1. **Naive Bayes Classification**
   **You are given a dataset containing records of weather conditions and whether a person decided to play tennis on that day. The goal is to use the Naive Bayes Classification method to predict whether the person will play tennis given the weather conditions. The dataset is as follows:**

   **Using the Naive Bayes Classifier, calculate the probability that the person will play tennis given that the weather is:**
   **• Outlook: Sunny**
   **• Temperature: Cool**
   **• Humidity: High**
   **• Wind: Strong**

We determine the likelihood of playing tennis given the weather conditions.

= P(PlayTennis=Yes, P(Xsunny, Xcool, Xhigh, Xstrong))
= P(PlayTennis=Yes) * (P(Xoutlook=Sunny | PlayTennis = Yes)
* (P(Xtemperature = Cool | PlayTennis = Yes)
* (P(Xhumidity=high | PlayTennis = Yes)
* (P(Xwind= strong | PlayTennis = Yes)

P(PlayTennis=Yes) = 9/14
(P(Xoutlook=Sunny | PlayTennis = Yes) = 2/9
(P(Xtemperature = Cool | PlayTennis = Yes) = 3/9
(P(Xhumidity=high | PlayTennis = Yes) = 3/9
(P(Xwind=stroong | PlayTennis = Yes) = 3/9

= 9/14 * 2/9 * 3/9 * 3/9 * 3/9
= 0.00529

We determine the likelihood of playing <u>not</u> tennis given the weather conditions.

= P(PlayTennis=No, P(Xsunny, Xcool, Xhigh, Xstrong))
= P(PlayTennis=No) * (P(Xoutlook=Sunny | PlayTennis = No)
* (P(Xtemperature = Cool | PlayTennis = No)
* (P(Xhumidity=high | PlayTennis = No)
* (P(Xwind= strong | PlayTennis = No)

P(PlayTennis=Yes) = 5/14
(P(Xoutlook=Sunny | PlayTennis = Yes) = 3/5

(P(Xtemperature = Cool | PlayTennis = Yes) = 1/5

(P(Xhumidity=high | PlayTennis = Yes) = 4/5

(P(Xwind=stroong | PlayTennis = Yes) = ⅗

= 5/14 * ⅗ * ⅕ * ⅘ * ⅗

= 0.02057

We compare the two likelihoods. The higher likelihood is what we predict. Given the weather conditions, the likelihood of not playing tennis is greater than the likelihood of playing tennis.

Therefore, we predict that the person will not play tennis.

## 2. Logistic Regression & Gradient Descent

1. **Compute hθ(x) for each training example, assuming initial parameter values are θ0 = 0 and θ1 = 0.**
2. **Calculate the log-likelihood ℓ(θ) using the initial parameter values.**
3. **Describe the steps to maximize the log-likelihood. Discuss how gradient ascent (the counterpart to gradient descent) could be applied to adjust the parameters θ0 and θ1 to maximize ℓ(θ). Then perform the first step of gradient ascent to update the parameters θ0 and θ1. Assume a learning rate α = 0.01**

To maximize the log-likellihood, we first must define the log-likelihood function. Then, we compute the log-likelihood. Next, we compute gradients if we want to optimize our log-likelihood function. Lastly, we update the parameters.

Gradient ascent will be applied to adjust the parameters θ0 and θ1 to maximize ℓ(θ). We want to use gradient ascent because we want to maximize rather than minimize.

1. $h\theta(1) = 1 / ( 1 + e^{(0+0*1)}) = 0.5$
   $h\theta(2) = 1 / ( 1 + e^{(0+0*2)}) = 0.5$
   $h\theta(3) = 1 / ( 1 + e^{(0+0*3)}) = 0.5$
   $h\theta(4) =1 / ( 1 + e^{(0+0*4)}) = 0.5$

2. $\ell(\theta) = \sum_{i=1}^{4} [ y(i) \log(h\theta(x(i))) + (1 - y(i)) \log(1 - h\theta(x(i))) ]$

   $= 0 * \log(0.5) + (1-0)\log(1-0.5) + 0 * \log(0.5) + (1-0)\log(1-0.5) + 1 * \log(0.5) + (1-1)\log(1-0.5) + 1 * \log(0.5) + (1-1)\log(1-0.5)$
   $= -1.20411$

3. To maximize ℓ(θ) , we need to maximize the gradient descent on each parameter θ. Then, we would update each parameter simultaneously for each iteration.

$$\theta j := \theta j + \alpha \sum_{i=1}^{m} (y(i) - h\theta(x(i)))\ x\_j(i)$$

$\theta 0 := 0 + 0.01(0) = 0$
$\theta 1 := 0 + 0.01(0 - 1/2) = -0.015$

**3. Convex Functions**
**Let f: Rn → R and g : Rn → R be convex functions.**
**Consider the following combinations of f and g:**

**1. The sum h(x) = f (x) + g(x)**

h(x) is convex.

Proof:

$f(\lambda x+(1-\lambda)y)+g(\lambda x+(1-\lambda)y) \leq \lambda f(x)+(1-\lambda)f(y)+\lambda g(x)+(1-\lambda)g(y)$

$=$

$h(\lambda x+(1-\lambda)y) \;\leq\; \lambda h(x)+(1-\lambda)f(y)$

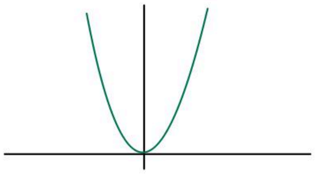
The convex inequality holds for h(x). Therefore, h(x) is necessarily convex.

**2. The pointwise maximum h(x) = max{f (x), g(x)}**

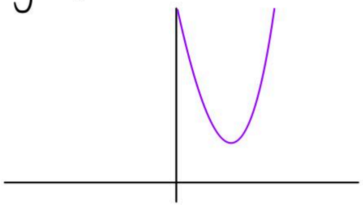
h(x) is not convex.

Proof: Counter example. h(x) is the function with the maximum point at each point. When h(x) chooses the maximum point from f(x) and g(x) for all points, it results in the bottom right graph. h(x) is not convex as it is not true that two distinct points do lie above the graph. More importantly, h(x) is not continuous and all convex functions are continuous. Therefore, h(x) is not convex.

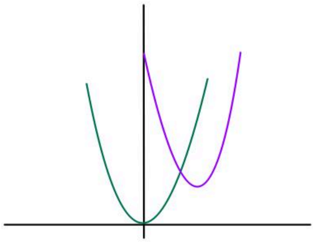
f(x)



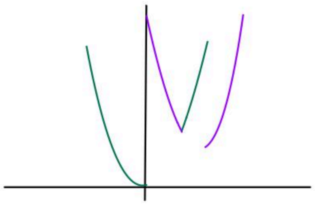
g(x)



f(x) & g(x) overlay for demonstration purposes



h(x) (picked max pt)



**3. The composition with an affine function a(x) = Ax + b, where A is a matrix and b is a vector, resulting in h(x) = f (a(x)) = f (Ax + b)**

f(a(x)) is convex.

Proof: a(x) is an affine function, meaning it is a linear function plus a constant. Affine functions preserve convexity. Since f(x) is convex, f(a(x)) is also convex by the property of affine functions.

1. **Multiclass Perceptron**
1. **For each data point in the training dataset, apply the multiclass perceptron update rule if the prediction does not match the actual activity class. If a tie occurs, resolve it by breaking ties in ascending order, i.e., you would predict activity class corresponding to the smallest number in the tie. Note that you only need to conduct a single perceptron update pass for each data point, beginning with training sample 1 and proceeding to 3.**

Sample 1:
x = [1,20,70]

w_0 * x = (w_0 * x_0) + (w_1 * x_1) + (w_2 * x_2) = 0.5*1 + 0*20 + -0.5*70 = -34.5
w_1 * x = (w_0 * x_0) + (w_1 * x_1) + (w_2 * x_2) = 0.5*1 + 0*20 + -0.5*70 = -34.5
w_2 * x = (w_0 * x_0) + (w_1 * x_1) + (w_2 * x_2) = 0.5*1 + 0*20 + -0.5*70 = -34.5

Break ties by picking the smallest number (class) in the tie.
So, we predict class 0.
This is incorrect. The correct class is class 1.

Update weight vectors:
w_0 - x = [-0.5, -20, -70.5] = w_0
w_1 + x = [1.5, 20, 69.5] = w_1

Sample 2:
x = [1,15,80]

w_0 * x = (w_0 * x_0) + (w_1 * x_1) + (w_2 * x_2)
= -0.5*1 + -20*15 + -70.5*80 = -5940.5
w_1 * x = 1.5*1 + 20*15 + 69.5*80 = 5861.5
w_2 * x = 0.5*1 + 0*15 + -0.5*80 = -39.5

Predicted class 1.
This is incorrect. Correct class is 0.
Update weight vectors:
w_0 + x = [0.5, -5, 9.5] = w_0
w_1 - x = [0.5, 5, -10.5] = w_1

Sample 3:
x = [1,30,85]

w_0* x = 658
w_1*x = -742
w_2*x = -42

Predicted class 0. Correct class is class 2. Update weight vectors.

w_0 - x = [-0.5, -35, -75.5] = w_0
w_2 + x = [1.5, 30, 84.5] = w_2

Final weight vectors for reference:
w_0 = [-0.5, -35, -75.5]
w_1 = [0.5, 5, -10.5]
w_2 = [1.5, 30, 84.5]

2. **After updating the model with a single pass through the three training data points, predict the activity class for new weather conditions: Temperature = 22, Humidity = 85.**

x = [1,22,85]

w_0 * x = -7188
w_1 * x = -782
w_2 * x = 7844

Predicted activity class 2.