DS-GA 1003: Homework 3
SVM and Sentiment Analysis

Due on Tuesday, Feb 29, 2016

Professor David Ronsenberg

Yuhao Zhao
Yz3085

2. Calculating Subgradients

• 2.1 Subgradients for pointwise maximum of functions.

Since $f = \max_{i=1,...,,m} f_i(x)$. and f_i is convex
for $\forall x, y, c$

$$f(cy + (1-c)x) = \max_{i=1,...,,m} f_i(cy + (1-c)x) \leq \max_{i=1,...,,m} cf_i(y) + (1-c)f_i(x) = cf(y) + (1-c)f(x)$$

Therefore, f is convex. For any k that $f_k(x) = f(x)$ and $g \in \partial f_k(x)$

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

Therefore,

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

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

• 2.2 Subgradient of hinge loss for linear prediction.

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

A subgradient of J is:

$$\begin{cases} 0 \quad , 1 - yw^T x < 0 \\ -yx, otherwise \end{cases}$$

3. Perceptron

– 3.1.

If the data D are linearly separable, $\forall y_i = 1, \hat{y_i} = w^T x_i < 0$ and $\forall y_i = -1, \hat{y_i} = w^T x_i > 0$
This means

$$\forall y_i, -y_i\hat{y_i} < 0, l(\hat{y_i}, y_i) = max\{0, -\hat{y_i}y_i\} = 0$$

the average perceptron loss on D is therefore 0.

– 3.2.

If we run SGD to minimize the empirical risk, the perceptron loss is $l(\hat{y_i}, y_i) = max\{0, -y_i w^T x_i\}$
A subgradient is

$$\begin{cases} 0 \quad , -y_i w^T x_i \leq 0 \\ -y_i x_i, otherwise \end{cases}$$

If we set step size = 1, we update w by:

$$w_{i+1} = \begin{cases} w_i \quad , -y_i w_i^T x_i \leq 0 \\ w_i + y_i x_i, otherwise \end{cases}$$

This is exactly the Perceptron Algorithm.

– **3.3.**

From the Perceptron Algorithm, in each step we update w_i by adding $y_i x_i$ or 0 to w_i.
Therefore, our output is actually $w_n = \sum \alpha_i x_i$
For x_i that is a support vector, it satisfies that $-y_i w_i^T x_i > 0$, which w_i is the weight vector at the i-th iteration.

4. The Data

```
def shuffle_data():
'''
pos_path is where you save positive review data.
neg_path is where you save negative review data.
return: shuffled data
'''
        pos_path = "/Users/cryan/Desktop/1003/github/as3/data/neg"
        neg_path = "/Users/cryan/Desktop/1003/github/as3/data/pos"

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

        review = pos_review + neg_review
        random.shuffle(review)
return np.array(review)

data = np.array(shuffle_data())
data_train = data[:1500]
data_test = data[1500:]

np.save("X_train", map(lambda x: x[:-1],data_train))
np.save("X_test", map(lambda x: x[:-1],data_test))
np.save('Y_train', map(lambda x: x[-1],data_train))
np.save('Y_test', map(lambda x: x[-1],data_test))
```

5. Sparse Representation

```
def tokenlizer(text):
'''
converts a list of words into bag-of-words

Args:
text: a list of words

Returns: a dictionary/ hash table of text
'''
        res = Counter()
        for i in text:
                res[i] +=1
return res
```

6. Support Vector Machine via Pegasos

• 3.1.

Since the objective function is :

$$L = \frac{\lambda}{2}||w||^2 + \frac{1}{m}\sum max\{0, 1 - y_i w^T x_i\}$$

With SGD, in each step, we are updating w w.r.t $\frac{\lambda}{2}||w||^2 + max\{0, 1 - y_i w^T x_i\}$
This objective function is convex, one subgradient is :

$$\begin{cases} \lambda w \quad , 1 - y_i w^T x_i \leq 0 \\ \lambda w - y_i x_i, otherwise \end{cases}$$

Then the corresponding update, if we choose $\eta_t = \frac{1}{\lambda t}$is :

$$w_{i+1} = \begin{cases} w_i - \lambda \eta_i w_i = (1 - \eta_i \lambda) w_i \quad , 1 - y_i w^T x_i \leq 0 \\ w_i - \eta_i(\lambda w_i - y_i x_i) = (1 - \eta_i \lambda) w_i + \eta_i y_i x_i, otherwise \end{cases}$$

This update is identical to the pseudocode.

• 3.2.

```
def pegasos_svm_sgd(X,y,lambda_ = 10,n_ite = 1,print_time = False):
'''
pegasos svm with pure sgd approach

Args:
X: Train data
y: Train lable
lambda_: regulization
n_ite: max iterations
print_time: whether count the operation time

Returns: sparse representaion of the weight

'''
X = np.array(map(lambda a: tokenlizer(a) , X),dtype = object)
num_instances = X.shape[0]
t = 0.0
n = 0
w = Counter()
time_ = time.time()
while n < n_ite:
        generator = np.random.permutation(list(xrange(num_instances))) # define ramdom sampling sequenc
        for i in generator:
                t+=1
                eta = 1/(t*lambda_)
                if dotProduct(w,X[i])*y[i] <1:
                        increment(w,- eta*lambda_,w)
```

```
                        increment(w,eta*y[i],X[i])
                else:
                        increment(w,- eta*lambda_,w)
        n+=1
if print_time:
        print( time.time() -time_ )
return w
```

- **3.3.**

Since $s_{t+1} = (1 - \eta_t \lambda) s_t, w = sW$:
Our new update rule is :

$$W_{t+1} = W_t + \frac{1}{s_{t+1}} \eta_t y_j x_j$$

Then

$$\frac{w_{t+1}}{s_{t+1}} = \frac{w_t}{s_t} + \frac{1}{s_{t+1}} \eta_t y_j x_j$$

Therefore we have:

$$w_{t+1} = \frac{s_{t+1}}{s_t} w_t + \eta_t y_j x_j = (1 - \eta_t \lambda) w_t + \eta_t y_j x_j$$

This is equivalent to the Pegasos algorithm. Our new condition will be

$$y_j W_t^T x_j < \frac{1}{s_t}$$

Code:

```
def scale(x,c):
'''
scale dicitionary by constant c

Args:
x: input weight vector
c: increment constant

Returns: scaled dictionary

'''
if x == Counter():
        return x
else:
        temp = np.array(x.items(),dtype= object)
        temp[:,1] = temp[:,1]*c
        return dict(temp)


def pegasos_svm_sgd_2(X,y,lambda_ = 1,n_ite = 10,counter = False,print_time = False):
'''
updated pegasos svm with pure sgd approach
```

```
    Args:
    X: Train data
    y: Train lable
    lambda_: regulization
    n_ite: max iterations
    counter: whether count the # of nondifferentiable case
    print_time: whether count the operation time

    Returns: sparse representaion of the weight

    '''
    X = np.array(map(lambda a: tokenlizer(a) , X),dtype = object)
    num_instances = X.shape[0]
    t = 1.0
    n = 0
    W = Counter()
    s=1.0
    count = 0.0
    time_ = time.time()
            while n < n_ite:
            generator = np.random.permutation(list(xrange(num_instances))) # define ramdom sampling sequenc
            for i in generator:
                    t+=1
                    eta = 1/(t*lambda_)
                    s = (1 - eta*lambda_)*s
                    temp = dotProduct(W,X[i])
                    if temp ==0 and counter==True:
                            count+=1.0
                    if temp*y[i] <1/s:
                            increment(W,1/s *eta *y[i], X[i])
            n+=1
    if print_time:
            print( time.time() -time_ )
    if counter:
            print count/(num_instances*n_ite)
    return scale(W,s)
```

- **3.4.**

```
>>> pegasos_svm_sgd(X_train,y_train,1,1,print_time = True)
41.4353728294
>>>w2 = pegasos_svm_sgd_2(X_train,y_train,1,1,print_time = True) # 0.38 s
0.291805028915
>>> Counter(w2).most_common(3)
[('bad', 0.1485676215856098), ('have', 0.06826211585894239), ('any', 0.08860759493670896)]
>>> Counter(w1).most_common(3)
[('bad', 0.12391738840772837), ('have', 0.09593604263824126), ('this', 0.08927381745503007)]
```

The first algorithm takes 41 s to do 1 iteration and the second takes only 0.292s. And the two algorithms returns a similar weight. As we can see, the most 2 heavily weighted words in w1 and w2 are the similar, while the third are not.

- **3.5.**

```
def loss_0_1(X,y,w):
'''
0_1 loss function for svm

Args:
X: testing data
y: true lables
w: weight

Returns: loss

'''
        X = np.array(map(lambda a: tokenlizer(a) , X),dtype = object)
return np.mean(map(lambda t: 0 if t>0 else 1, np.array(map(lambda t: dotProduct(w,t),X))*y))
```

- **3.6.**

```
>>> try_list = np.power(10.0,list(range(-8,5)))
    loss_list = np.zeros(13)
    for i,j in enumerate(try_list):
            w = pegasos_svm_sgd_2(X_train,y_train,lambda_ = j,n_ite = 10)
            loss_list[i] = loss_0_1(X_test,y_test,w)
>>> loss_list
array([ 0.218,  0.21 ,  0.22 ,  0.204,  0.204,  0.226,  0.288,  0.23 ,
0.246,  0.346,  0.526,  0.526,  0.526])
>> try_list_2
array([ 0.202,  0.21 ,  0.222,  0.204,  0.22 ,  0.22 ,  0.212,  0.216,
0.22 ,  0.206,  0.214,  0.206,  0.226,  0.208,  0.206,  0.312,
0.21 ,  0.206,  0.218,  0.22 ])
>> lambda_opt = try_list_2[np.where(loss_list_2 == min(loss_list_2))[0][0]]
w_opt = pegasos_svm_sgd_2(X_train,y_train,lambda_ = lambda_opt,n_ite = 20)
>> lambda_opt
1.0000000000000001e-05
```

The best regularization that gives the lowest loss is 10^{-5}