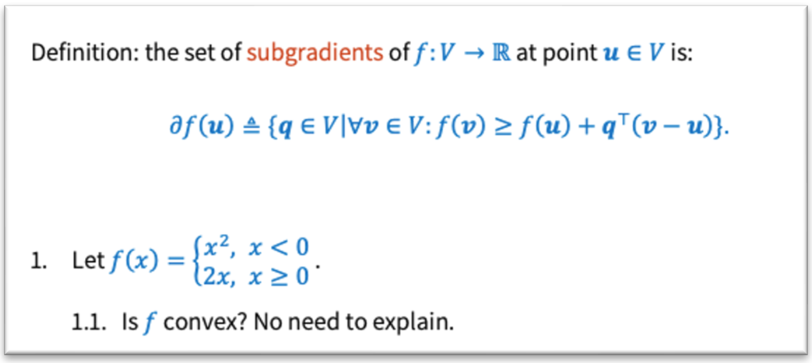
Short HW4 – Optimization, Regression, and Boosting

Part A – Optimization



Definition: the set of subgradients of $f: V \to \mathbb{R}$ at point $\boldsymbol{u} \in V$ is:

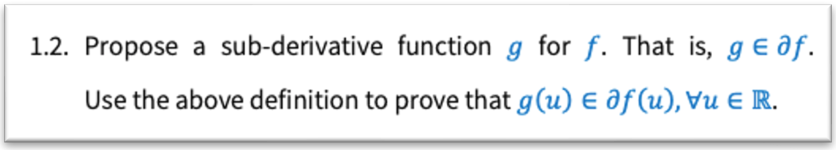
$$\partial f(\boldsymbol{u}) \triangleq \{\boldsymbol{q} \in V \mid \forall \boldsymbol{v} \in V: f(\boldsymbol{v}) \geq f(\boldsymbol{u}) + \boldsymbol{q}^\top(\boldsymbol{v} - \boldsymbol{u})\}.$$

1. Let $f(x) = \begin{cases} x^2, & x < 0 \\ 2x, & x \geq 0 \end{cases}$.

   1.1. Is f convex? No need to explain.

Answer 1.1:

f is convex.



1.2. Propose a sub-derivative function g for f. That is, $g \in \partial f$.

Use the above definition to prove that $g(u) \in \partial f(u), \forall u \in \mathbb{R}$.

Answer 1.2:

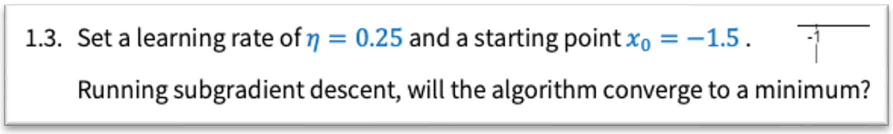
Let $f(x) = \begin{cases} x^2, & x < 0 \\ 2x, & x \geq 0 \end{cases}$ and suppose $g(x) \in \partial f(x) = \begin{cases} 2x, & x < 0 \\ 2, & x \geq 0 \end{cases}$.

We want to show that $\forall u, v \in \mathbb{R}$, $f(v) \geq f(u) + g(u)^T(v - u)$.

Notice that f(x) is convex, hence the tangent line $\forall v \in \mathbb{R}$ is under the function, explicitly, denote t(x) as the tangent line for some fixed point $u \in \mathbb{R}$ then $f(v) \geq t(v)$ $\forall v \in \mathbb{R}$.

Thus, let $u \in \mathbb{R}$, the tangent line is given by $t(x) = f(u) + f'(u)(x - u)$.

Now, using convexity of f, $\forall v \in \mathbb{R}: f(v) \geq t(v) = f(u) + g(u)(v - u)$.



1.3. Set a learning rate of $\eta = 0.25$ and a starting point $x_0 = -1.5$.

Running subgradient descent, will the algorithm converge to a minimum?

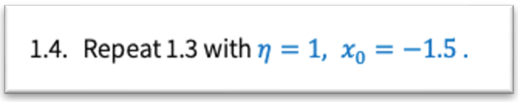
Answer 1.3:

i	$x_i = x_{i-1} - \frac{\nabla f(x_{i-1})}{4}$	$f(x_i)$	$\frac{\partial}{\partial x} f(x_i) = g(x_i)$
0	-1.5	2.25	-3
1	-0.75	0.5625	-1.5
2	-0.375	0.140625	-0.75
3	-0.1875	0.03515625	−0.375
4	-0.09375	0.0087890625	−0.1875
5	-0.046875	0.002197265625	−0.09375
6	-0.0234375	0.00054931640625	−0.046875
7	-0.01171875	0.000137329101563	−0.0234375

.
.
.

538	-8.3353e-163	0	-3.33416e-162

Exploring the function we can determine that the function f(x) has a minimum point at (0,0), so convergence will look like getting close to $x = 0 , y = 0$.
Looking at the x_i and $f(x_i)$values can tell the gradient descent algorithm goes to the expected direction although we can see the gradient values are decreasing as well, leading to a small step in each iteration. Thus convergence will be indeed achieved but after great amount of iterations.

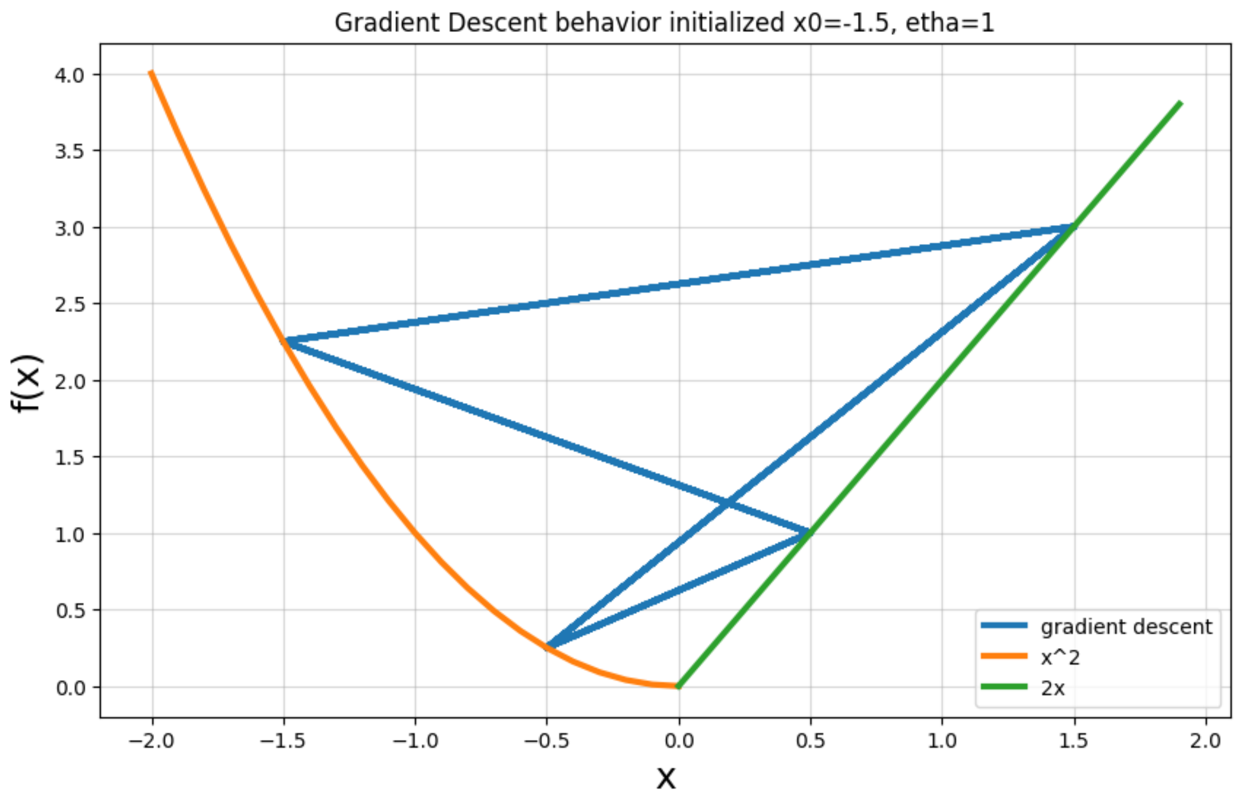


1.4. Repeat 1.3 with $\eta = 1,\ x_0 = -1.5$.

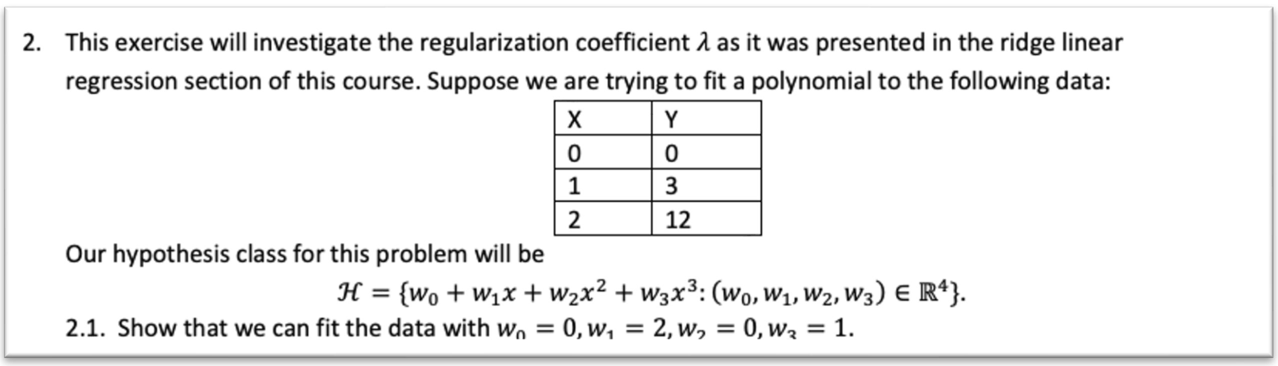
Answer 1.4:

With the initialization $x_0 = -1.5,\ \eta = 1$ the GD algorithm does not converge to the local minimum. This is due to a large step size, yielding a circular loop.

i	$x_i = x_{i-1} - \nabla f(x_{i-1})$	$f(x_i)$	$\frac{\partial}{\partial x} f(x_i) = g(x_i)$
0	-1.5	2.25	-3
1	1.5	3	2
2	-0.5	0.25	-1
3	0.5	1	2
4	-1.5	2.25	-3
5	1.5	3	2
6	-0.5	0.25	-1



Part B – Regression



2. This exercise will investigate the regularization coefficient λ as it was presented in the ridge linear regression section of this course. Suppose we are trying to fit a polynomial to the following data:

X	Y
0	0
1	3
2	12

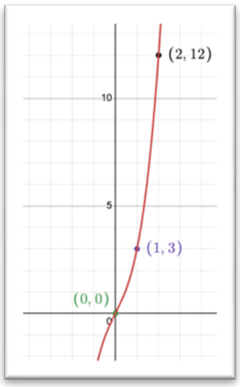
Our hypothesis class for this problem will be

$$\mathcal{H} = \{w_0 + w_1 x + w_2 x^2 + w_3 x^3 : (w_0, w_1, w_2, w_3) \in \mathbb{R}^4\}.$$

2.1. Show that we can fit the data with $w_0 = 0, w_1 = 2, w_2 = 0, w_3 = 1$.

Answer 2.1:

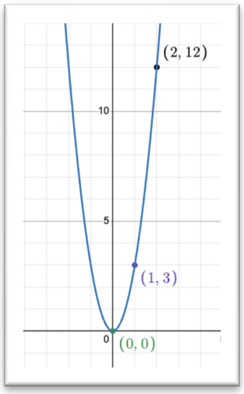
Given $h(x) = 0 + 2x + 0x^2 + x^3 \in \mathcal{H}$, it can be seen that:
$h(0) = 0\,, h(1) = 3, h(2) = 12$
So the given model can predict with 100% accuracy all the given data and thus fits it.



2.2. Show that our hypothesis class is too expressive for the problem we're dealing with. In other words, find a simple quadratic polynomial that fits the data perfectly.

Answer 2.2:

Let $w = (0,0,3) \in \mathbb{R}^3$ defining $h(x) = 3x^2$.
Notice that $h(0) = 0$, $h(1) = 3, h(2) = 12$
So the data can be fitted by a simple quadratic polynomial, hence
$\mathcal{H}$ cubic hypothesis class is it too expressive.



2.3. Denote the mean squared error (MSE)

$$\mathcal{L}(w) = \frac{1}{m}\|Xw - y\|_2^2,$$

Where X is the appropriate Vandermonde matrix.
Calculate $\mathcal{L}(w)$ for the quadratic model in (2.2) and the cubic model in (2.1).

Answer 2.3:

$$X_{cubic} = \begin{bmatrix} 0^0 & 0^1 & 0^2 & 0^3 \\ 1^0 & 1^1 & 1^2 & 1^3 \\ 2^0 & 2^1 & 2^2 & 2^3 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \end{bmatrix} \quad X_{quadradic} = \begin{bmatrix} 0^0 & 0^1 & 0^2 \\ 1^0 & 1^1 & 1^2 \\ 2^0 & 2^1 & 2^2 \end{bmatrix} = \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix}$$

$$y = \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix} \quad , \quad m = 3$$

$$w_{cubic} = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 1 \end{bmatrix} \quad , \quad w_{quadradic} = \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix}$$

$$\mathcal{L}(w_{cubic}) = \frac{1}{3}\left\| \begin{bmatrix} 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 \\ 1 & 2 & 4 & 8 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 2 \\ 0 \\ 1 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix} \right\|_2^2 = \frac{1}{3}\left\| \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix} \right\|_2^2 = 0$$

$$\mathcal{L}(w_{quadradic}) = \frac{1}{3}\left\| \begin{bmatrix} 1 & 0 & 0 \\ 1 & 1 & 1 \\ 1 & 2 & 4 \end{bmatrix} \cdot \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix} \right\|_2^2 = \frac{1}{3}\left\| \begin{bmatrix} 0 \\ 1 \\ 4 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix} \right\|_2^2 = \frac{1}{3}\left\| \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix} \right\|_2^2 = 0$$

2.4. The best <u>line</u> for fitting the data is $y = 6x - 1$. Calculate $\mathcal{L}(w)$ for this line.

<u>Answer 2.4:</u>

Let $w = (-1,6) \in \mathbb{R}^2$ defining $h(x) = -1 + 6x$.

$$X_{linear} = \begin{bmatrix} 0^0 & 0^1 \\ 1^0 & 1^1 \\ 2^0 & 2^1 \end{bmatrix} = \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix}$$

$$w_{linear} = \begin{bmatrix} -1 \\ 6 \end{bmatrix} \quad , \; y = \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix}, \qquad m = 3$$

$$\mathcal{L}(w_{linear}) = \frac{1}{3}\left\| \begin{bmatrix} 1 & 0 \\ 1 & 1 \\ 1 & 2 \end{bmatrix} \cdot \begin{bmatrix} -1 \\ 6 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix} \right\|_2^2 = \frac{1}{3}\left\| \begin{bmatrix} -1 \\ 5 \\ 11 \end{bmatrix} - \begin{bmatrix} 0 \\ 3 \\ 12 \end{bmatrix} \right\|_2^2 = \frac{1}{3}\left\| \begin{bmatrix} -1 \\ 2 \\ -1 \end{bmatrix} \right\|_2^2 = \frac{6}{3} = 2$$

2.5. Now denote the MSE with regularization as show in class

$$\mathcal{L}_\lambda(w) = \frac{1}{m}\|Xw - y\|_2^2 + \lambda\|w\|_2^2.$$

Here $\lambda > 0$ is a hyperparameter, which is not given. As we learned in class, the regularization imposes a "cost" on models with large coefficients. Calculate $\mathcal{L}_\lambda(w)$ for each of the three models in (2.1), (2.2) and (2.4).

<u>Answer 2.5:</u>

$$\|w_{cubic}\|_2^2 = 5 \quad \mathcal{L}_\lambda(w_{cubic}) = 5 \cdot \lambda$$

$$\left\|w_{quadradic}\right\|_2^2 = 9 \quad \mathcal{L}_\lambda\left(w_{quadradic}\right) = 9 \cdot \lambda$$

$$\|w_{linear}\|_2^2 = 37 \quad \mathcal{L}_\lambda(w_{linear}) = 2 + 37 \cdot \lambda$$

2.6. As it turns out, $\mathcal{L}_\lambda(w)$ would never prefer the simple quadratic polynomial over the cubic polynomial we found, no matter the value of $\lambda > 0$. Can you explain why?

<u>Answer 2.6:</u>

Since the first argument is zero for both polynomial, the MSE with regularization is explicitly minimizes $\|w\|_2^2$. Note that $\forall\lambda > 0$, $\mathcal{L}_\lambda(w_{cubic}) = 5\lambda < 9\lambda = \mathcal{L}_\lambda\left(w_{quadradic}\right)$
Hence, for any $\lambda > 0$ the minimizer will be $\mathcal{L}_\lambda(w_{cubic})$.

2.7. Suggest a way to fix the regularization method to prefer the model we consider to be simpler.

Answer 2.7:

We could use the MSE with lasso regularization instead, which differs by the norm of the regulator.
Then, the objective is given by $\mathcal{L}_\lambda(w) = \frac{1}{m}\|Xw - y\|_2^2 + \lambda\|w\|_1$
It will produce the following objectives:

$$\|w_{cubic}\|_1 = 3 \quad \mathcal{L}_\lambda(w_{cubic}) = 3 \cdot \lambda$$

$$\left\|w_{quadradic}\right\|_1 = 3 \quad \mathcal{L}_\lambda\left(w_{quadradic}\right) = 3 \cdot \lambda$$

Note the MSE remained unchanged

So we get the same objective for both functions. Although there is an advantage for the quadradic polynomial.
If we look at the intersection between the range Lasso creates and a two-dimensional surface, we get a rhombus. We saw in the tutorial that an optimal solution of convex groups (such as the image of f) will be obtained close to the corners of a rhombus. We also saw that if the optimal solution is exactly on the corner, all the entries will zero out but the corner and if the solution is close tot the corner only part of the entries will zero out. Thus we can infer that $w_{quadradic}$ is a better solution for the lasso regression with higher accuracy to be obtained when the GD algorithm runs to optimize the MSE with lasso regression for this particular data set.

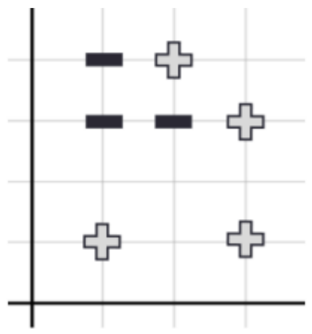
$$w_{cubic} = \begin{bmatrix} 0 \\ 2 \\ 0 \\ 1 \end{bmatrix} \qquad w_{quadradic} = \begin{bmatrix} 0 \\ 0 \\ 3 \end{bmatrix}$$

Part C - Boosting

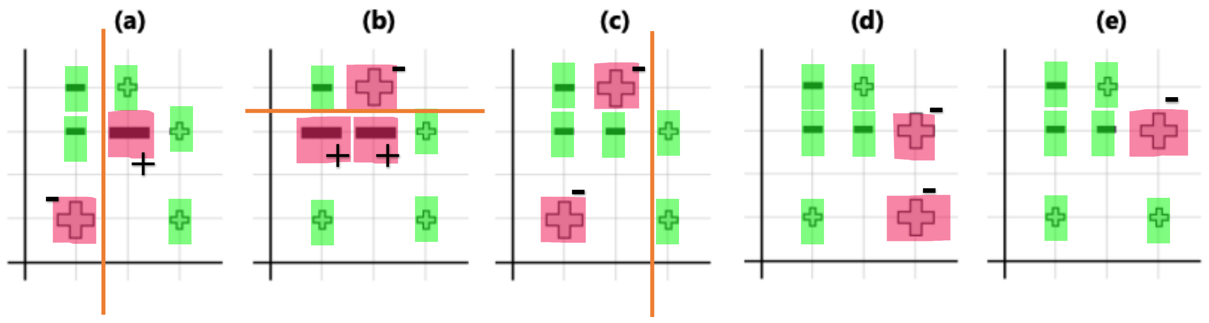
Part C – Boosting

3. Given the following data with binary labels (“+”, “-”).

   We run AdaBoost with Decision stumps as weak classifiers.
   The sizes of the shapes in the figures indicate the probabilities that the algorithm assigns to each sample (high probability = large shape). Initially, the algorithm starts from a uniform distribution.



   Only some of the following figures depict possible distributions that can be obtained after one iteration of AdaBoost. **Which ones?** For each such distribution, propose a weak classifier that can lead to its figure (use a clear drawing or a short description of that classifier).



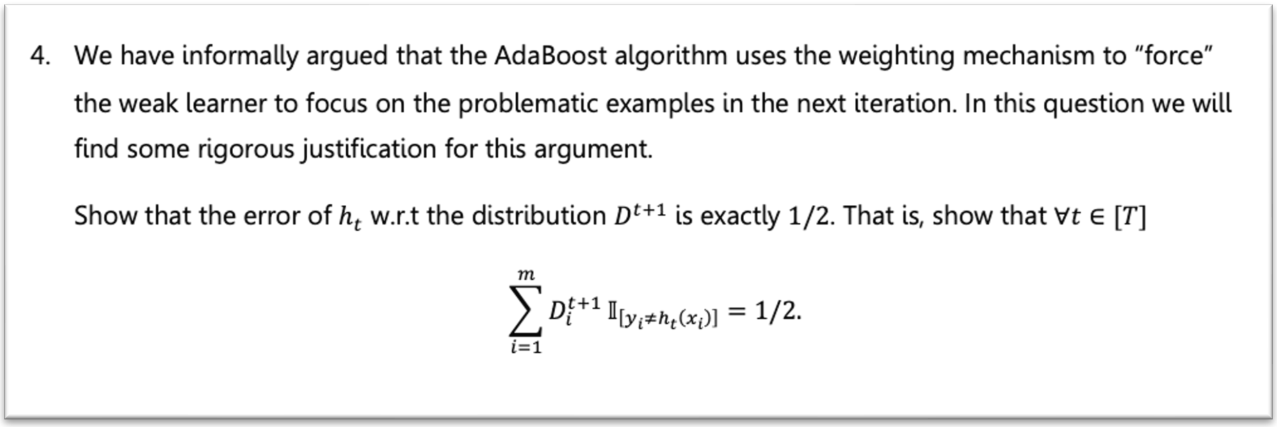
Answer 3:

Shortly, (a) and (c) are the only possibilities.
As we seen in the tutorial, the AdaBoost algorithm initialize all the data points with uniform probability. The key feature of AdaBoost is the greedy improvement of the weak learner in each iteration, meaning AdaBoost will update the probability of the data points that were classified correctly to be lower and the probability of the data points that were misclassified to be higher. That way of the next iteration the weak learner will have more weight to fit the misclassified dots. Using that rule and looking at the diagrams we can mark the dots as follows:
Green: Classified correctly by the weak learner (smaller)
Red: misclassified by the weak learner (bigger)
If red, what classification it got and if green it got the same classification as it is.
Now we will remember AdaBoost uses stamp trees, yielding vertical/horizontal separation of the 2-d space.
First we will see which distribution can be separated after one iteration:

(a) **Can be** separated after one iteration, the classifier creates the orange vertical line, right to it classified + and left to it classified -.
(b) **Can be** separated after one iteration, the classifier creates the orange horizontal line, above it classified - and below it classified +.
(c) **Can be** separated after one iteration, the classifier creates the orange vertical line, right to it classified + and left to it classified -.

(d) **Cannot be** separated using one iteration of AdaBoost.
(e) **Cannot be** separated using one iteration of AdaBoost.

Now, we need to consider that the weak learner is chosen by an ERM algorithm A. Hence, the possible distribution that might be obtained is the one with the lowest error.
Notice that $\epsilon_a = \epsilon_c = \frac{2}{7}$, $\epsilon_b = \frac{3}{7}$. Thus the weak learner at distribution b will never be chosen since there are better classifiers, although a and c have evenly likelihood to be chosen.
All together, A and C are the only options.



4. We have informally argued that the AdaBoost algorithm uses the weighting mechanism to "force" the weak learner to focus on the problematic examples in the next iteration. In this question we will find some rigorous justification for this argument.

Show that the error of h_t w.r.t the distribution D^{t+1} is exactly 1/2. That is, show that $\forall t \in [T]$

$$\sum_{i=1}^{m} D_i^{t+1}\, \mathbb{I}_{[y_i \neq h_t(x_i)]} = 1/2.$$

Answer 4:

Recall from the tutorial : (for my use)

$$(1)\ D_i^{t+1} = \frac{D_i^t \cdot e^{-\alpha_t y_i h_t(x_i)}}{z_t} = \frac{D_i^t \cdot e^{-\alpha_t y_i h_t(x_i)}}{\sum_{i=1}^{m} D_i^t \cdot e^{-\alpha_t y_i h_t(x_i)}}$$

$$(2)\ z_t = \sum_{i=1}^{m} D_i^t \cdot e^{-\alpha_t y_i h_t(x_i)}$$

$$(3)\ \alpha_t = \frac{1}{2}\log\left(\frac{1}{\epsilon_t} - 1\right) = \frac{1}{2}\log\left(\frac{1-\epsilon_t}{\epsilon_t}\right) \quad => \ e^{2\alpha_t} = \frac{1-\epsilon_t}{\epsilon_t}$$

$$(4)\ \epsilon_t = \sum_{i=1}^{m} D_i^t \cdot \mathbb{I}_{[y_i \neq h_t(x_i)]}$$

Another way to determine ϵ_t is by the misclassified points : $\epsilon_t = \frac{\sum_{[y_i \neq h_t(x_i)]} D_i^t}{\sum_{[y_i \neq h_t(x_i)]} D_i^t + \sum_{[y_i = h_t(x_i)]} D_i^t}$

Deriving from the above equation:

$$\epsilon_t \left(\sum_{[y_i \neq h_t(x_i)]} D_i^t + \sum_{[y_i = h_t(x_i)]} D_i^t \right) = \sum_{[y_i \neq h_t(x_i)]} D_i^t$$

$$\epsilon_t \cdot \sum_{[y_i = h_t(x_i)]} D_i^t = \sum_{[y_i \neq h_t(x_i)]} D_i^t - \epsilon_t \cdot \sum_{[y_i \neq h_t(x_i)]} D_i^t$$

$$(5)\quad \sum_{[y_i = h_t(x_i)]} D_i^t = \left(\frac{1-\epsilon_t}{\epsilon_t}\right) \sum_{[y_i \neq h_t(x_i)]} D_i^t$$

All together one has:

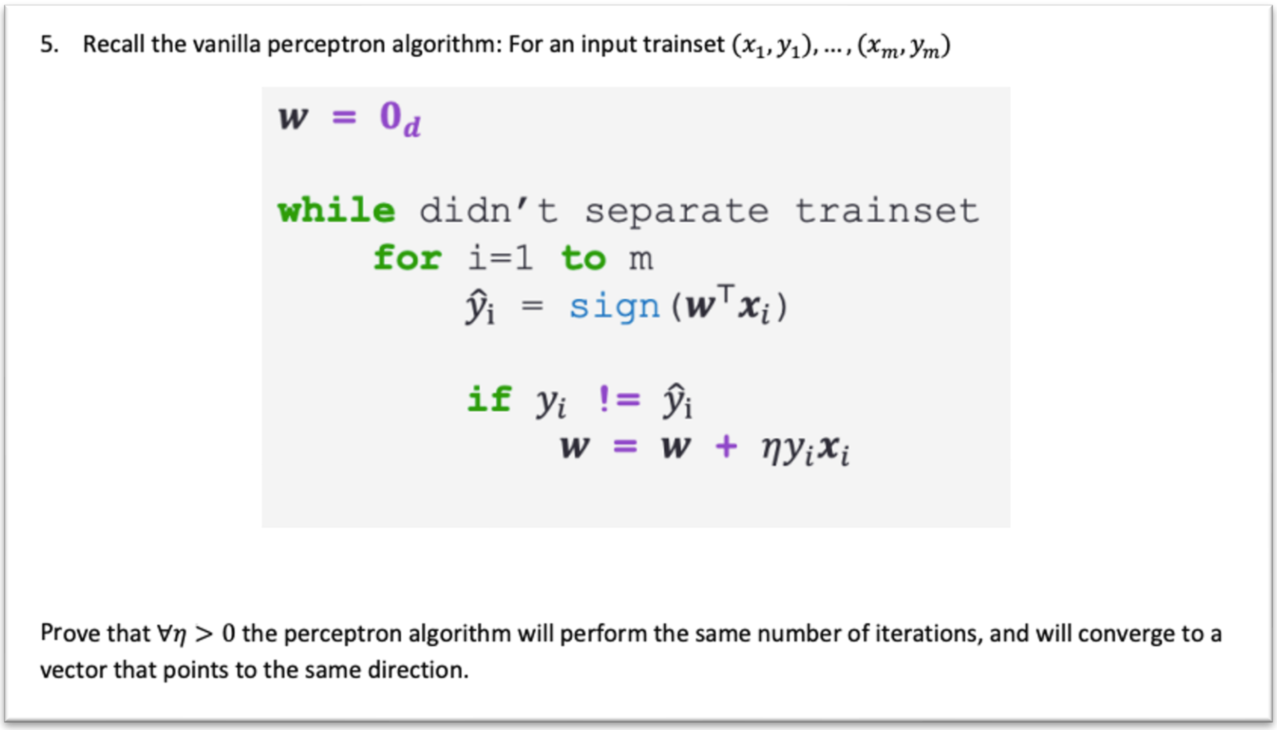
$$\sum_{i=1}^{m} D_i^{t+1} \cdot \mathbb{1}_{[y_i \neq h_t(x_i)]} \overset{(1)}{\cong} \sum_{[y_i \neq h_t(x_i)]} \frac{D_i^t \cdot e^{-\alpha_t y_i h_t(x_i)}}{z_t} + \sum_{[y_i = h_t(x_i)]} \frac{D_i^t \cdot e^{-\alpha_t y_i h_t(x_i)}}{z_t}$$

$$= \sum_{[y_i \neq h_t(x_i)]} \frac{D_i^t \cdot e^{-\alpha_t \cdot (-1)}}{z_t} = \sum_{[y_i \neq h_t(x_i)]} \frac{D_i^t \cdot e^{\alpha_t}}{z_t}$$

$$= \sum_{[y_i \neq h_t(x_i)]} \frac{D_i^t}{z_t \cdot e^{-\alpha_t}} \overset{(2)}{\cong} \frac{\sum_{[y_i \neq h_t(x_i)]} D_i^t}{\left(\sum_{[y_i \neq h_t(x_i)]} D_i^t \cdot e^{\alpha_t} + \sum_{[y_i = h_t(x_i)]} D_i^t \cdot e^{-\alpha_t}\right) \cdot e^{-\alpha_t}}$$

$$= \frac{\sum_{[y_i \neq h_t(x_i)]} D_i^t}{\sum_{[y_i \neq h_t(x_i)]} D_i^t + \sum_{[y_i = h_t(x_i)]} D_i^t \cdot e^{-2\alpha_t}} \overset{(5)}{\cong} \frac{\sum_{[y_i \neq h_t(x_i)]} D_i^t}{\sum_{[y_i \neq h_t(x_i)]} D_i^t + \left(\frac{1-\epsilon_t}{\epsilon_t}\right) \sum_{[y_i \neq h_t(x_i)]} D_i^t \, e^{-2\alpha_t}}$$

$$= \frac{1}{1 + \left(\frac{1-\epsilon_t}{\epsilon_t}\right) \cdot e^{-2\alpha_t}} \overset{(3)}{\cong} \frac{1}{1 + \left(\frac{1-\epsilon_t}{\epsilon_t}\right) \cdot \frac{\epsilon_t}{1-\epsilon_t}} = \frac{1}{2}$$



5. Recall the vanilla perceptron algorithm: For an input trainset $(x_1, y_1), \dots, (x_m, y_m)$

```
w = 0_d

while didn't separate trainset
    for i=1 to m
        ŷ_i = sign(w^T x_i)

        if y_i != ŷ_i
            w = w + η y_i x_i
```

Prove that $\forall \eta > 0$ the perceptron algorithm will perform the same number of iterations, and will converge to a vector that points to the same direction.

Answer 5:

First, we will show that regardless of η, the updated w will be on the same direction.
Notice that $y_i x_i \in \{-1,1\}, \eta \in \mathbb{R}$ thus $\eta y_i x_i = \pm(\eta, \dots, \eta) \in \mathbb{R}^d$ and the update is simply adding the same scalar to all of the entries of the current w. η only determines how far to move in the direction of $y_i x_i$ although the direction of the updated w doesn't change.
More formally: Let $\eta_1 \neq \eta_2 > 0$ and let (x_i, y_i) a misclassified point.
Let $w_1 = w + \eta_1 y_i x_i \quad , w_2 = w + \eta_2 y_i x_i$.
Notice that $w_2 - w_1 = (\eta_2 - \eta_1) y_i x_i$
This means that the difference is a scalar multiple, meaning that the difference is on the same line as w_1 and w_2. Thus, w_1 and w_2 are collinear (point to the same direction).

Secondly, we will show that algorithm will converge with the same number of iterations.
As we seen, different η's, produce w in the same direction (not necessary with the same size).
Since the sign of the margin depends only on the direction ($sign(w^T x_i)$), different η's still lead to the same prediction. Thus, it will take the same number of iterations to converge.

Lastly, since each iteration's direction doesn't depend on η, by simple induction also the last iteration direction do not depend on η.

All together, $\forall \eta > 0$, the perceptron will converge to the same direction with the same number of iterations.