Christopher Boyd
216 869 356

EECS 4404: A4 - SGD and SoftSVM

Stochastic Gradient Descent

1a)

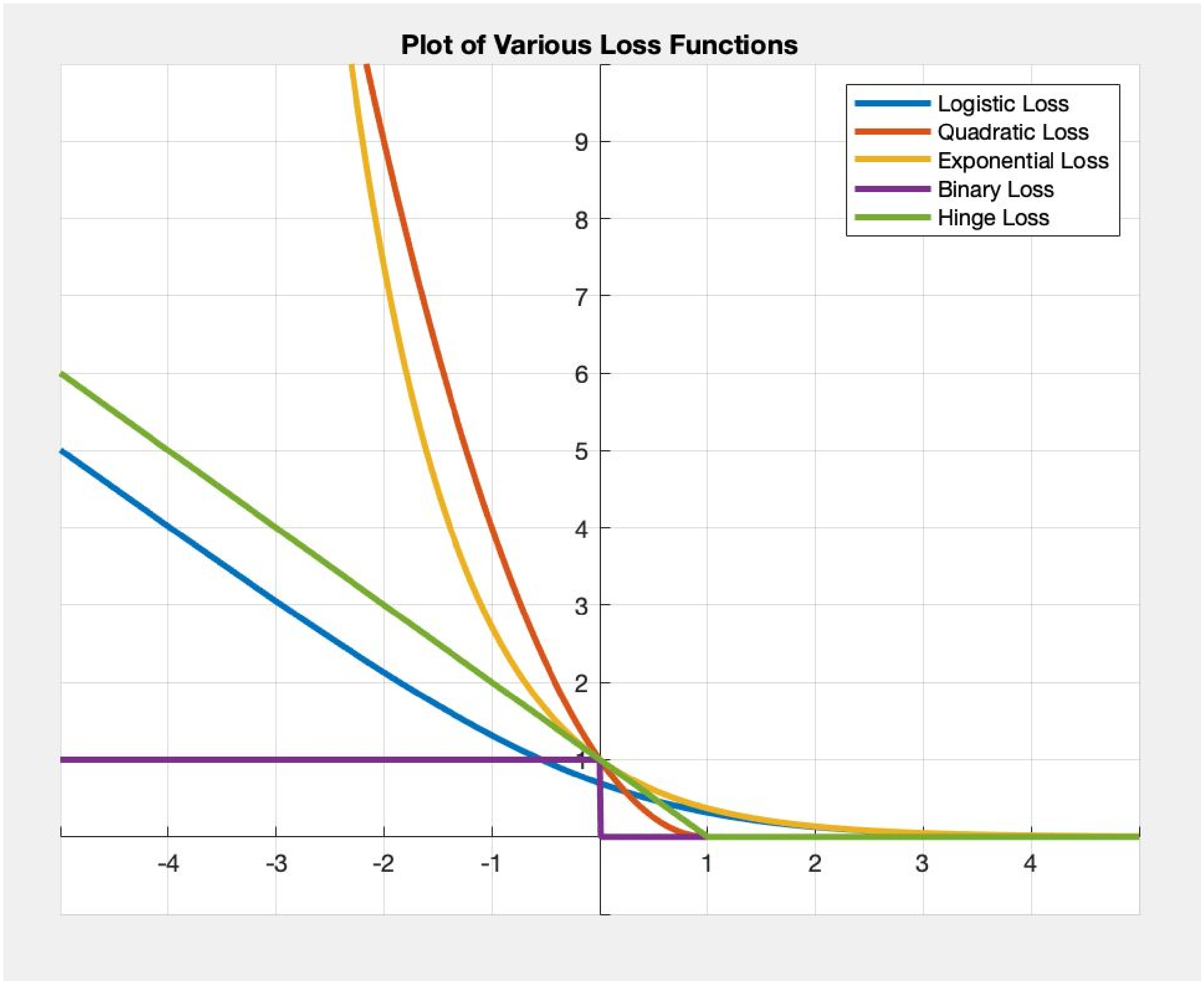


Fig 1. Comparison of Various Loss Functions

Of the presented loss functions, binary loss, stands apart. This is due to a prediction being either correct or incorrect. There is no distinction for the amount of correctness or incorrectness of a prediction. Compare this to all of the other loss functions (logistic, quadratic, exponential and hinge) where there is a sharp increase in loss as the level of incorrectness increases. Additionally, each of linear loss functions will show a small amount of loss even for correct predictions that fall close to the predictors hyperplane. This is seen in the small positive values for the graphs from 0 to ~1 for quadratic and hinge loss and from 0 ~ 2.5 for logistic and exponential losses. The linear loss functions differ in how quickly the loss increases as a prediction's incorrectness increases (distance from the hyperplane). Quadratic and exponential loss functions increase at a much faster rate than the hinge and logistic loss functions.

1b)

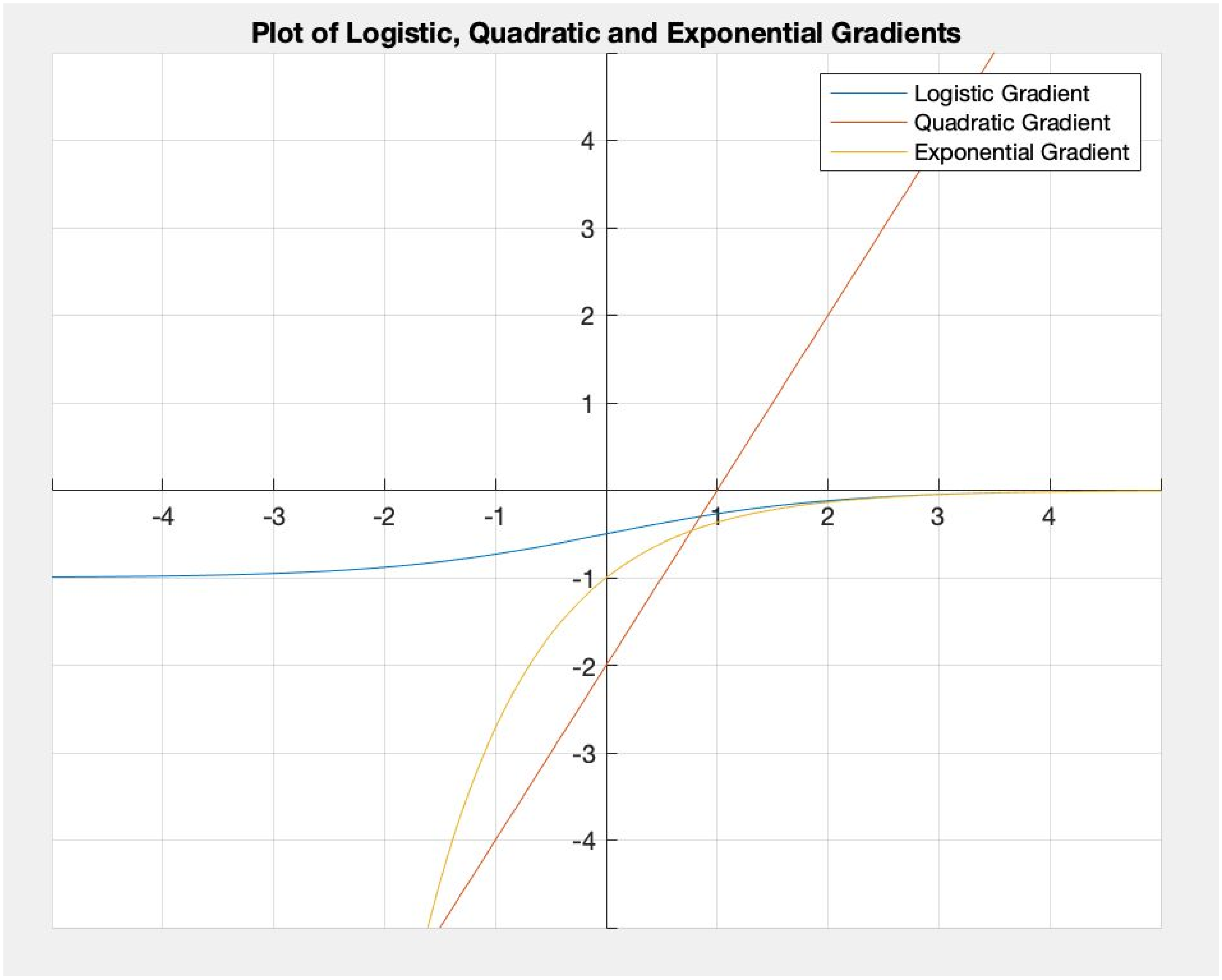


Fig 2. Loss Gradients for Logistic, Quadratic and Exponential Loss Functions

1c)

w_log = 2.5922, w_quad = 11.9027, w_exp = 74.4176

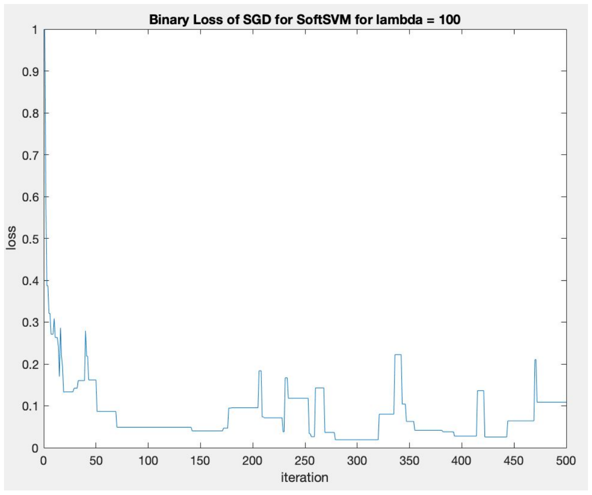
The magnitude of each update for the logistic and exponential gradients depends highly on t<w,x> as the functions have changing slopes. The quadratic gradient, being a straight line, does not depend on t<w,x> as it consistently updates the same amount due to the consistent slope.

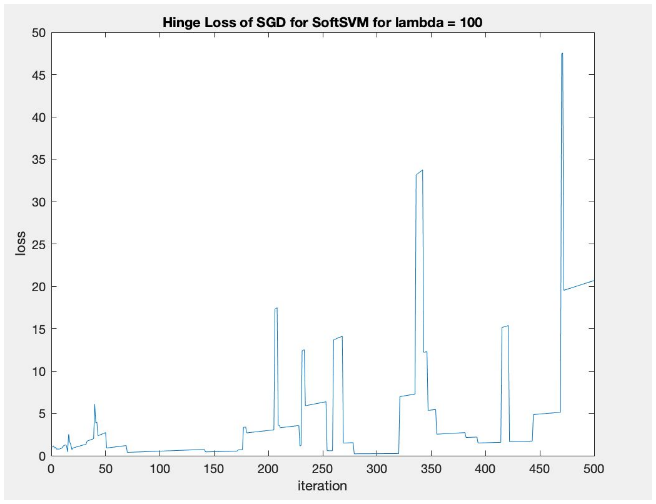
SoftSVM Optimization

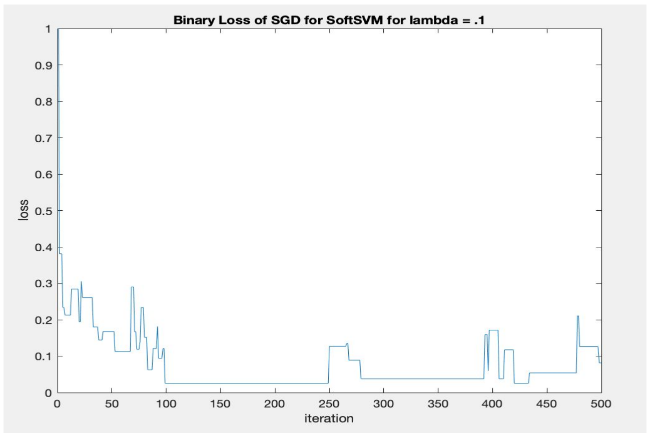
2a)

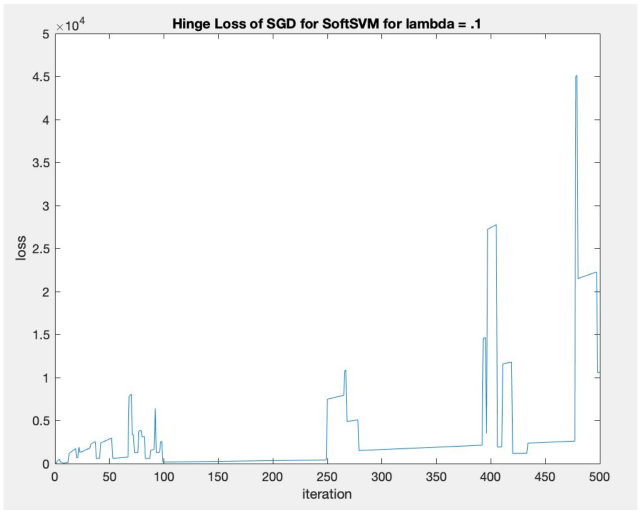
```
input = load("data_banknote_authentication.txt");
data = input;
data(:,5) = 1;
labels = input(:,5);
labels(labels(:)==0) = -1;

theta = zeros(1,5);
w = zeros(1,5);
T = 500;
lambda = .001;
i = 1;
b_loss = zeros(1,T);
h_loss = zeros(1,T);
for j = 1:T
    w = (1/lambda * j) * theta;
    i = ceil(rand * numel(labels));
    update = labels(i) * dot(w, data(i,:));
    if update < 1
        theta = theta + labels(i) * data(i,:);
    else
        theta = theta;
    end
    b_loss(j) = binary_loss(w,data,labels);
    h_loss(j) = logistic_loss(w,data,labels);
end
```

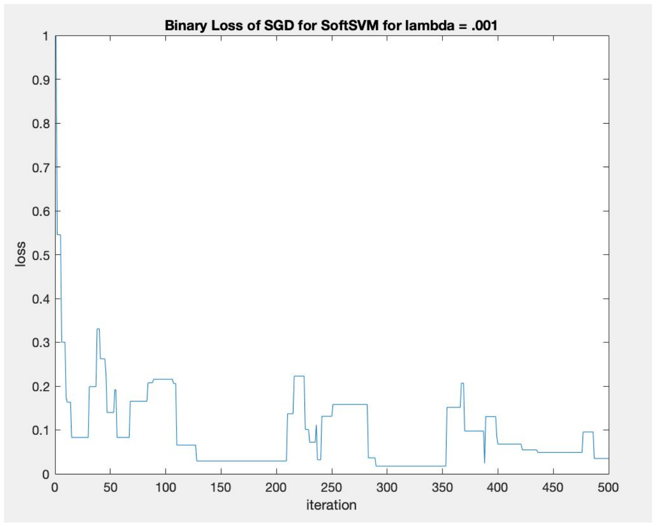
2b)

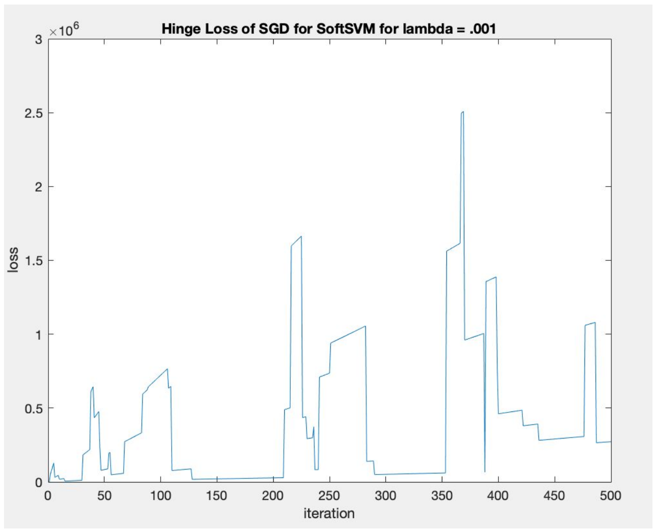












2c)
The curves are in no way monotone. I would not expect them to be, either. For each iteration the algorithm selects a random input/output pair from the data set that is used to update the weight vector. With these points being random, we would expect the errors at each point to be random as well.

What can be clearly seen is that the loss values drastically increase in line with the decreasing values used for λ. This follows from the 1 / λj factor in the algorithm (see Lecture 17, pg 20). As λ decreases, the update change grows dramatically.

To find a linear predictor of minimal loss I would select a high value for λ, 100 produces a consistent number of low binary losses. Run the SGD algorithm for several hundred iterations and select the earliest weight vector that produced the minimum binary loss (if multiple weight vectors produced the same minimum binary loss).

2d)
Binary loss for each of the “one-vs-all” classifiers using perceptron algorithm without normalized weight vectors:
loss_w1 = 0.3333, loss_w2 = 0.1429, loss_w3 = 0.3524
Binary loss for argmax <wi,x>: 0.4619
Binary loss for each of the “one-vs-all” classifiers using perceptron algorithm with normalized weight vectors:
loss_w1 = 0.3333, loss_w2 = 0.1429, loss_w3 = 0.3524
Binary loss for argmax <wi,x>: 0.4619

Normalizing the weight vector determined using the perceptron algorithm does not affect the binary loss when selecting the predictor label based on the max of the dot products of the three weight vectors and a given point. This follows from normalizing the weight vector not changing the hyperplane separating the one-vs-all classifiers trained. Normalizing the weight vectors does change the actual value of the dot product, but not the relationship between the three dot product values for the three one-vs-all classifiers.