HW4

Friday, April 7, 2017 2:37 AM

1a. Let $K(x,y) = g(x)g(y)$ where g is a real valued function

If for any $\{x_1, \ldots, x_n\} \subset X^d$

the matrix: $\begin{bmatrix} K(x_1,x_1) & \cdots & K(x_1,x_n) \\ \vdots & \ddots & \vdots \\ K(x_n,x_1) & \cdots & K(x_n,x_n) \end{bmatrix}$

is symmetric PSD, then K is PSD

K is symmetric if $K(x,x') = K(x',x)$

Using $K(x,y) = g(x)g(y) = g(y)(x)$, so K is symmetric

K is PSD if for any $a \in \mathbb{R}^n$, $a^T K a \geq 0$

Let $K = \begin{bmatrix} g(x_1) \cdots g(x_n) \end{bmatrix} \begin{bmatrix} g(x_1) \\ \vdots \\ g(x_n) \end{bmatrix}$

so $K = x^T x$ where $x = \begin{bmatrix} g(x_1) \cdots g(x_n) \end{bmatrix}$

In HW3 we proved $a^T X^T X a$ will always be ≥ 0

$\therefore K$ is a PSD kernel

1c. If $\langle f, f \rangle_H \geq 0$, and we let.

and we let $f = K(X_f, \cdot)$

we have $\langle K(x_f, \cdot), K(x_f, \cdot) \rangle_H$

and therefore $K(x_f, x_f) \geq 0$

So $K(x, y)$ satisifies the positivity

1d. If $0 < K(x,x) < \infty, \ \forall x$

Let the inner product be represented by $\langle f, f \rangle_H$

We want to show if $\langle f, f \rangle_H = 0$, then $f(x) = 0, \ \forall x$

The reproducing property is: $|f(x)| = |\langle f, K(x, \cdot) \rangle| \leq \|f\| \sqrt{K(x,x)}$

so $|f(x)| \leq \langle f, f \rangle_H \sqrt{K(x,x)}$

Because $K(x,x) > 0$, $|f(x)| \leq 0$ if $\langle f, f \rangle_H = 0$

Because $|f(x)| \geq 0$ by definition, so if $\langle f, f \rangle_H = 0$ then:

$$|f(x)| \leq 0 \quad \text{and} \quad |f(x)| \geq 0$$

$$\text{so} \quad |f(x)| = f(x) = 0$$

So $K(x,y)$ satisties $\langle f, f \rangle_H = 0 \Rightarrow f(x) = 0, \ \forall x$

Training Error and Validation Error (respectively)

$2a.$

2*a*.

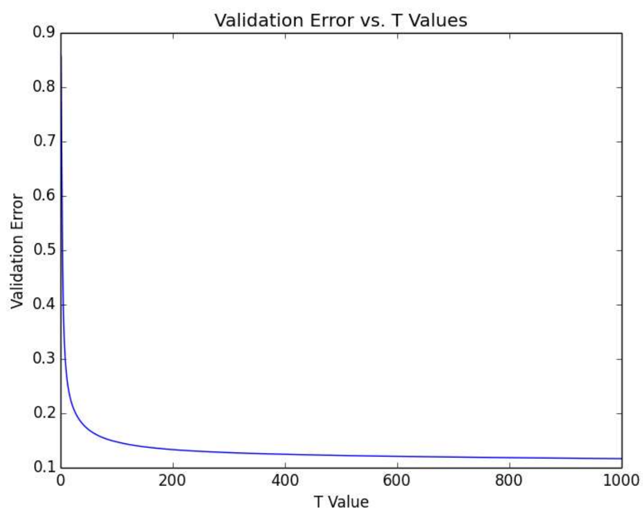
$$f(w) = \frac{1}{N}\sum_{p} \max\{0, 1 - y_p w^T x_p\} + \frac{\lambda}{2} w^T w$$

$$\frac{d}{dw} f(w) = \frac{1}{N}\sum_{p} \max\{0, -x_p^T y_p\} + \lambda w$$

2b. Code shown below

2c and 2d.
Eta values tried: 1e-01, 1e-02, 1e-03, 1e-04, 1e-05, 1e-06, 1e-07
Lambda values tried: 1e-01, 1e-02, 1e-03, 1e-04, 1e-05, 1e-06, 1e-07

Best eta value: 1e-06
Best lambda value: 1e-07
Best T: 1000



2e. Test error: 0.107528

3a.

$$f(a) = \frac{1}{N}\sum_{p} \max\{0, 1 - y_p((Ka)_p + b)\} + \frac{\lambda}{2} a^T K a$$

$$\frac{d}{da} f(a) = \frac{1}{N}\sum_{p} \max\{0, -K_p^T y_p\} + \lambda K a$$

3b. Code included below

3c and 3d.

Eta values tried: 1e-08, 1e-09, 1e-10, 1e-11, 1e-12, 1e-13, 1e-14
Lambda values tried: 1e-08, 1e-09, 1e-10, 1e-11, 1e-12, 1e-13, 1e-14

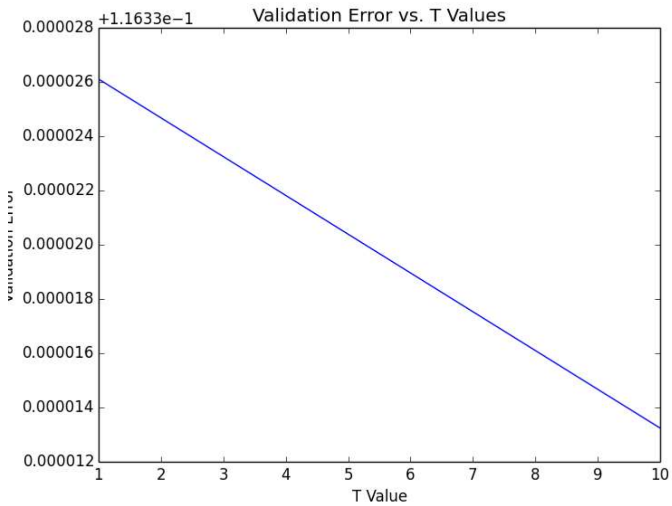
Out of all of the combinations attempted, the best results were:

Best eta value: 1e-14
Best lambda value: 1e-14
Best Training error: 0.116
Best Validation error: 0.116

Training Error and Validation Error (respectively)

				eta				
		1e-01	**1e-02**	**1e-03**	**1e-04**	**1e-05**	**1e-06**	**1e-07**
	1e-01	438.860	47.414	2.540	0.242	0.085	0.104	0.147
	1e-02	501.446	26.544	2.626	0.225	0.089	0.103	0.147
lambda	**1e-03**	210.441	35.640	2.913	0.233	0.087	0.104	0.147
	1e-04	285.370	37.176	2.843	0.234	0.084	0.104	0.147
	1e-05	427.453	39.242	2.889	0.234	0.085	0.104	0.147
	1e-06	460.390	24.646	2.639	0.234	0.085	0.104	0.147
	1e-07	311.453	24.646	2.639	0.234	0.085	0.104	0.147

				eta				
		1e-01	1e-02	1e-03	1e-04	1e-05	1e-06	1e-07
	1e-01	502.709	63.591	4.442	0.446	0.118	0.117	0.148
	1e-02	661.250	45.695	4.600	0.427	0.119	0.117	0.148
lambda	1e-03	394.044	55.187	4.919	0.438	0.120	0.117	0.148
	1e-04	484.868	56.881	4.844	0.440	0.118	0.117	0.148
	1e-05	620.746	58.969	4.894	0.440	0.118	0.117	0.148
	1e-06	648.228	44.747	4.628	0.440	0.118	0.117	0.148
	1e-07	511.779	44.747	4.628	0.440	0.118	0.117	0.148



CODE SECTION

LINEAR SVM

Setup code was the same as HW3, so did not include in this submission.

linsvm.py - contains the loss function and gradient for linear SVM

gradientdescent.py - contains the gradient descent algorithm for linear SVM

```python
import gradientdescent as gd
import numpy as np

def loss(X, Y, w, l, N):
    loss = 0
    for i in range(N):
        h = float(w.T * X[i].T * Y[i])
        if h < 1:
            loss += 1 - h

    loss *= 1.0 / N
    loss += (l / 2) * w.T * w

    return loss

def gradient(X, Y, w, eta, l, N):
    gsum = 0
    for i in range(N):
        h = float(w.T * X[i].T * Y[i])
        if h < 1:
            gsum += -X[i].T * Y[i]

lambda_values = [1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11]
eta_values = [1e-5, 1e-6, 1e-7, 1e-8, 1e-9, 1e-10, 1e-11]

lambda_values = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7]
eta_values = [1e-1, 1e-2, 1e-3, 1e-4, 1e-5, 1e-6, 1e-7]

gd.run_experiment(loss, gradient, eta_values, lambda_values)
```

```python
import numpy as np
import os
import os.path
import pickle
import matplotlib.pyplot as plt

import setup as s

(training_X, training_Y, training_N, val_X, val_Y, val_N, test_X, test_Y, test_N) = s.run()

def train_increment(gradient, w, eta, l):
    return gradient(training_X, training_Y, w, eta, l, training_N)

def train(gradient, T, eta, l):
    w = np.zeros((785,1))
    for t in range(0, T):
        w = train_increment(gradient, w, eta, l)

    return w

def train_err(loss, w, l):
    return float(loss(training_X, training_Y, w, l, training_N))

def val_err(loss, w, l):
    return float(loss(val_X, val_Y, w, l, val_N))

def test_err(loss, w, l):
    return float(loss(test_X, test_Y, w, l, test_N))

def run_experiment(loss, gradient, eta_values, lambda_values):
    T = 1000
    best_combo = (0, 0)
    lowest_err = float("inf")

    for l in lambda_values:
        for eta in eta_values:
            w = train(gradient, T, eta, l)

            tr_err = train_err(loss, w, l)
            v_err = val_err(loss, w, l)

            print "eta: " + str(eta)
            print "lambda: " + str(l)
            print "training error: " + str(tr_err)
            print "validation error: " + str(v_err)
            print

            if v_err < lowest_err:
                best_combo = (eta, l)
                lowest_err = v_err
```

```python

    best_T = 0
    lowest_err = float("inf")

    max_T = 1000

    T_values = range(1, max_T + 1)
    err_values = []

    w = np.zeros((785,1))

    best_eta = best_combo[0]
    best_lambda = best_combo[1]

    for T in T_values:
        w = train_increment(gradient, w, best_eta, best_lambda)

        err = val_err(loss, w, best_lambda)

        err_values.append(err)

        if err < lowest_err:
            best_T = T
            lowest_err = err


    w = train(gradient, best_T, best_eta, best_lambda)

    print "best T: " + str(best_T)
    print "best eta: " + str(best_eta)
    print "best lambda: " + str(best_lambda)
    print "test error: " + str(test_err(loss, w, best_lambda))

    plt.plot(T_values, err_values, '-')

    plt.title('Validation Error vs. T Values')
    plt.xlabel('T Value')
    plt.ylabel('Validation Error')

    plt.show()

```


KERNEL SVM

kernsvm.py - contains the loss function and gradient for kernel SVM

gradientdescent.py - contains the algorithm for gradient descent for kernel SVM

```python
import gradientdescent as gd
import numpy as np

def loss(K, Y, w, l, N):
    loss = 0
    for i in range(N):
        h = float(Y[i] * (K*w)[i])
        if h < 1:
            loss += 1 - h

    loss *= 1.0 / N
    loss += (l / 2) * w.T * K * w

    return loss

def gradient(K, Y, w, eta, l, N):
    gsum = 0
    for i in range(N):
        h = float(Y[i] * (K*w)[i])
        if h < 1:
            gsum += K[i].T * Y[i]

    return w - eta * ((1.0 / N) * gsum + l * K * w)

lambda_values = [1e-08, 1e-09, 1e-10, 1e-11, 1e-12, 1e-13, 1e-14]
eta_values = [1e-08, 1e-09, 1e-10, 1e-11, 1e-12, 1e-13, 1e-14]

gd.run_experiment(loss, gradient, eta_values, lambda_values)

```

```python
import numpy as np
import os
import os.path
import pickle
import matplotlib.pyplot as plt

import setup as s

(training_X, training_Y, training_N, val_X, val_Y, val_N, test_X, test_Y,
test_N) = s.run()

training_K = np.power((training_X * training_X.T + 1), 3)
val_K = np.power((training_X * val_X.T + 1), 3)
test_K = np.power((training_X * test_X.T + 1), 3)

training_KY = training_K * training_Y
val_KY = val_K * val_Y
test_KY = test_K * test_Y

print 'kernel matrices built'

def train_increment(gradient, w, eta, l):
    return gradient(training_K, training_Y, w, eta, l, training_N)

def train(gradient, T, eta, l):
    w = np.zeros((training_N,1))
    for t in range(0, T):
        w = train_increment(gradient, w, eta, l)

    return w

def train_err(loss, w, l):
    return float(loss(training_K, training_Y, w, l, training_N))

def val_err(loss, w, l):
    return float(loss(val_K, val_Y, w, l, val_N))

def test_err(loss, w, l):
    return float(loss(test_K, test_Y, w, l, test_N))

def run_experiment(loss, gradient, eta_values, lambda_values):
    T = 10
    best_combo = (0, 0)
    lowest_err = float("inf")

    for l in lambda_values:
        for eta in eta_values:
            w = train(gradient, T, eta, l)

            tr_err = train_err(loss, w, l)
```

```
            v_err = val_err(loss, w, l)

            print "eta: " + str(eta)
            print "lambda: " + str(l)
            print "training error: " + str(tr_err)
            print "validation error: " + str(v_err)
            print

            if v_err < lowest_err:
                best_combo = (eta, l)
                lowest_err = v_err

    best_T = 0
    lowest_err = float("inf")

    max_T = 10

    T_values = range(1, max_T + 1)
    err_values = []

    w = np.zeros((training_N, 1))

    best_eta = best_combo[0]
    best_lambda = best_combo[1]

    for T in T_values:
        w = train_increment(gradient, w, best_eta, best_lambda)

        err = val_err(loss, w, best_lambda)

        err_values.append(err)

        if err < lowest_err:
            best_T = T
            lowest_err = err


    w = train(gradient, best_T, best_eta, best_lambda)

    print "best T: " + str(best_T)
    print "best eta: " + str(best_eta)
    print "best lambda: " + str(best_lambda)
    #  print "test error: " + str(test_err(loss, w, best_lambda))

    plt.plot(T_values, err_values, '-')

    plt.title('Validation Error vs. T Values')
    plt.xlabel('T Value')
    plt.ylabel('Validation Error')

```

```
    plt.show()


```

3/3