 Calculating Subgradients

2.1 [Subgradients for pointwise maximum of functions]

for any x, $f(x) \geq f_k(x)$; $g \in \partial f_k(x)$, So:

$$
\begin{aligned}
& f(z) \geq f_k(z) \geq f_k(x) + g^T(z-x) \\
& \because f(x) = f_k(x) \\
& \therefore f(z) \geq f(x) + g^T(z-x) \\
& \therefore g \in \partial f(x)
\end{aligned}
$$

2.2 [Subgradient of hinge loss for linear prediction]

if $1 - yw^T x > 0$

$$
\begin{aligned}
& f(z) \geq f(w) + g^T(z-w) \\
& 1 - yz^T x \geq 1 - yw^T x + g^T(z-w) \\
& -yz^T x \geq -yw^T x + g^T(z-w) \\
& \because z - w \text{ could be greater or equal or smaller than } 0 \\
& -y(z-w)^T = g^T(z-w) \\
& -yx^T(z-w) = g^T(z-w) \\
& \therefore \begin{cases} -yx^T, \text{if } yw^T x < 1 \\ 0, \text{if } yw^T x \geq 1 \end{cases}
\end{aligned}
$$

3 Perceptron

3.1

Since $\{x|w^T x = 0\}$ is a separating hyperplane: $y_i w^T x_i > 0, \forall i \in \{1, \ldots, n\}$.

$$l(\hat{y}_i, y_i)/n = l(w^T x_i, y_i)/n = max(0, -y_i w^T x_i)/n = 0/n = 0$$

Thus any separating hyperplane of D is an empirical risk minimizer for perceptron loss.

3.2

stochastic subgradient descent:
initialize: w = 0
repeat:

$$(x_i, y_i) \in D_n$$
$$w \leftarrow w - \eta \partial l(w_i x_i, y_i)$$

in this case, the loss function is $max\{0, -y_i w^T x_i\}$
if $-y_i w^T x_i > 0$, the subgradient is:

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

The last inequality is the same as the one in the previous question, so the subgradient is $-y_i x_i$

$\eta = 1$

$$\begin{cases} w \leftarrow w - (-y_i x_i), y_i w^T x_i < 0 \\ w \leftarrow w - 1 \times 0, y_i w^T x_i \geq 0 \end{cases}$$

$$\begin{cases} w \leftarrow w + y_i x_i, y_i w^T x_i < 0 \\ w \leftarrow w, y_i w^T x_i \geq 0 \end{cases} \quad The last bracket shows it is doing the same as Perceptron algorithm.$$

3.3

At first, the w=0. If the $y_i w^T x_i < 0$, the $w^{(k+1)} = y_i x_i$, and if it continues, the $w^{(k+1)} = y_i x_i + y_i x_i$ is the form of $\sum \alpha_i x_i$. When the direction is zero, the w remains unchanged, $\sum_{i=1}^{m-1} \alpha_i x_i + 0 = \sum_{i=1}^{m} \alpha_i x_i, \alpha_i = 0$

4 The Data

```
def read_data(file):
    f = open(file)
    lines = f.read().split(' ')
    symbols = '${}()[].,:;+-*/&|<>=~" '
    trantab = str.maketrans({ord(k): None for k in list(symbols)})
    words = map(lambda Element: Element.translate(trantab).strip(), lines)
    words = filter(None, words)
    return words

def shuffle_data():
    '''
    pos_path is where you save positive review data.
    neg_path is where you save negative review data.
    '''
    pos_path = "/Users/cyian/Desktop/NYU/SPRING2018/DS-GA1003/hw3-svm/data/pos"
    neg_path = "/Users/cyian/Desktop/NYU/SPRING2018/DS-GA1003/hw3-svm/data/neg"

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

    review = pos_review + neg_review
    random.shuffle(review)

    with open('review.pickle', 'wb') as handle:
        pickle.dump(review, handle, protocol=pickle.HIGHEST_PROTOCOL)

    with open('review.pickle', 'rb') as handle:
        b = pickle.load(handle)
    return b

data = load.shuffle_data()
train,test = train_test_split(data, test_size = 0.25, random_state=42)
```

5 Sparse Representations

```
from collections import Counter
def bag_of_words(sentence):
    cnt = Counter()
    for word in sentence:
        cnt[word]+=1
```

```
6       return cnt
```

6 Support Vector Machine via Pegasos

6.1 [Written]

When the gradient of $J_i(w)$ is not defined, it means the function has a kink, and should be defined by subgradient. It exists when $1 - y_i w^T x_i = 0$. So the function is not defined where $y_i w^T x_i = 1$, while it's defined where $y_i w^T x_i \neq 1$

6.2 [Written]

Based on the rules given and that the two parts of $\frac{\lambda}{2}||w||^2$ and $max0, 1 - y_i w^T x_i$ are convert functions, the subgradient of $J_i(w)$ is $\partial\frac{\lambda}{2}||w||^2 + \partial max0, 1 - y_i w^T x_i$. The calculations in the first problem shows that the $\partial max0, 1 - y_i w^T x_i = -y_i x_i$ if $y_i w^T x_i < 1$ and $\partial\frac{\lambda}{2}||w||^2 = \lambda w$.
Hence:

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

6.3 [Written]

When $y_i w^T x_i < 1$:

$$\begin{aligned} w &\leftarrow w - \eta \partial J(w) \\ &= w - \eta(\lambda w - y_i x_i) \\ &= (1 - \eta\lambda)w + \eta y_i x_i \end{aligned}$$

When $y_i w^T x_i \geq 1$:

$$\begin{aligned} w &\leftarrow w - \eta \partial J(w) \\ &= w - \eta\lambda w \\ &= (1 - \eta\lambda)w \end{aligned}$$

The preceding rules proves doing SGD with the subgradient direction is the same as given in the pseudocode.

6.4

```
def obj_function_for_one(Xi, yi, lamda, w):
    return util.dotProduct(w, w)*lamda/2 + max(0, 1-yi*util.dotProduct(w,Xi))
def obj_function(X,y,lamda,w):
    m = len(y)
    obj = 0
    for idx in range(m):
        obj+=obj_function_for_one(X[idx],y[idx], lamda,w)
    return obj/m

def pegasos(X, y, X_t, y_t, lamda, w = {}, t=0, iteration = 1000):
    start = time.time()
    m = len(y)
    lst = list(range(m))
    itr = 0
    while itr<iteration:
        obj = 0
        np.random.shuffle(lst)
        for idx in lst:
            t +=1
            eta = 1./(t*lamda)
            ywx = y[idx]*util.dotProduct(w, X[idx])
            #d1 = (1-eta*lamda)*w
            util.increment(w, -eta*lamda, w)
            if ywx<1:
                util.increment(w, eta*y[idx], X[idx])
        itr+=1
    end = time.time()
    print("Total time:", end-start)
    print("Loss is: ", obj_function(X_t, y_t, lamda, w))
    return w
```

6.5

$$
\begin{aligned}
W_{t+1} &= W_t + \frac{1}{s_{t+1}}\eta_t y_i x_j \\
s_{t+1}W_{t+1} &= s_{t+1}(W_t + \frac{1}{s_{t+1}}\eta_t y_i x_j) \\
s_{t+1}W_{t+1} &= (1-\eta_t\lambda)s_t W_t + \eta_t y_i x_j \\
w_{t+1} &= (1-\eta_t\lambda)w_t + \eta_t y_i x_j
\end{aligned}
$$

```
def pegasos_sw(X, y, X_t, y_t, lamda, w = {}, t=0, iteration = 1000, threshold = 1e-2):
```

```
    start = time.time()
    m = len(y)
    lst = list(range(m))
    change = 1
    itr = 0
    s = 1
    W = {}
    prev_loss = obj_function(X_t, y_t, lamda, w)
    while change > threshold and itr<iteration:
        np.random.shuffle(lst)
        for idx in lst:
            t +=1
            eta = 1./(t*lamda)
            ywx = y[idx]*util.dotProduct(W, X[idx])
            s = (1-eta*lamda)*s
            if s==0:
                s=1
                W = {}
            elif ywx<1./s:
                util.increment(W, eta*y[idx]/s, X[idx])
        w = {}
        util.increment(w, s, W)
        latter_loss = obj_function(X_t, y_t, lamda, w)
        change = latter_loss-prev_loss
        prev_loss = latter_loss
        itr+=1
    end = time.time()
    print("Total time: {}".format(end-start))
    print("Loss is: ", obj_function(X_t, y_t, lamda, w))
    return w
```

6.6

```
w_non = pegasos(X_train, y_train, X_test, y_test,
        1, {}, 0, iteration = 5)
w_sw = pegasos_sw(X_train, y_train, X_test, y_test,
        lamda = 1, iteration = 5)
```

The result for the pegasos:

```
Total time: 71.24002504348755
Loss is:  0.8189490488890906
```

The result for the pegasos sw function:

```
Total time: 24.786855936050415
Loss is:  0.8821493888888471
```

We can see that the losses for both are quite similar, it is reasonable since I added shuffle in both of the functions. It is proven that these two implements of pegasos algorithm are essentially the same.

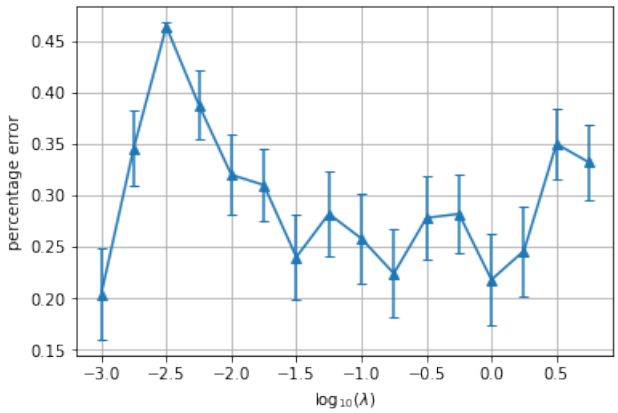
6.7

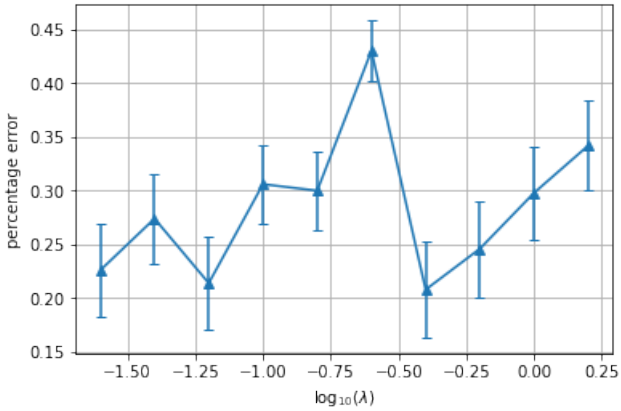
```
def predict(X_i, w):
    if util.dotProduct(X_i, w) > 0:
        return 1
    else:
        return -1

def percent_err(X, y, w):
    m = len(y)
    count_err = 0
    for idx in range(m):
        if predict(X[idx], w)!=y[idx]:
            count_err+=1
    return count_err/m
percent_err(X_test, y_test, w_sw)
```

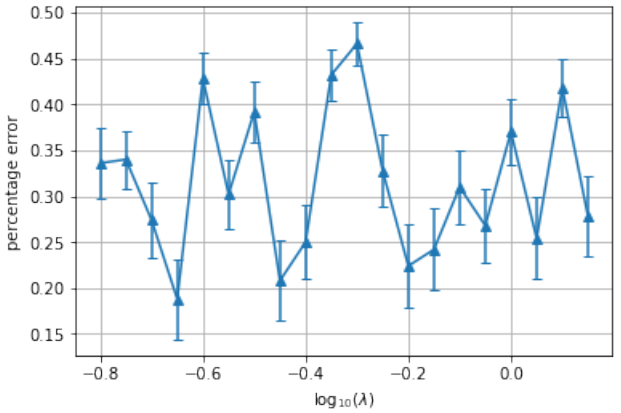
The returned percentage error is 0.286.

6.8

```
def predict_all(X, w):
    m = len(X)
    predict_list = [predict(X[idx], w) for idx in range(m)]
    return predict_list

lambda_list = np.arange(-3,1, 0.25, dtype=float)
err_list = []
std_list = []
for i in 10**lambda_list:
    w_lamda = pegasos_sw(X_train, y_train, X_test, y_test,lamda = i,iteration = 400,
    threshold = 1e-10)
    err_list.append(percent_err(X_test, y_test, w_lamda))
    std_list.append(np.std(predict_all(X_test, w_lamda)))
plt.errorbar(lambda_list, err_list, std_list/np.sqrt(500), marker='^', capsize=3)
plt.xlabel('$\log_{10}(\lambda)$')
plt.ylabel('percentage error')
plt.grid()
plt.show()
```







I kept zooming in until the precision of the lambda. The best performed model is with lambda as $10^{-}0.65 = 0.22$

7 Error Analysis

```
def times_dict(short, long):
    return_dic = {}
    for key, value in short.items():
        if key not in long.keys():
            return_dic[key] = 0
        else:
            return_dic[key] = long[key]*value
    return return_dic
def times_dict_abs(short, long):
    return_dic = {}
    for key, value in short.items():
        if key not in long.keys():
            return_dic[key] = 0
        else:
            return_dic[key] = abs(long[key]*value)
    return return_dic
wixi_0 = times_dict(X_test[diff_idx[0]], w_best)
wixi_1 = times_dict(X_test[diff_idx[1]], w_best)
abs_wixi_0 = times_dict_abs(X_test[diff_idx[0]], w_best)
abs_wixi_1 = times_dict_abs(X_test[diff_idx[1]], w_best)
X_test_fill = X_test[diff_idx[0]].copy()
df_0 = pd.DataFrame([wixi_0, abs_wixi_0, X_test_fill, w_best],
index=['wixi', '|wixi|', 'xi','wi']).T
df_0.sort_values(by='|wixi|', ascending=False)
X_test_fill_2 = X_test[diff_idx[1]].copy()
df_1 = pd.DataFrame([wixi_1, abs_wixi_1, X_test_fill_2, w_best],
index=['wixi', '|wixi|', 'xi','wi']).T
df_1 = df_1.fillna(0)
df_1.sort_values(by='|wixi|', ascending=False)
```

	wixi	|wixi|	xi	wi
and	1.627157	1.627157	20.0	8.135785e-02
the	0.781352	0.781352	51.0	1.532063e-02
to	-0.726409	0.726409	11.0	-6.603722e-02
of	0.498713	0.498713	16.0	3.116957e-02
movie	-0.405733	0.405733	6.0	-6.762211e-02
is	0.348677	0.348677	10.0	3.486765e-02
an	-0.294790	0.294790	6.0	-4.913169e-02
all	-0.282639	0.282639	5.0	-5.652786e-02
this	-0.270488	0.270488	4.0	-6.762211e-02
on	-0.268375	0.268375	4.0	-6.709381e-02
only	-0.263092	0.263092	2.0	-1.315461e-01
a	-0.228225	0.228225	8.0	-2.852808e-02
as	0.223998	0.223998	4.0	5.599956e-02
many	0.215545	0.215545	3.0	7.184849e-02
well	0.200753	0.200753	2.0	1.003766e-01
for	0.167999	0.167999	6.0	2.799978e-02
most	0.135244	0.135244	2.0	6.762211e-02
was	0.131018	0.131018	8.0	1.637723e-02
queen	0.122565	0.122565	8.0	1.532063e-02
there	-0.120452	0.120452	2.0	-6.022594e-02
not	0.110943	0.110943	6.0	1.849042e-02
have	-0.099848	0.099848	1.0	-9.984827e-02
if	-0.097207	0.097207	1.0	-9.720678e-02
great	0.090867	0.090867	1.0	9.086721e-02
war	0.090339	0.090339	3.0	3.011297e-02

	wixi	|wixi|	xi	wi
and	1.627157	1.627157	20.0	8.135785e-02
to	-0.990558	0.990558	15.0	-6.603722e-02
the	0.704749	0.704749	46.0	1.532063e-02
you	0.502939	0.502939	7.0	7.184849e-02
this	-0.473355	0.473355	7.0	-6.762211e-02
of	0.467543	0.467543	15.0	3.116957e-02
just	-0.449053	0.449053	10.0	-4.490531e-02
that	-0.436374	0.436374	14.0	-3.116957e-02
is	0.418412	0.418412	12.0	3.486765e-02
a	-0.399393	0.399393	14.0	-2.852808e-02
great	0.272602	0.272602	3.0	9.086721e-02
on	-0.268375	0.268375	4.0	-6.709381e-02
film	-0.266262	0.266262	8.0	-3.328276e-02
world	0.251470	0.251470	4.0	6.286743e-02
are	-0.237734	0.237734	6.0	-3.962233e-02
what	0.237734	0.237734	5.0	4.754680e-02
be	-0.234564	0.234564	4.0	-5.864105e-02
or	-0.217659	0.217659	4.0	-5.441467e-02
many	0.215545	0.215545	3.0	7.184849e-02
movie	-0.202866	0.202866	3.0	-6.762211e-02
but	-0.173282	0.173282	4.0	-4.332041e-02
very	0.161659	0.161659	2.0	8.082955e-02
bad	-0.155320	0.155320	1.0	-1.553195e-01
like	-0.142640	0.142640	3.0	-4.754680e-02
only	-0.131546	0.131546	1.0	-1.315461e-01

As we can see from the two charts before, all the top features that contribute the most to the model are the neutral words. Those words may mislead the positive and negative tone in the samples since those non-neutral words didn't play an important part.