Machine Learning HW03

Ben Zhang, bz957

February 23, 2018

2 Calculating Subgradients

HW 3.

2.1. $\because g \in \partial f_k(x)$

$\therefore f_k(x_0) + g^T \cdot (z - x_0) \leq f_k(z)$

$\because f_k(x_0) = f(x) = \max\limits_{i=1\dots m} f_i(x)$, k be any index

$\Rightarrow f(x_0) + g^T(z - x_0) \leq f(z)$

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

2.2 $J(w) = \max\{0, 1 - y w^T x\}$

$$\partial J(w) = \begin{cases} 0, & y w^T x > 1 \\ [-yx, 0], & y w^T x = 1 \\ -yx, & y w^T x < 1 \end{cases}$$

3 Perceptron

3.1

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

$$\frac{1}{n}\sum_{i=1}^{n} \ell(\hat{y}_i, y_i) = \frac{1}{n}\sum_{i=1}^{n} \max\{0, -\hat{y}_i y_i\}$$

if $\{x \mid w^T x = 0\}$ is a hyperplane,

then $y = 1, -1$ are on 2 sides of the hyperplane

$\Rightarrow \hat{y}_i y_i > 0$ for $i \in [1, n]$.

$\therefore \frac{1}{n}\sum_{i=1}^{n} \ell(\hat{y}_i, y_i) = \frac{1}{n}\sum_{i=1}^{n} 0 = 0$

then the average perceptron loss on D is 0

for any separating hypeplane of D,

e.g. $\{x \mid w^T x = \varepsilon\}$, $\varepsilon \in R$

$y = 1, -1$ are on different sides of the hyperplane

$\Rightarrow \hat{y}_i y_i = (w^T x_i - \varepsilon) \cdot y_i > 0$ for $i \in [1, n]$

$\frac{1}{n}\sum_{i=1}^{n} \ell(\hat{y}, y) = 0$

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

3.2

In SSGD, $g = \begin{cases} 0 , & yw^Tx > 0 \\ -yx , & yw^Tx < 0 \\ [-yx, 0] , & yw^Tx = 0 \end{cases}$

we need to find $w^{k+1} = w^k - \eta \cdot g^k$

repeat k, $J_{best}^k = \min \{ f(w)^1, \dots, f(w)^k \}$

$\ell(\hat{y}, y) = \max(0, -\hat{y}y) = \max(0, -\hat{y}w^Tx)$

for i in $[1, n]$, if $y_i w^T x_i > 0$, $w^{k+1} = w^k - 0$

if $y_i w^T x_i < 0$, $w^{k+1} = w^k - (-y_i x_i)$

$= w^k + y_i x_i$

continue, till $J_{best} = \min \{ J(w)^1 \dots J(w)^k \}$

until $y_i w^T x_i > 0$

so SSGD is same as Perceptron Algorithm

3.3

according the Peceptron Algorithm,

$$\text{if } (y_i x_i^T w^{(k)} \le 0)$$

$$w^{k+1} = w^{k} + y_i x_i$$

and because $w = \sum_{i=1}^{n} \alpha_i x_i$ for some $\alpha_1 \ldots \alpha_n \in R$

$\Rightarrow$ the supported vector has once (one time or more) been classified on the wrong side of the hyperplane,

the unsupported vector has never been classified on the wrong side of the hyperplane

4 The Data

```
import os
import numpy as np
import pickle
import random
import matplotlib.pyplot as plt
import pandas as pd


def shuffle_data():
    '''
    pos_path is where you save positive review data.
    neg_path is where you save negative review data.
    '''
    pos_path = "/Users/zhangben/Google Drive/NYU/Classes/1003MachineLearning/hw/
               hw3-svm/data/pos"
    neg_path = "/Users/zhangben/Google        Drive/NYU/Classes/1003MachineLearning/hw
               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)

    return review

'''
Now you have read all the files into list 'review' and it has been shuffled.
Save your shuffled result by pickle.
*Pickle is a useful module to serialize a python object structure.
*Check it out. https://wiki.python.org/moin/UsingPickle
'''


review=shuffle_data()
pickle.dump(review, open( "review.p", "wb" ))


from sklearn.model_selection import train_test_split

review = pickle.load(open( "review.p", "rb" ))

review_X = list(i[:-1] for i in review)
review_Y = list(i[-1] for i in review)

X_train, X_test, y_train, y_test = train_test_split(review_X, review_Y,
                                        train_size=0.75, random_state=42)
```

5 Sparse Representations

```
from collections import Counter
def counter(bag):
    cnt = Counter()
    for word in bag:
        cnt[word] += 1
    return cnt
```

6 Support Vector Machine via Pegasos

6.1-6.3

6.1. $$\nabla_w \max\{0, 1 - y_i w^T x_i) = \begin{cases} 0 & , y_i w^T x_i > 1 \\ -y_i x_i & , y_i w^T x_i < 1 \\ \text{undefined} & , y_i w^T x_i = 1 \end{cases}$$

$$\nabla_w J(w) = \nabla w \left(\frac{\lambda}{2}\|w\|^2 + \max\{0, 1 - y_i w^T x_i\right)$$

$$= [\lambda w + \nabla_w \max\{0, 1 - y_i w^T x_i\}$$

$$= \begin{cases} \lambda w & , y_i w^T x_i > 1 \\ \lambda w - y_i x_i & , y_i w^T x_i < 1 \\ \text{undefined} & , y_i w^T x_i = 1 \end{cases}$$

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

if $y_i w^T x_i < 1$, $J_i(w) = \frac{\lambda}{2}\|w\|^2 + (1 - y_i w^T x_i)$

$$\nabla J_i(w) = \lambda w - y_i x_i$$

if $y_i w^T x_i > 1$, $\nabla J_i(w) = \lambda w$

if $y_i w^T x_i = 1$, $y = \lambda w$,

we need to prove this:

$$f(x+\Delta x) \geq f(x) + \lambda w \cdot \Delta x, \quad \forall \Delta x \geq 0$$

$$f(x+\Delta x) \geq f(x) + (\lambda w - y_i x_i) \cdot \Delta x$$

$$> f(x) + \lambda w \Delta x, \quad \forall \Delta x < 0$$

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

6.3. $\eta_t = 1/(\lambda t)$,

$$\min\{\frac{\lambda}{2}\|w\|^2 + \frac{1}{n}\sum_{i=1}^{n} \max\{0, 1 - y_i w^T x_i\}$$

$y_i w^T x_i < 1$, $g = \lambda w - y_i x_i$,

$$w_{t+1} = w_t - \eta_t g_t$$
$$= w_t - (\lambda w_t - y_i x_i) \cdot \eta_t$$
$$= (1 - \eta_t \lambda) w_t + \eta_t y_i x_i$$

else, $g = \lambda w$

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

So $\eta_t = 1/(t\lambda)$, then doing SGD with subgradient direction from the previous problem is the same as given in pseudocode.

6.4

```
X_train = list(counter(X_train[i]) for i in range(len(X_train)))
X_test = list(counter(X_test[i]) for i in range(len(X_test)))

def dotProduct(d1, d2):
    """
    @param dict d1: a feature vector represented by a mapping from a feature (string)
    @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


def Pegasos(X,y,Lambda,epoch):
    w = {}
    t = 0
    l = len(X)
    for _ in range(epoch):
        for i in range(l):
            t += 1
            s = 1/(t*Lambda)
            for f, v in w.items():

                w[f] = v * (1-s*Lambda)
            if y[i]*dotProduct(w,X[i]) <1:
                for f, v in X[i].items():
                    w[f] = w.get(f, 0) + v * s*y[i]

    return w
```

6.5

6.5. $\because w = sW$,

$$\therefore W_{t+1} = \frac{w_{t+1}}{s_{t+1}} = \frac{(1-\eta_t\lambda)w_t + \eta_t y_i x_i}{(1-\eta_t\lambda)\cdot s_t}$$

$$= \frac{w_t}{s_t} + \frac{1}{(1-\eta_t\lambda)s_t}\cdot \eta_t y_i x_i$$

$$= W_t + \frac{1}{s_{t+1}}\cdot \eta_t y_i x_i$$

$\Rightarrow$ Pegasos update step is equivalent to:

$$s_{t+1} = (1-\eta_t\lambda)s_t$$

$$W_{t+1} = W_t + \frac{1}{s_{t+1}}\eta_t y_i x_i$$

```
def Pegasos_m(X,y,Lambda,epoch):
    W = {}
    w = {}
    t = 1
    s = 1
    l = len(X)
    for _ in range(epoch):
        for i in range(l):
            t += 1
```

```
            eta = 1/(t*Lambda)
            s = (1-eta*Lambda)*s
            if s*y[i]*dotProduct(W,X[i]) <1:
                for f, v in X[i].items():
                    W[f] = W.get(f, 0) + v * 1/s*eta*y[i]
        for f, v in W.items():
            w[f] = v * s

    return w
```

6.6

```
import timeit
print(timeit.timeit("Pegasos(X_train,y_train,0.01,5)",
                          setup = "from __main__ import Pegasos,
                              X_train,y_train", number=1))

import timeit
print(timeit.timeit("Pegasos_m(X_train,y_train,0.01,5)",
                          setup = "from __main__ import Pegasos_m,
                              X_train,y_train", number=1))
```

The time of $Pegasos(X_train, y_train, 0.01, 5)$ is 49.96064996905625, The time of $Pegasos_m(X_train, y_train, 0.01, 5)$ is 1.2036136691458523

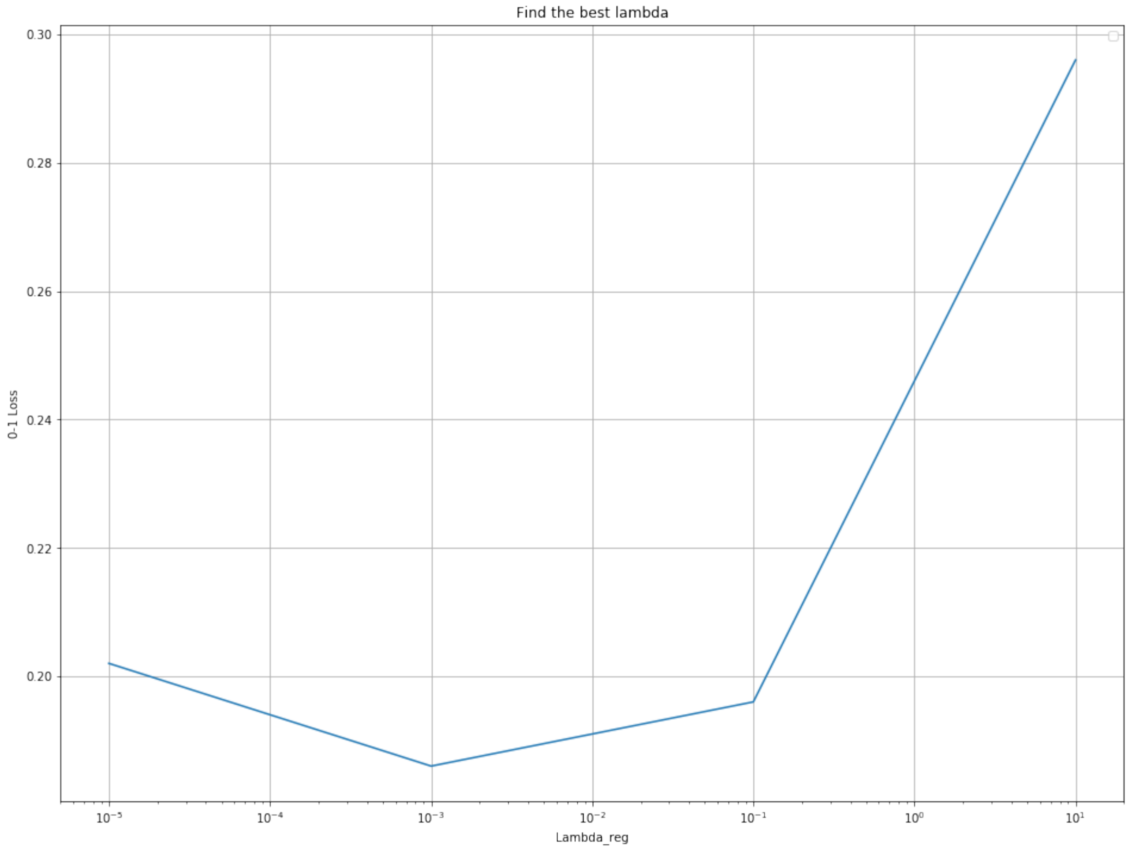
6.7

```
def Pegasos_Loss(X,y,w):
    l = len(X)
    x = list(counter(X[i]) for i in range(l))
    loss_sum = 0
    for i in range(l):
        if np.sign(dotProduct(w,X[i])) != y[i]:
            loss_sum += 1
    loss = loss_sum/l

    return loss

loss = Pegasos_Loss(X_train,y_train,w)
>>>loss=0.068

def Pegasos_m_Loss(X,y,w):
    l = len(X)
    x = list(counter(X[i]) for i in range(l))
    loss_sum = 0
    for i in range(l):
        if np.sign(dotProduct(w,X[i])) != y[i]:
            loss_sum += 1
    loss = loss_sum/l

    return loss

loss_m = Pegasos_m_Loss(X_train,y_train,w)
>>>loss_m=0.068
```

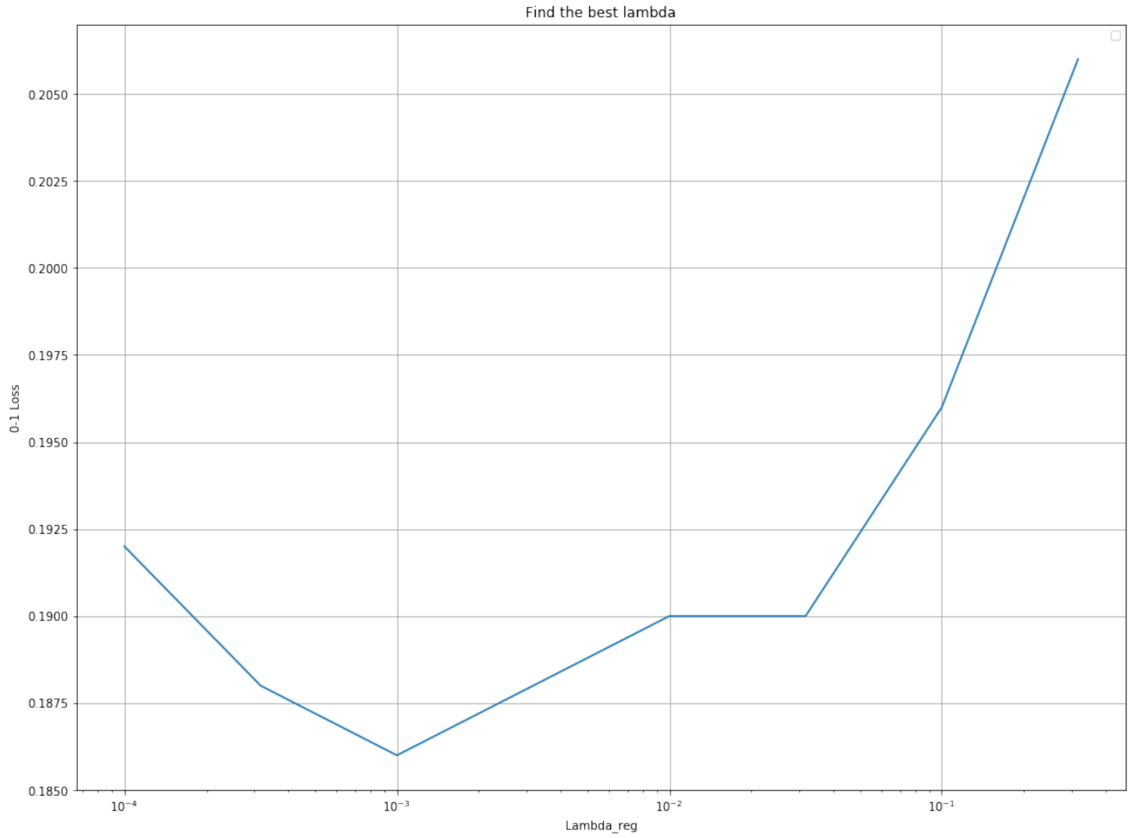
6.8

```
def Pegasos_lambda(X,y,Lambda,epoch):
    W = {}
```

```
    w = {}
    t = 1
    s = 1
    l = len(X)
    losshist=[]
    losshist.append(float("inf"))

    for e in range(epoch):
        for i in range(l):
            t += 1
            eta = 1/(t*Lambda)
            s = (1-eta*Lambda)*s
            if s*y[i]*dotProduct(W,X[i]) <1:
                for f, v in X[i].items():
                    W[f] = W.get(f, 0) + v * 1/s*eta*y[i]
        for f, v in W.items():
            w[f] = v * s

        loss_sum = 0
        for i in range(l):
            loss_sum += max(0, 1-y[i]*dotProduct(w,X[i]))
        losshist.append(Lambda/2*dotProduct(w,w)+1/l*loss_sum)

        if (np.absolute(losshist[e]-losshist[e+1]))<1e-10:
            break

    return w


lambda_grid = np.unique(10.**np.arange(-5,3,2))
losshist = []
for l in lambda_grid:
    w = Pegasos_lambda(X_train,y_train,l,1000)
    losshist.append(Pegasos_m_Loss(X_test, y_test, w))


fig, ax = plt.subplots(figsize = (16, 12))
ax.semilogx(lambda_grid, losshist)
ax.set_xlabel("Lambda_reg")
ax.set_ylabel("0-1 Loss")
ax.set_title("Find the best lambda")
legend = ax.legend(loc='best')
legend.FontSize = 8
plt.grid(True)
plt.show()
```




```
lambda_grid = np.unique(10.**np.arange(-4,0,0.5))
losshist = []
for l in lambda_grid:
    w = Pegasos_lambda(X_train,y_train,l,1000)
    losshist.append(Pegasos_m_Loss(X_test, y_test, w))

fig, ax = plt.subplots(figsize = (16, 12))
ax.semilogx(lambda_grid, losshist)
ax.set_xlabel("Lambda_reg")
ax.set_ylabel("0-1 Loss")
ax.set_title("Find the best lambda")
legend = ax.legend(loc='best')
legend.FontSize = 8
plt.grid(True)
plt.show()
```

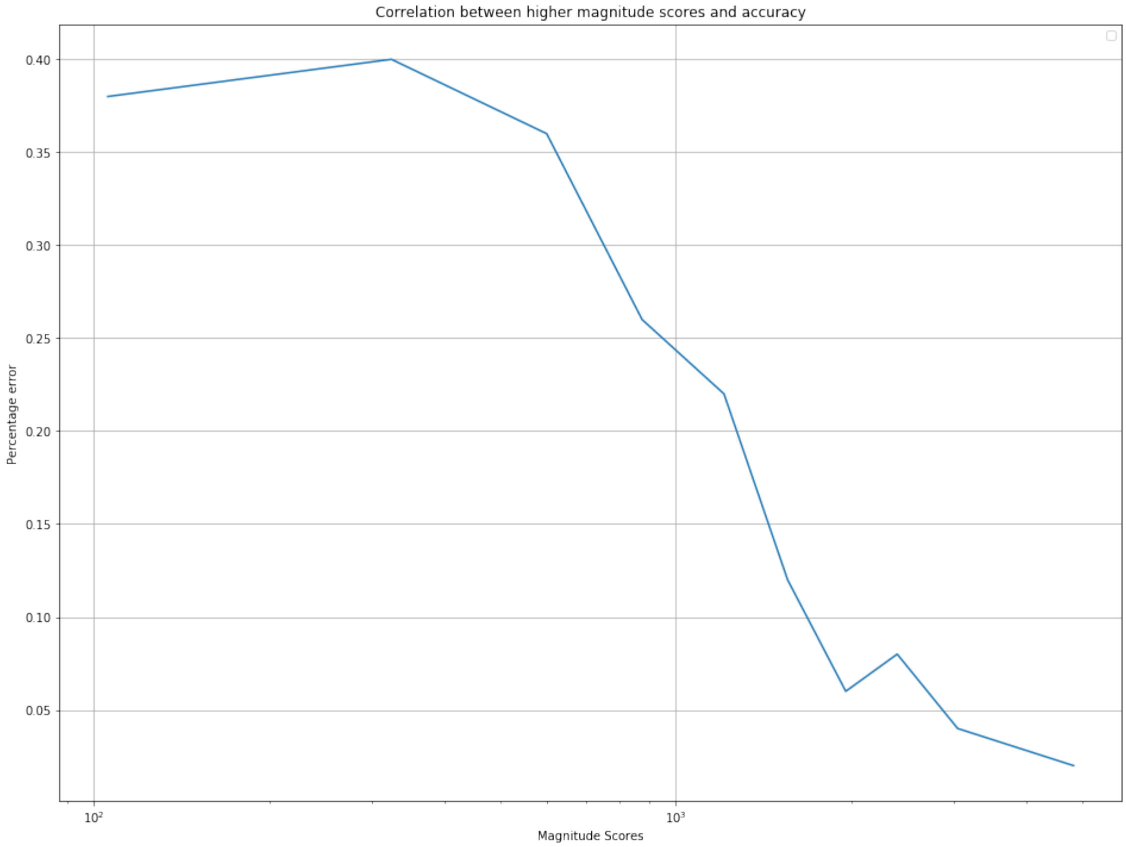


$\Lambda = 10^{-3}$ **minimizes the 0-1 loss on the validation set**

6.9

```
w=Pegasos_lambda(X_train,y_train,0.001,1000)
import collections
def confidence(X,y,w, number):
    df=pd.DataFrame()
    scores = []
    l=len(X)
    for i in range(l):
        scores.append(np.absolute(dotProduct(w,X[i])))

    df["Score Magnitude"] = scores
    df["Test Example"] = X
    df["Y"] = y

    df_sorted = df.sort_values(by='Score Magnitude',ascending = False)
    scores_sum = []
    accuracy = []
    s = l//number
    for p in range(0, l, s):
        scores_sum.append(sum(df_sorted["Score Magnitude"].iloc[p:p+s]))
        testx = df_sorted["Test Example"].iloc[p:p+s]
        testy = df_sorted["Y"].iloc[p:p+s]
        accuracy.append(Pegasos_m_Loss(testx.values,testy.values,w))
    return scores_sum, accuracy
score, accuracy = confidence(X_test,y_test,w, 10)
fig, ax = plt.subplots(figsize = (16, 12))
ax.semilogx(score, accuracy)
ax.set_xlabel("Magnitude Scores")
ax.set_ylabel("Percentage error")
ax.set_title("Correlation between higher magnitude scores and accuracy")
```

```
legend = ax.legend(loc='best')
legend.FontSize = 8
plt.grid(True)
plt.show()
```



The higher magnitude scores, the lower percentage error, the higher accuracy

6.10

```
def Pegasos_count1(X,y,Lambda,epoch):
    W = {}
    w = {}
    t = 1
    s = 1
    l = len(X)
    count = 0
    for _ in range(epoch):
        for i in range(l):
            t += 1
            eta = 1/(t*Lambda)
            s = (1-eta*Lambda)*s
            cond = s*y[i]*dotProduct(W,X[i])
            if cond <1:
                for f, v in X[i].items():
                    W[f] =W.get(f, 0) + v * 1/s*eta*y[i]
            elif cond<= 1 +1e-2 and cond>=1 - 1e-2:
                count +=1
        for f, v in W.items():
            w[f] = v * s

    return count
count1 = Pegasos_count1(X_train,y_train,0.001,50)
count1
```

Let $Lambda_reg$ = 0.001**(best** $Lambda_reg$ **I found above), I runned 50 epochs of train dataset, I get 1 time when** $0.99 <= y_i W^T x_i <= 1.01$**. So it it not very often when** $y_i W^T x_i = 1$ **or close to 1.**

```
w = Pegasos_m(X_train, y_train, 0.001, 50)
loss_original = Pegasos_m_Loss(X_test, y_test, w)
>>>loss_original = 0.204

def Pegasos_skip(X, y, Lambda, epoch):
    W = {}
    w = {}
    t = 1
    s = 1
    l = len(X)

    for _ in range(epoch):
        for i in range(l):
            t += 1
            eta = 1/(t*Lambda)
            s_backup = s
            s = (1-eta*Lambda)*s
            cond = s*y[i]*dotProduct(W, X[i])

            if cond<= 1 +1e-2 and cond>=1 - 1e-2:
                s = s_backup
            elif cond <1 +1e-2:
                for f, v in X[i].items():
                    W[f] = W.get(f, 0) + v * 1/s*eta*y[i]

        for f, v in W.items():
            w[f] = v * s

    return w

w_skip  = Pegasos_skip(X_train, y_train, 0.001, 50)
loss_skip = Pegasos_m_Loss(X_test, y_test, w_skip)
>>>>loss_skip = 0.192


def Pegasos_narrowstep(X, y, Lambda, epoch):
    W = {}
    w = {}
    t = 1
    s = 1
    l = len(X)

    for _ in range(epoch):
        for i in range(l):
            t += 1
            eta = 1/(t*Lambda)
            s_backup = s
            s = (1-eta*Lambda)*s
            cond = s*y[i]*dotProduct(W, X[i])

            if cond<= 1 +1e-2 and cond>=1 - 1e-2:
                s = (1-eta*0.8*Lambda)*s_backup
            elif cond <1 +1e-2:
                for f, v in X[i].items():
                    W[f] = W.get(f, 0) + v * 1/s*eta*y[i]
```

```
        for f, v in W.items():
            w[f] = v * s

    return w

w_narrowstep  = Pegasos_narrowstep(X_train, y_train, 0.001, 50)
loss_narrowstep = Pegasos_m_Loss(X_test, y_test, w_narrowstep)
>>>>loss_narrowstep=0.19
```

Conclusion: compared the original svm Pegasos algorithm achieve 0.204 error percentage, while skipping update algorithm using the same dataset and parameter gets 0.192, the shortening algorithm gets 0.19. So when come to $y_i * w^t * x_i = 1$, shortening the step size by a small percentage achieves the best result

7 Error Analysis

```
w=Pegasos_lambda(X_train, y_train, 0.001, 1000)
l = len(X_test)
incorrect = []
index = []
for i in range(l):
    if np.sign(dotProduct(w, X_test[i])) != y_test[i]:
        incorrect.append(X_test[i])
        index.append(i)
        if len(incorrect) >3:
            break


def ErrorAnalysis(example, w):
    d_feature = []
    d_abwixi=[]
    d_xi=[]
    d_wi=[]
    d_xiwi=[]
    df = pd.DataFrame()
    for f, v in example.items():
        d_feature.append(f)
        d_abwixi.append(np.absolute(w.get(f, 0)* v))
        d_xi.append(v)
        d_wi.append(w.get(f, 0))
        d_xiwi.append((w.get(f, 0)* v))
    df["Feature of Example"] = d_feature
    df["|WiXi| of Example"] = d_abwixi
    df["Xi of Example"] = d_xi
    df["Wi of Example"] = d_wi
    df["WiXi of Example"] = d_xiwi

    return df

for e in incorrect:
    df = ErrorAnalysis(e, w).sort_values(by='|WiXi| of Example', ascending=False)
    print(df)
```

	Feature of Example	|WiXi| of Example	Xi of Example	Wi of Example	WiXi of Example
7	and	7.279903e+00	7	1.039986e+00	7.279903e+00
4	of	3.999947e+00	6	-6.666578e-01	-3.999947e+00
33	the	3.399955e+00	15	2.266636e-01	3.399955e+00
18	will	3.093292e+00	2	1.546646e+00	3.093292e+00
103	worst	2.839962e+00	1	-2.839962e+00	-2.839962e+00
101	was	2.519966e+00	3	8.399888e-01	2.519966e+00
153	well	2.333302e+00	1	2.333302e+00	2.333302e+00
108	only	2.266636e+00	1	-2.266636e+00	-2.266636e+00
65	plot	2.239970e+00	1	-2.239970e+00	-2.239970e+00
85	when	2.239970e+00	3	-7.466567e-01	-2.239970e+00
61	he	2.186638e+00	2	1.093319e+00	2.186638e+00
15	two	2.186638e+00	2	-1.093319e+00	-2.186638e+00
38	few	2.186638e+00	2	1.093319e+00	2.186638e+00
6	characters	1.999973e+00	2	-9.999867e-01	-1.999973e+00
31	in	1.839975e+00	6	3.066626e-01	1.839975e+00
30	?	1.839975e+00	2	-9.199877e-01	-1.839975e+00
37	right	1.826642e+00	1	1.826642e+00	1.826642e+00

I analyzed 4 examples, finding there are two reasons causing wrong prediction:1, stop words like 'and','of' take too much weight, while the majority predict value of sentimental words are equals zero, affecting the prediction accuracy.2,Some possitive or negative words appeared after 'not', 'isn't'. Solutions: 1,Remove stop words like 'and','the', 'of' from the features. 2,Creating bigram features, such as transforming 'not', 'good' into 'not good' or 'nongood',this will fix the problem

8 Features

8.1

```
review=shuffle_data()
pickle.dump(review, open( "review.p", "wb" ))

from sklearn.model_selection import train_test_split

review = pickle.load(open( "review.p", "rb" ))
review_X = list(i[:-1] for i in review)
review_Y = list(i[-1] for i in review)
X_train, X_test, y_train, y_test = train_test_split(review_X, review_Y, train_size=0.

def ngrams(input, n):
  output = []
  for i in range(len(input)-n+1):
    output.append(input[i:i+n])
  return output

from collections import Counter
```

```
def counter(bag):
    cnt = Counter()
    for word in bag:
        cnt[word] += 1
    return cnt

X_train = ngrams(X_train,2)
X_test = ngrams(X_test,2)

X_train = list(counter(X_train[i]) for i in range(len(X_train)))
X_test = list(counter(X_test[i]) for i in range(len(X_test)))

w = Pegasos(X_train,y_train,0.001,50)
Loss_ngrams = Pegasos_m_Loss(X_test,y_test,w)

>>>Loss_ngrams = 0.16
loss_original = Pegasos_m_Loss(X_test,y_test,w)
>>>loss_original = 0.204
```

I used ngrams to construct a group of features. this feature pre-processing combines words like 'not', 'good' into a feature 'not good', which will avoid the wrong prediction somehow. The test results is apparently better than the original one, which improve from 0.204 to 0.183

8.2

```
from string import punctuation
from nltk.corpus import stopwords
from nltk import word_tokenize

review=shuffle_data()
pickle.dump(review, open( "review.p", "wb" ))

from sklearn.model_selection import train_test_split

review = pickle.load(open( "review.p", "rb" ))
review_X = list(i[:-1] for i in review)
review_Y = list(i[-1] for i in review)
X_train, X_test, y_train, y_test = train_test_split(review_X, review_Y, train_size=0.

def ngrams(input, n):
  output = []
  for i in range(len(input)-n+1):
    output.append(input[i:i+n])
  return output

from collections import Counter
def counter(bag):
    cnt = Counter()
    for word in bag:
        cnt[word] += 1
    return cnt

def tokenize(text):
    words = word_tokenize(text)
    words = [w.lower() for w in words]
    return [w for w in words if w not in stop_words and not w.isdigit()]
def tfidf(X):
    stop_words = stopwords.words('english') + list(punctuation)
    vocabulary = set()
```

```
    for i in X:
        words = tokenize(X[i])
        vocabulary.update(words)

    vocabulary = list(vocabulary)
    word_index = {w: idx for idx, w in enumerate(vocabulary)}

    VOCABULARY_SIZE = len(vocabulary)
    DOCUMENTS_COUNT = len(X)

    word_idf = defaultdict(lambda: 0)
    for i in X:
        words = set(tokenize(X[i]))
        for word in words:
            word_idf[word] += 1

    for word in vocabulary:
        word_idf[word] = math.log(DOCUMENTS_COUNT / float(1 + word_idf[word]))

    return [word_tf[word] * word_idf[word] word for i in len(X)]

X_train = ngrams(X_train,2)
X_test = ngrams(X_test,2)

X_train = list(counter(X_train[i]) for i in range(len(X_train)))
X_test = list(counter(X_test[i]) for i in range(len(X_test)))

X_train = list(tfidf(X_train[i]) for i in range(len(X_train)))
X_test = list(tfidf(X_test[i]) for i in range(len(X_test)))

w = Pegasos(X_train,y_train,0.001,50)
Multiple_features = Pegasos_m_Loss(X_test,y_test,w)
>>>Multiple_features = 0.174
```

I combined ngrams and tiidf to construct a group of features. The test results is apparently better than the original one, which improve from 0.204 to 0.174