Problem Set 1

Warm Up:

1. To prove that $f(x) = w * f_1(x)$ is a convex function, it must satisfy the inequality $\lambda f(x) + (1-\lambda)f(y) \geq f(\lambda x + (1-\lambda)y)$

   First, let's look at the left hand side and substitute: $f(x) = w * f_1(x)$:

   $$\lambda f(x) + (1-\lambda)f(y) = \lambda(w * f_1(x)) + (1-\lambda)(w * f_1(y)) = w(\lambda f_1(x) + (1-\lambda)f_1(y))$$

   Next, let's look at the right hand side and substitute $f(x) = w * f_1(x)$:

   $$f(\lambda x + (1-\lambda)y) = w * f_1(\lambda x + (1-\lambda)y)$$

   Substituting this back in, we get:

   $$w * (\lambda f_1(x) + (1-\lambda)f_1(y)) \geq w * f_1(\lambda x + (1-\lambda)y)$$

   Now, we know that $f_1(x)$ is a convex function, so we can say:

   $$\lambda f_1(x) + (1-\lambda)f(y) \geq f_1(\lambda x + (1-\lambda)y)$$

   Multiplying both sides by w, which doesn't affect the sign of the inequality as w ≥ 0, gives us:

   $$w * (\lambda f_1(x) + (1-\lambda)f_1(y)) \geq w * f_1(\lambda x + (1-\lambda)y)$$

   Which is the same as what we got by substituting $w * f_1(x)$ for $f(x)$. Thus, $f(x) = w * f_1(x)$ is a convex function.
2. To prove that $f(x) = f_1(x) + f_2(x)$ is a convex function, it must satisfy the inequality $\lambda f(x) + (1-\lambda)f(y) \geq f(\lambda x + (1-\lambda)y)$

   First, let's look at the left hand side and substitute: $f(x) = f_1(x) + f_2(x)$:

   $$\lambda f(x) + (1-\lambda)f(y) = \lambda(f_1(x) + f_2(x)) + (1-\lambda)(f_1(x) + f_2(x))$$

   Next, let's look at the right hand side and substitute $f(x) = f_1(x) + f_2(x)$

   $$f(\lambda x + (1-\lambda)y) = f_1(\lambda x + (1-\lambda)y) + f_2(\lambda x + (1-\lambda)y)$$

   Substituting this back in, we get:

   $$\lambda(f_1(x) + f_2(x)) + (1-\lambda)(f_1(x) + f_2(x)) \geq f_1(\lambda x + (1-\lambda)y) + f_2(\lambda x + (1-\lambda)y)$$

   Now, we know that $f_1(x)$ and $f_2(x)$ are convex functions, so we can say:

   $$\lambda f_1(x) + (1-\lambda)f_1(y) \geq f_1(\lambda x + (1-\lambda)y)$$
   $$\lambda f_2(x) + (1-\lambda)f_2(y) \geq f_2(\lambda x + (1-\lambda)y)$$

Next, we add the two inequalities to get

$$\lambda f_1(x) + (1-\lambda)f_1(y) + \lambda f_2(x) + (1-\lambda)f_2(y) \geq f_1(\lambda x + (1-\lambda)y) + f_2(\lambda x + (1-\lambda)y)$$

After simplifying, we get:

$$\lambda(f_1(x) + f_2(x)) + (1-\lambda)(f1_1(y) + f_2(y)) \geq f_1(\lambda x + (1-\lambda)y) + f_2(\lambda x + (1-\lambda)y)$$

Which is the same as what we got earlier by substitution. Thus, $f(x) = f_1(x) + f_2(x)$ is a convex function.

3. To prove that $f(x) = \max(f_1(x), f_2(x))$ is a convex function, it must satisfy the inequality $\lambda f(x) + (1-\lambda)f(y) \geq f(\lambda x + (1-\lambda)y)$

   Because of the max function, let's consider two cases:

   1. When $f_1(x) \geq f_2(x)$
   2. When $f_2(x) \geq f_1(x)$

   Case 1: $f_1(x) \geq f_2(x)$ Let's substitute $f(x) = \max(f_1(x), f_2(x))$ where $\max(f_1(x), f_2(x))$ = $f_1(x)$: $\lambda f(x) + (1-\lambda)f(y) \geq f(\lambda x + (1-\lambda)y)$ = $\lambda f_1(x) + (1-\lambda)f_1(y) \geq f_1(\lambda x + (1-\lambda)y)$

   This aligns with the standard inequality for convexity where for $f_1(x)$:

   $$\lambda f_1(x) + (1-\lambda)f_1(y) \geq f_1(\lambda x + (1-\lambda)y)$$

   Case 2: $f_2(x) \geq f_1(x)$ Let's substitute $f(x) = \max(f_1(x), f_2(x))$ where $\max(f_1(x), f_2(x))$ = $f_2(x)$: $\lambda f(x) + (1-\lambda)f(y) \geq f(\lambda x + (1-\lambda)y)$ = $\lambda f_2(x) + (1-\lambda)f_2(y) \geq f_2(\lambda x + (1-\lambda)y)$

   This aligns with the standard inequality for convexity where for $f_2(x)$:

   $$\lambda f_2(x) + (1-\lambda)f_2(y) \geq f_2(\lambda x + (1-\lambda)y)$$

   Since in both cases, the function is convex, $f(x) = \max(f_1(x), f_2(x))$ will be a convex function.

4. To prove that $f(x) = f_1(x)^2$ is a convex function, it must satisfy the inequality $\lambda f(x) + (1-\lambda)f(y) \geq f(\lambda x + (1-\lambda)y)$

   Let's look at the standard inequality and substitute: $f(x) = f_1(x)^2$:
   $\lambda f_1(x)^2 + (1-\lambda)f_1(y)^2 \geq f_1(\lambda x + (1-\lambda)y)^2$

   This is equal to $\lambda f_1(x)^2 + (1-\lambda)f_1(y)^2 - f_1(\lambda x + (1-\lambda)y)^2 \geq 0$

   This inequality will always hold true provided λ is greater than 1.

   Thus, $f(x) = f_1(x)^2$ is a convex function.

Problem 1:

1. We know that the linear term $a^T x + b$ is convex since a linear function is convex. Additionally, through the warmup, we know that the max of a convex function preserves convexity. So, the modified loss function is convex.

2.

```
Initialization:
- Choose initial values for a and b.
- Choose a step size parameter γ (learning rate).
- Set a stopping criteria (ex: max number of iterations).

Repeat until stopping criteria is met:
for i = 0 to numIterations:
        Compute the subgradient for parameter a:
                subgrad_a = 0
                for m in range(1, M + 1):
                        if aT · x[m] + b - y[m] > 0:
                                subgrad_a += y[m] * x[m]

        Compute the subgradient for parameter b:
                subgrad_b = 0
                for m in range(1, M + 1):
                        if aT · x[m] + b - y[m] > 0:
                                subgrad_b += y[m]

        Update parameters using subgradients:
        a = a + step_size * subgrad_a
        b = b + step_size * subgrad_b
```

One way you could pick the step size is initializing it to a very large value and then slowly decrementing it to a step size that makes the algorithm run much more efficiently.

3. A big advantage of using the modified loss function is that the max term helps us focus on the worst case scenarios, where the loss is the highest. It can be seen as evaluating the model's performance when predictions are furthest from their targets. This can be useful when one wants to improve performance of the model in less ideal situations. Another benefit of the modified loss function over the standard loss function is that the modified loss function may be less prone to overfitting as it maximizes the difference. This may make the model less inclined to attempt to fit the data as cleanly as possible, and it's possible that it may be less inclined to fit noisy data as well. One con of using the modified loss function is that using the max function may make calculations more difficult when optimizing. It will likely require more advanced techniques that use more resources and are harder to implement properly. Additionally, the usage of the max in the loss function may make it not differentiable at some points, which can make it difficult to calculate the gradients necessary.

Problem 2:

1. My algorithm took 32 iterations to find the perfect classifiers for the data.

For Iteration 1: w: [-235.78205425 30.65585373 -103.40801923 -28.63225313]

b: -548.0

For Iteration 2: w: [-345.74559137 38.2341255 -161.85289179 -38.96749417]

b: -322.0

For Iteration 3: w: [-440.38207866 40.1862109 -210.50276511 -46.30368614]

b: -115.0

Final weights: [-429.10854984 58.58415085 -208.97294734 -4.11165447]

Final bias: -155.0

2. My algorithm took 10680 iterations to find the perfect classifiers for the data.

   Iteration 1

   w: [0.29265951 0.51606929 -0.02156455 -0.14430998]

   b: -1.0

   Iteration 2

   w: [0.29265951 0.51606929 -0.02156455 -0.14430998]

   b: -1.0

   Iteration 3

   w: [-0.1257165 0.5883815 -0.26937881 0.55267775]

   b: 0.0

   Final weights: [-1.37673713e+01 1.98056397e+00 -6.79223890e+00 -7.84444800e-03]

   Final bias: -5.0

3. As the step size changes, the program may take more or less iterations to converge, or it may actually never converge. For example, when we use a decreasing step size of (1 / (1 + iteration)), the program actually never converges within a reasonable amount of steps. It likely requires millions of iterations to converge. However, when we use a constant step size that is different than 1, such as 2 or 5, the amount of iterations stays the same, but the values predicted by the program vary greatly.
4. The smallest two-dimensional data set where gradient descent with a step size of 1 may fail to converge is when the data points are perfectly separable into two classes, and the initial parameters are set such that the algorithm oscillates between values and never converges. The method may fail to converge more generally is the step size of 1 might be very large and may cause the algorithm to overshoot and fail to accurately predict the ideal values for w.

Problem 3:

1. a. We need to find a linear separator such that $w_1x_1 + w_2x_2 + b = 0$

   Plugging in the points we'll have: $-w_1 - w_2 + b = 1$ $w_1 + w_2 + b = 1$ $-w_1 + w_2 + b = -1$ $w_1 - w_2 + b = -1$

   Adding equations 1 and 2, we get $2b = 0$

   Adding equations 3 and 4, we get $2b = 0$

   We know b is 0, so let's substitute that into our original equations $-w_1 - w_2 = 1$ $w_1 + w_2 = 1$ $-w_1 + w_2 = -1$ $w_1 - w_2 = -1$

   After solving this system of equations, we get $w_1 = 0.5$ and $w_2 = 0.5$.

   So the linear separator that divides this is $0.5x_1 + 0.5x_2 = 0$

   b. We need to find a linear separator such that $w_1x_1^2 + w_2x_2^2 + b = 0$ Plugging in the points we'll have: $w_1^2 + w_2^2 + b = 1$ $w_1^2 + w_2^2 + b = 1$ $w_1^2 + w_2^2 + b = -1$ $w_1^2 + w_2^2 + b = -1$

   There is no solution to these equations as there are no values of w_1, w_2, b that will satisfy both conditions. Therefore, there is no linear separator.

   c. We need to find a linear separator such that $w_1x_1x_2 + b = 0$ Plugging in the points we'll have: $w_1 + b = 1$ $w_1 + b = 1$ $-w_1 + b = -1$ $-w_1 + b = -1$

   Let's add the first and third equations to get $2b = 0$. So, we know $b = 0$. Let's substitute that back in.

   $w_1 + 0 = 1$ -> $w_1 = 1$

   So, the linear separator for this data is $x_1x_2 = 0$

   d. We need to find a linear separator such that $w_1x_1sin(x_2) + b = 0$ Plugging in the points we'll have: $-w_1sin(-1) + b = 1$ $w_1sin(1) + b = 1$ $-w_1sin(1) + b = -1$ $w_1sin(-1) + b = -1$

   We have two unique equations: $-w_1sin(1) + b = 1$ $w_1sin(1) + b = 1$

   From this we know, $b = -1 + w1sin(1)$ or $b = 1 - w1sin(1)$

   From this, b can be either -1 or 1 depending on the value of w.
2. To perform polynomial regression using gradient descent, we first need to represent our data points as feature vectors and include all powers up to the kth power. For example, for 2 features, the feature

vector would be:

$$\Phi \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} 1 \\ x_1 \\ x_2 \\ x_1^2 \\ x_2^2 \end{bmatrix}$$

The larger your k value is, the more features you can accommodate, which can help greatly when the data is not linearly separable. The per iteration complexity of gradient descent is $O(m * n)$, where m is the degree of the polynomial and n is the number of data points you have to iterate through.

Problem 4:

1. The optimization problem I solved to find the perfect classifier is to find a hyperplane that best separates the two classes in the training data while maximizing the margin between them. The objective is to minimize the square of the Euclidean norm of w such that $y^{(i)}w^T x_i + b \geq 1$, for all i in the set.
2.

Weights:

```
[[ -0.69607841]
 [ 24.76674531]
 [ 40.15149569]
 [ 41.22081426]
 [  8.94285389]
 [ 11.53221129]
 [-16.33933404]
 [-10.26325461]
 [ -5.94665761]
 [-13.10014128]
 [-11.2854318 ]
 [-17.26401287]
 [-13.32157989]
 [  3.58849385]
 [-11.82224669]
 [  1.76748428]
 [  0.64142088]
 [-14.92656629]
 [ 25.14249461]
 [ 25.72386069]
 [ 26.33431068]
 [ 32.42237144]
 [ -4.07120256]
```

```
 [ 14.73918214]
 [ 14.71248721]
 [ 18.37504251]
 [-31.59987655]
 [-24.63605925]
 [ -1.06260087]
 [  2.70821962]]
```

```
Bias: 28.511

Optimal Margin: 0.0095

Support Vectors: [ 26 71 101 345 825 843 886]
```

3. Attached to submission