Algorithms Required For The Midterm Exam

- Sample Mean:

$$\bar{x} = \frac{\sum_{i=1}^{n} x_i}{n} \text{ and } \bar{y} = \frac{\sum_{i=1}^{n} y_i}{n}$$

where n is the number of points in the dataset, x_i are the x values from each point in the dataset, and y_i are the y values from each point in the dataset.

- Biased Sample Variance:

$$s^2_{x,\text{biased}} = \text{var}_{x,\text{biased}} = \boxed{\frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2} \text{ and } s^2_{y,\text{biased}} = \text{var}_{y,\text{biased}} = \boxed{\frac{1}{n}\sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2}$$

where $\bar{x}$ is the Sample Mean of the x's, $\bar{y}$ is the Sample Mean of the y's, n is the number of points in the dataset, x_i are the x values from each point in the dataset, and y_i are the y values from each point in the dataset.

We could also find the Sum of Squared Differences first and then divide by n to find variance:

$$\text{SSD}_x = \sum_{i=1}^{n}\left(x_i - \bar{x}\right)^2 \text{ and } \text{SSD}_y = \sum_{i=1}^{n}\left(y_i - \bar{y}\right)^2$$

$$s^2_{x,\text{biased}} = \text{var}_{x,\text{biased}} = \frac{1}{n}\text{SSD}_x \text{ and } s^2_{y,\text{biased}} = \text{var}_{y,\text{biased}} = \frac{1}{n}\text{SSD}_y$$

- Biased Sample Covariance:

$$\text{cov}_{(x,y),\text{biased}} = \frac{1}{n}\sum_{i=1}^{n}\left(x_i - \bar{x}\right)\left(y_i - \bar{y}\right)$$

where $\bar{x}$ is the Sample Mean of the x's, $\bar{y}$ is the Sample Mean of the y's, n is the number of points in the dataset, x_i are the x values from each point in the dataset, and y_i are the y values from each point in the dataset.

We could also find the Sum of Product of Differences first and then divide by n to find the covariance between the x's and the y's:

$$\text{SPD}_{x,y} = \sum_{i=1}^{n}\left(x_i - \bar{x}\right)\left(y_i - \bar{y}\right)$$

$$\text{cov}_{(x,y),\text{biased}} = \frac{1}{n}\text{SPD}_{x,y}$$

- Linear Regression y :
  - Slope:

$$w_1 = \boxed{\frac{\text{cov}_{(x,y),\text{biased}}}{\text{var}_{x,\text{biased}}}} = \frac{\cancel{\frac{1}{n}}\text{SPD}_{x,y}}{\cancel{\frac{1}{n}}\text{SSD}_x} = \frac{\sum_{i=1}^{n}(x_i - \overline{x})(y_i - \overline{y})}{\sum_{i=1}^{n}(x_i - \overline{x})^2}$$

where w_1 is the weight (Slope) from the equation $y = w_0 + w_1 x$, $\overline{x}$ is the Sample Mean of the x's, $\overline{y}$ is the Sample Mean of the y's, n is the number of points in the dataset, x_i are the x values from each point in the dataset, and y_i are the y values from each point in the dataset. m is also used for Slope.

  - Intercept:

$$w_0 = -w_1\overline{x} + \overline{y}$$

where w_0 is the weight (Intercept) from the equation $y = w_0 + w_1 x$, $\overline{x}$ is the Sample Mean of the x's, and $\overline{y}$ is the Sample Mean of the y's. b is also used for Intercept.

- Classification using a Separating Plane z :
  - Sample Mean for each Class:

$$\left(\overline{x}_{\text{Class 1}}, \overline{y}_{\text{Class 1}}\right) \qquad \left(\overline{x}_{\text{Class 2}}, \overline{y}_{\text{Class 2}}\right)$$

$$\left(\frac{\sum_{i=1}^{n_{\text{Class 1}}} x_{i,\text{Class 1}}}{n_{\text{Class 1}}}, \frac{\sum_{i=1}^{n_{\text{Class 1}}} y_{i,\text{Class 1}}}{n_{\text{Class 1}}}\right) \text{ and } \left(\frac{\sum_{i=1}^{n_{\text{Class 2}}} x_{i,\text{Class 2}}}{n_{\text{Class 2}}}, \frac{\sum_{i=1}^{n_{\text{Class 2}}} y_{i,\text{Class 2}}}{n_{\text{Class 2}}}\right)$$

where $n_{\text{Class 1}}$ is the number of points in the Class 1 dataset, $n_{\text{Class 2}}$ is the number of points in the Class 2 dataset, $x_{i,\text{Class 1}}$ are the x values from each point in the Class 1 dataset, $x_{i,\text{Class 2}}$ are the x values from each point in the Class 2 dataset, $y_{i,\text{Class 1}}$ are the y values from each point in the Class 1 dataset, and $y_{i,\text{Class 2}}$ are the y values from each point in the Class 2 dataset.

- Midpoint:

$$\left(x_{\text{midpoint}}, y_{\text{midpoint}}\right)$$
$$\left(\frac{\bar{x}_{\text{Class 1}} + \bar{x}_{\text{Class 2}}}{2}, \frac{\bar{y}_{\text{Class 1}} + \bar{y}_{\text{Class 2}}}{2}\right)$$

where $\bar{x}_{\text{Class 1}}$ is the Sample Mean of the x's in Class 1, $\bar{x}_{\text{Class 2}}$ is the Sample Mean of the x's in Class 2, $\bar{y}_{\text{Class 1}}$ is the Sample Mean of the y's in Class 1, $\bar{y}_{\text{Class 2}}$ is the Sample Mean of the y's in Class 2.

- Normal Vector $\vec{\mathbf{n}}$:
  - "Normal" to the separating line at the $z = 0$ level set for the plane z.

$$\begin{aligned}\vec{\mathbf{n}} &= \underbrace{\mathbf{p}_{\text{Positive Class Sample Mean}}}_{\text{Ending Point}} - \underbrace{\mathbf{p}_{\text{midpoint}}}_{\text{Starting Point}} \\ &= \left(\bar{x}_{\text{Positive Class}}, \bar{y}_{\text{Positive Class}}\right) - \left(x_{\text{midpoint}}, y_{\text{midpoint}}\right) \\ &= \boxed{\left(\bar{x}_{\text{Positive Class}} - x_{\text{midpoint}}, \bar{y}_{\text{Positive Class}} - y_{\text{midpoint}}\right)} \\ &= \left(n_x, n_y\right)\end{aligned}$$

where $\bar{x}_{\text{Positive Class}}$ is the Sample Mean of the x's for whatever class you want the plane z to produce positive values, $\bar{y}_{\text{Positive Class}}$ is the Sample Mean of the y's for whatever class you want the plane z to produce positive values, x_{midpoint} is the x value from the midpoint, y_{midpoint} is the y value from the midpoint, n_x is the x value of the normal vector, and n_y is the y value of the normal vector.

- Normalized Normal Vector:
  - "Normalized" such that the magnitude of the normal vector is 1.

$$\begin{aligned}\vec{\mathbf{n}}_{\text{normalized}} &= \frac{\vec{\mathbf{n}}}{\left\|\vec{\mathbf{n}}\right\|} \\ &= \frac{\left(n_x, n_y\right)}{\sqrt{n_x^2 + n_y^2}} \\ &= \left(\frac{n_x}{\sqrt{n_x^2 + n_y^2}}, \frac{n_y}{\sqrt{n_x^2 + n_y^2}}\right) \\ &= \boxed{\left(\cos(\theta), \sin(\theta)\right)}\end{aligned}$$

where $\vec{\mathbf{n}}$ is the normal vector, n_x is the x value of the normal vector, n_y is the y value of the normal vector, θ is the angle toward which the that the normalized normal vector is pointing.
Concentrate on this version:

$$\vec{\mathbf{n}}_{\text{normalized}} = \left(\cos(\theta), \sin(\theta)\right)$$

- Separating Plane z :

$$
\begin{aligned}
z &= \vec{\mathbf{n}}_{\text{normalized}} \bullet (\mathbf{x} - \mathbf{x}_0) \\
&= \underbrace{(\cos(\theta), \sin(\theta))}_{\text{Direction to Positive Class}} \bullet \left((x, y) - \underbrace{(x_0, y_0)}_{\substack{\text{Starting Point} \\ \text{for the} \\ \text{Separating} \\ \text{Line } (z=0)}} \right) \\
&= (\cos(\theta), \sin(\theta)) \bullet (x - x_0, y - y_0) \\
&= \boxed{\cos(\theta)(x - x_0) + \sin(\theta)(y - y_0)} \\
&= \cos(\theta) x - \cos(\theta) x_0 + \sin(\theta) y - \sin(\theta) y_0 \\
&= \boxed{(-\cos(\theta) x_0 - \sin(\theta) y_0) + (\cos(\theta)) x + (\sin(\theta)) y} \\
&= \boxed{w_0 + w_1 x + w_2 y}
\end{aligned}
$$

where θ is the angle toward which the that the normalized normal vector is pointing and (x_0, y_0) is the starting point from which the normalized normal vector points. Remember the normalized normal vector ALWAYS points toward a class. For the purposes of this test, it will ALWAYS point toward a class mean …the one that we want to be "positive".

  - The separating line that $\vec{\mathbf{n}}_{\text{normalized}}$ is normal to is the $z = 0$ level set.

- Weights:

$$
\begin{aligned}
w_0 &= -\cos(\theta) x_0 - \sin(\theta) y_0 \\
w_1 &= \cos(\theta) \\
w_2 &= \sin(\theta)
\end{aligned}
$$

where w_0, w_1, and w_2 are the weights from $z = w_0 + w_1 x + w_2 y$, θ is the angle toward which the that the normalized normal vector is pointing and (x_0, y_0) is the starting point from which the normalized normal vector points.

- Probability:
  - Product Rule:

$$P(A,B) = P(A|B)P(B) \qquad \text{or} \qquad P(A,B) = P(B|A)P(A)$$

$$P(A,B|X) = P(A|B,X)P(B,X) \qquad \text{or} \qquad P(A,B|X) = P(B|A,X)P(A,X)$$

  - Total Probability (Marginalization):

$$P(A) = P(A|X)P(X) + P(A|\neg X)P(\neg X)$$

  - Bayes' Rule:

$$\begin{aligned} P(A|B) &= \frac{P(B|A)P(A)}{P(B)} \\ &= \frac{P(B|A)P(A)}{\underbrace{P(B|A)P(A) + P(B|\neg A)P(\neg A)}_{\text{Total Probability}}} \end{aligned}$$

  - Conditional Independence:

$$P(A,B|X) = P(A|X)P(B|X)$$

$$P(A,B,C|X) = P(A|X)P(B|X)P(C|X)$$

- k-Nearest Neighbor:

function K-NEAREST-NEIGHBOR (*labelled-dataset*, *k*, *unlabeled-vector*) **return** *class-label*
 sort the *labelled-dataset* by closest distance to the *unlabeled-vector*
 return the *class-label* based on the majority vote of the class labels of the *k* closest labelled vectors from the sorted *labelled-dataset*

- Logistic Regression (using the Perceptron Learning Rule):

$$\underbrace{w_c \leftarrow w_c - \alpha \sum_{r=1}^{m} \left(\left(h_{\mathbf{w}}(\mathbf{x}_r) - y_r \right) x_{r,c} \right)}_{\text{Simultaneously update for } c=0,1,2,\ldots,n}$$

where r stands for row and c stands for column related to the $m \times n$ matrix of training data, $r = 1,\ldots,m$, $c = 0,1,\ldots,n$, $\mathbf{x}_r \in \mathbb{R}^{n+1}$ is the row vector from the $m \times n$ matrix of training data with $x_{r,0} = 1$ as the first component $\mathbf{x}_r = (1, x_{r,1}, x_{r,2}, \ldots, x_{r,n})$ where $x_{r,c}$ is the cu, $y_r \in \{0,1\}$ is the class label associated with each row vector $\mathbf{x}_r$, $\mathbf{w} \in \mathbb{R}^{n+1}$ is the current weight vector with $\mathbf{w} = (w_0, w_1, w_2, \ldots, w_n)$, $h_{\mathbf{w}}(\ )$ is the logistic function using the current weight vector $\mathbf{w}$ (the function that we're try to fit), and α is the learning rate.

- A* Search:

```
function A-STAR-SEARCH ( problem ) returns a solution or failure
    node ← a node with STATE = problem.INITIAL-STATE, PATH-COST = 0
    frontier ← a priority queue ordered by PATH-COST (i.e. f(n) = g(n) + h(n) where
        g(n) is the actual cost to reach the node and h(n), the heuristic, is the cost to get
        from the node to the goal) with node as the only element
    explored ← an empty set

    loop do
        if EMPTY?( frontier ) then return failure
        node ← POP( frontier ) /* chooses the node with the lowest f(n) in frontier */
        if problem.GOAL-TEST( node.State ) then return SOLUTION( node )
        add node.STATE to explored
        for each action in problem.ACTIONS( node.STATE ) do
            child ← CHILD-NODE( problem, node, action )
            if child.STATE is not in explored or frontier then
                frontier ← INSERT( child, frontier )
            /* PATH-COST: f(n) = g(n) + h(n) where
            g(n) is the actual cost to reach the node and
            h(n), the heuristic, is the cost to get from the node to the goal */
            else if child.STATE is in frontier with higher PATH-COST then
                replace that frontier node with child
```