# COMPSCI 671D Fall 2019
# Homework 1
Vinayak Gupta (vg101)

## 1 Perceptron

**(a)** $\mathbf{x}_1 = [1, 1]$, $\mathbf{x}_2 = [0, -1]$, and $\mathbf{x}_3 = [-2, 1]$, $y_1 = 1$, $y_2 = -1$, and $y_3 = 1$

Assume $\max_i \|\mathbf{x}_i\|_2 = 1$
To rescale the points, divide all the points by $\sqrt{5}$ as currently $\max_i \|\mathbf{x}_i\|_2 = \sqrt{5}$ when i=3

Now $\mathbf{x}_1 = [1/\sqrt{5}, 1/\sqrt{5}]$, $\mathbf{x}_2 = [0, -1/\sqrt{5}]$, and $\mathbf{x}_3 = [-2/\sqrt{5}, 1/\sqrt{5}]$

**1st Mistake**:

$$w^0 x_1 = 0$$
$$w^1 = w^0 + y_i x_i => w^1 = [1/\sqrt{5}, 1/\sqrt{5}]$$

Now, we will calculate the margin of every point from the new hyperplane and if there is a non-positive margin, we stop and improve our hyperplane

Margin at iteration T is given by $y_i w^T x_i$ and we calculate this for each point:

$$\text{1st point: } 1([1/\sqrt{5}, 1/\sqrt{5}].[1/\sqrt{5}, 1/\sqrt{5}]) > 0$$
$$\text{2nd point: } -1([1/\sqrt{5}, 1/\sqrt{5}].[0, -1/\sqrt{5}]) > 0$$
$$\text{3rd point: } 1([1/\sqrt{5}, 1/\sqrt{5}].[-2/\sqrt{5}, 1/\sqrt{5}]) < 0$$

**2nd Mistake**:

$$w^2 = w^1 + y_i x_i => w^2 = [1/\sqrt{5}, 1/\sqrt{5}] + 1([-2/\sqrt{5}, 1/\sqrt{5}])$$
$$w^2 = [-1/\sqrt{5}, 2/\sqrt{5}]$$

$$\text{1st point: } 1([-1/\sqrt{5}, 2/\sqrt{5}].[1/\sqrt{5}, 1/\sqrt{5}]) = 1/5 > 0$$
$$\text{2nd point: } -1(-[1/\sqrt{5}, 2/\sqrt{5}].[0, -1/\sqrt{5}]) = 2/5 > 0$$
$$\text{3rd point: } 1([-1/\sqrt{5}, 2/\sqrt{5}].[-2/\sqrt{5}, 1/\sqrt{5}]) = 4/5 > 0$$

Margin of every point from the hyperplane is $> 0$ and the algorithm makes **2** mistakes.
Also, $w^2$ is already normalized so we can calculate the $\delta$ which comes out to be 1/5 as for all $i, y_i \mathbf{w}^T \mathbf{x}_i \geq 1/5$

Now suppose the algorithm starts from $\mathbf{x}_2$ instead

**1st Mistake**:

$$w^0 x_2 = 0$$
$$w^1 = w^0 + y_i x_i => w^1 = [0, 1/\sqrt{5}]$$

$$\text{2nd point: } -1([0, 1/\sqrt{5}].[0, -1/\sqrt{5}]) = 1/5 > 0$$
$$\text{3rd point: } 1([0, 1/\sqrt{5}].[-2/\sqrt{5}, 1/\sqrt{5}]) = 1/5 > 0$$
$$\text{1st point: } 1([0, 1/\sqrt{5}].[1/\sqrt{5}, 1/\sqrt{5}]) = 1/5 > 0$$

Margin of every point from the hyperplane is $> 0$ and the algorithm makes **1** mistake.
Also, to normalize $w^1$, we divide it by $1/\sqrt{5}$ and therefore the margins by the same amount (Since we are dividing by a scalar value). We can calculate the $\delta$ which comes out to be $1/\sqrt{5}$ as for all $i, y_i \mathbf{w}^T \mathbf{x}_i \geq 1/\sqrt{5}$

**(b)** $\mathbf{x}_1 = [1, 0]$, $\mathbf{x}_2 = [0, -1]$, and $\mathbf{x}_3 = [-2, 1]$, $y_1 = 1$, $y_2 = -1$, and $y_3 = 1$

Assume $\max_i \|\mathbf{x}_i\|_2 = 1$
To rescale the points, divide all the points by $\sqrt{5}$ as currently $\max_i \|\mathbf{x}_i\|_2 = \sqrt{5}$ when i=3

Now $\mathbf{x}_1 = [1/\sqrt{5}, 0]$, $\mathbf{x}_2 = [0, -1/\sqrt{5}]$, and $\mathbf{x}_3 = [-2/\sqrt{5}, 1/\sqrt{5}]$

**1st Mistake**:

$$w^0 x_1 = 0$$
$$w^1 = w^0 + y_i x_i => w^1 = [1/\sqrt{5}, 0]$$

$$\text{1st point: } 1([1/\sqrt{5}, 0].[1/\sqrt{5}, 0]) > 0$$
$$\text{2nd point: } -1([1/\sqrt{5}, 0].[0, -1/\sqrt{5}]) = 0$$

**2nd Mistake**:

$$w^2 = w^1 + y_i x_i => w^2 = [1/\sqrt{5}, 0] - 1([0, -1/\sqrt{5}])$$
$$w^2 = [1/\sqrt{5}, 1/\sqrt{5}]$$

$$\text{3rd point: } 1([1/\sqrt{5}, 1/\sqrt{5}].[-2/\sqrt{5}, 1/\sqrt{5}]) < 0$$

**3rd Mistake**:

$$w^3 = w^2 + y_i x_i => w^3 = [1/\sqrt{5}, 1/\sqrt{5}] + 1([-2/\sqrt{5}, 1/\sqrt{5}])$$
$$w^3 = [-1/\sqrt{5}, 2/\sqrt{5}]$$

$$\text{1st point: } 1([-1/\sqrt{5}, 2/\sqrt{5}].[1/\sqrt{5}, 0]) < 0$$

**4th Mistake**:

$$w^4 = w^3 + y_i x_i => w^4 = [-1/\sqrt{5}, 2/\sqrt{5}] + 1([1/\sqrt{5}, 0])$$
$$w^4 = [0, 2/\sqrt{5}]$$

$$\text{2nd point: } -1([0, 2/\sqrt{5}].[0, -1/\sqrt{5}]) > 0$$
$$\text{3rd point: } 1([0, 2/\sqrt{5}].[-2/\sqrt{5}, 1/\sqrt{5}]) > 0$$
$$\text{1st point: } 1([0, 2/\sqrt{5}].[1/\sqrt{5}, 0]) = 0$$

**5th Mistake**:

$$w^5 = w^4 + y_i x_i => w^5 = [0, 2/\sqrt{5}] + 1([1/\sqrt{5}, 0])$$
$$w^5 = [1/\sqrt{5}, 2/\sqrt{5}]$$

$$\text{2nd point: } -1([1/\sqrt{5}, 2/\sqrt{5}].[0, -1/\sqrt{5}]) > 0$$
$$\text{3rd point: } 1([1/\sqrt{5}, 2/\sqrt{5}].[-2/\sqrt{5}, 1/\sqrt{5}]) = 0$$

**6th Mistake**:

$$w^6 = w^5 + y_i x_i => w^6 = [1/\sqrt{5}, 2/\sqrt{5}] + 1([-2/\sqrt{5}, 1/\sqrt{5}])$$
$$w^6 = [-1/\sqrt{5}, 3/\sqrt{5}]$$

$$\text{1st point: } 1([-1/\sqrt{5}, 3/\sqrt{5}].[1/\sqrt{5}, 0]) < 0$$

**7th Mistake**:

$$w^7 = w^6 + y_i x_i => w^7 = [-1/\sqrt{5}, 3/\sqrt{5}] + 1([1/\sqrt{5}, 0])$$
$$w^7 = [0, 3/\sqrt{5}]$$

$$\text{2nd point: } -1([0, 3/\sqrt{5}].[0, -1/\sqrt{5}]) > 0$$
$$\text{3rd point: } 1([0, 3/\sqrt{5}].[-2/\sqrt{5}, 1/\sqrt{5}]) > 0$$
$$\text{1st point: } 1([0, 3/\sqrt{5}].[1/\sqrt{5}, 0]) = 0$$

**8th Mistake**:

$$w^8 = w^7 + y_i x_i => w^8 = [0, 3/\sqrt{5}] + 1([1/\sqrt{5}, 0])$$
$$w^8 = [1/\sqrt{5}, 3/\sqrt{5}]$$

$$\text{2nd point: } -1([1/\sqrt{5}, 3/\sqrt{5}].[0, -1/\sqrt{5}]) = 3/5 > 0$$
$$\text{3rd point: } 1([1/\sqrt{5}, 3/\sqrt{5}].[-2/\sqrt{5}, 1/\sqrt{5}]) = 1/5 > 0$$
$$\text{1st point: } 1([1/\sqrt{5}, 3/\sqrt{5}].[1/\sqrt{5}, 0]) = 1/5 > 0$$

Margin of every point from the hyperplane is $> 0$ and the algorithm makes **8** mistakes. Also, to normalize $w^8$, we divide it by $\sqrt{2}$ and therefore the margins by the same amount We can calculate the $\delta$ which comes out to be $1/5\sqrt{2}$ as for all $i, y_i \mathbf{w}^T \mathbf{x}_i \geq 1/5\sqrt{2}$

Now, if $\mathbf{x}_3$ is changed to [-10, 1]
$\mathbf{x}_1 = [1, 1]$, $\mathbf{x}_2 = [0, -1]$, and $\mathbf{x}_3 = [-10, 1]$, $y_1 = 1$, $y_2 = -1$, and $y_3 = 1$

Assume $\max_i \|\mathbf{x}_i\|_2 = 1$
To rescale the points, divide all the points by $\sqrt{101}$ as currently $\max_i \|\mathbf{x}_i\|_2 = \sqrt{101}$ when i=3

Now $\mathbf{x}_1 = [1/\sqrt{101}, 1/\sqrt{101}]$, $\mathbf{x}_2 = [0, -1/\sqrt{101}]$, and $\mathbf{x}_3 = [-10/\sqrt{101}, 1/\sqrt{101}]$

**1st Mistake**:

$$w^0 x_1 = 0$$
$$w^1 = w^0 + y_i x_i => w^1 = [1/\sqrt{101}, 1/\sqrt{101}]$$

$$\text{1st point: } 1([1/\sqrt{101}, 1/\sqrt{101}].[1/\sqrt{101}, 1/\sqrt{101}]) > 0$$
$$\text{2nd point: } -1([1/\sqrt{101}, 1/\sqrt{101}].[0, -1/\sqrt{101}]) > 0$$
$$\text{3rd point: } 1([1/\sqrt{101}, 1/\sqrt{101}].[-10/\sqrt{101}, 1/\sqrt{101}]) < 0$$

**2nd Mistake**:

$$w^2 = w^1 + y_i x_i => w^2 = [1/\sqrt{101}, 1/\sqrt{101}] + 1([-10/\sqrt{101}, 1/\sqrt{101}])$$
$$w^2 = [-9/\sqrt{101}, 2/\sqrt{101}]]$$

$$\text{1st point: } 1([-9/\sqrt{101}, 2/\sqrt{101}].[1/\sqrt{101}, 1/\sqrt{101}]) < 0$$

**3rd Mistake**:

$$w^3 = w^2 + y_i x_i => w^3 = [-9/\sqrt{101}, 2/\sqrt{101}] + 1([1/\sqrt{101}, 1/\sqrt{101}])$$
$$w^3 = [-8/\sqrt{101}, 3/\sqrt{101}]$$

$$\text{2nd point: } -1([-8/\sqrt{101}, 3/\sqrt{101}].[0, -1/\sqrt{101}]) > 0$$
$$\text{3rd point: } 1([-8/\sqrt{101}, 3/\sqrt{101}].[-10/\sqrt{101}, 1/\sqrt{101}]) > 0$$
$$\text{1st point: } 1([-8/\sqrt{101}, 3/\sqrt{101}].[1/\sqrt{101}, 1/\sqrt{101}]) < 0$$

**4th Mistake**:

$$w^4 = w^3 + y_i x_i => w^4 = [-8/\sqrt{101}, 3/\sqrt{101}] + 1([1/\sqrt{101}, 1/\sqrt{101}])$$
$$w^4 = [-7/\sqrt{101}, 4/\sqrt{101}]$$

$$\text{2nd point: } -1([-7/\sqrt{101}, 4/\sqrt{101}].[0, -1/\sqrt{101}]) > 0$$
$$\text{3rd point: } 1([-7/\sqrt{101}, 4/\sqrt{101}].[-10/\sqrt{101}, 1/\sqrt{101}]) > 0$$
$$\text{1st point: } 1([-7/\sqrt{101}, 4/\sqrt{101}].[1/\sqrt{101}, 1/\sqrt{101}]) < 0$$

**5th Mistake**:

$$w^5 = w^4 + y_i x_i => w^5 = [-7/\sqrt{101}, 4/\sqrt{101}] + 1([1/\sqrt{101}, 1/\sqrt{101}])$$
$$w^5 = [-6/\sqrt{101}, 5/\sqrt{101}]$$

$$\text{2nd point: } -1([-6/\sqrt{101}, 5/\sqrt{101}].[0, -1/\sqrt{101}]) > 0$$
$$\text{3rd point: } 1([-6/\sqrt{101}, 5/\sqrt{101}].[-10/\sqrt{101}, 1/\sqrt{101}]) > 0$$
$$\text{1st point: } 1([-6/\sqrt{101}, 5/\sqrt{101}].[1/\sqrt{101}, 1/\sqrt{101}]) < 0$$

**6th Mistake**:

$$w^6 = w^5 + y_i x_i => w^6 = [-6/\sqrt{101}, 5/\sqrt{101}] + 1([1/\sqrt{101}, 1/\sqrt{101}])$$
$$w^6 = [-5/\sqrt{101}, 6/\sqrt{101}]$$

$$\text{2nd point: } -1([-5/\sqrt{101}, 6/\sqrt{101}].[0, -1/\sqrt{101}]) = 6/101 > 0$$
$$\text{3rd point: } 1([-5/\sqrt{101}, 6/\sqrt{101}].[-10/\sqrt{101}, 1/\sqrt{101}]) = 56/101 > 0$$
$$\text{1st point: } 1([-5/\sqrt{101}, 6/\sqrt{101}].[1/\sqrt{101}, 1/\sqrt{101}]) = 1/101 > 0$$

Margin of every point from the hyperplane is $> 0$ and the algorithm makes **6** mistakes.
Also, to normalize $w^6$, we divide it by $\sqrt{61}/\sqrt{101}$ and therefore the margins by the same amount
We can calculate the $\delta$ which comes out to be $1/\sqrt{61 * 101}$ as for all $i, y_i \mathbf{w}^T \mathbf{x}_i \geq 1/\sqrt{61 * 101}$

**(c)** As calculated in the above two parts, the $\delta$ when $\mathbf{x}_1 = [1, 1]$, $\mathbf{x}_2 = [0, -1]$, and $\mathbf{x}_3 = [-2, 1]$ is $\mathbf{1/5}$. It changes to $\mathbf{1/5\sqrt{2}}$ when $\mathbf{x}_1$ becomes $[1, 0]$ and to $\mathbf{1/\sqrt{61 * 101}}$ when $\mathbf{x}_3$ becomes $[-10, 1]$. This happens because of the normalization of data that has to be done since we assume $\max_i \|\mathbf{x}_i\|_2 = 1$. The greater the 2-norm normalization factor, the more the rescaling of data happens. This suggests that if there is an outlier (a point which is far from the other points) and its 2-norm is big (as is the case when $\mathbf{x}_3$ becomes $[-10, 1]$), then the points that are already close to each other get even further closer and it becomes more difficult to find a hyperplane separating them (if the points have different $y_i$ sign) thus reducing the $\delta$. When $\mathbf{x}_1$ becomes $[1, 0]$ then the points $\mathbf{x}_1$ and $\mathbf{x}_2$ come closer and even though the $\max_i \|\mathbf{x}_i\|_2 = 1$ does not change, finding a hyperplane separating the two becomes more tough and hence $\delta$ reduces. Finally, since the total number of mistakes depends inversely on the squared value of $\delta$, these changes increase the total number of possible mistakes performed by the algorithm

**(d)** Assume that we know the data is separable by a hyperplane $\mathbf{w}^*$, then $-\mathbf{w}^*$ is a hyperplane that classifies all the points incorrectly. We initialize $w^0$ to be $-w^*$ then and start with the point that has the least distance from this hyperplane. At every iteration, we select the inclassified point that has the least distance from the hyperplane because this suggests that the algorithm will make the least amount of improvement in every iteration. This is equivalent to saying that between any two consecutive iterations, the angle between the hyperplanes separating the points at those iterations should be minimum. This can be shown as follows:

Suppose $\|\mathbf{w}^t\|_2 = 1$ and $\|\mathbf{w}^{t+1}\|_2 = 1$

$cos(\theta) = w^{t+1}.w^t = (w^t + y_i x_i).w^t$
$= w^t.w^t + (y_i x_i).w^t$
$= 1 +$ (distance of the inclassified point from hyperplane)

Now, if we select the inclassified point which has the minimum distance from the hyperplane (which is -ve), $cos(\theta)$ would be closer to 1 and hence $\theta$ is small which means the angle between the hyperplanes at two consecutive iterations should be minimum. Therefore, selecting the inclassified point which has the minimum distance from hyperplane at every iteration gives the algorithm that causes the Perceptron algorithm to make maximum mistakes before converging.

## 2 Decision Trees

**(a)** The decision tree representing the logical rule

$$(A \wedge B) \vee (\bar{A} \wedge C)$$

with the least possible number of nodes is shown in the following Figure 1:

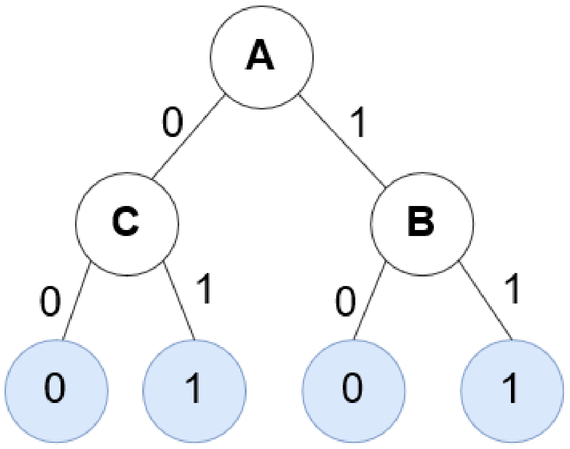
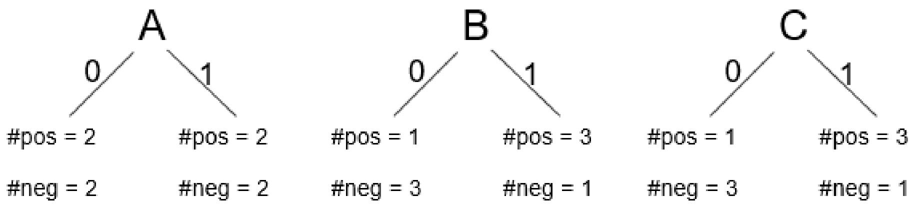


Figure 1: Optimal decision tree representing $(A \wedge B) \vee (\bar{A} \wedge C)$

**(b)** The C4.5 algorithm at every node splits on the feature that gives the maximum Information Gain:

$$\begin{aligned}\text{Gain(S;A)} &= \text{expected reduction in entropy due to branching on attribute A}\\ &= \text{original entropy - entropy after branching}\end{aligned}$$

Let 0 value be depicted by negatives. The following figure shows how the first split occurs on different features:



$$\text{Original Entropy} = H_0 = H([\tfrac{1}{2}, \tfrac{1}{2}])$$

$$\text{Entropy if branching on A=>}\ H_{1A} = \tfrac{1}{2}H([\tfrac{1}{2}, \tfrac{1}{2}]) + \tfrac{1}{2}H([\tfrac{1}{2}, \tfrac{1}{2}])$$

$$\text{Gain(S;A)} = H([\tfrac{1}{2}, \tfrac{1}{2}]) - (\tfrac{1}{2}H([\tfrac{1}{2}, \tfrac{1}{2}]) + \tfrac{1}{2}H([\tfrac{1}{2}, \tfrac{1}{2}]))$$

$$= 0$$

$$\text{Entropy if branching on B}\Rightarrow H_{1B} = \tfrac{1}{2}H([\tfrac{1}{4},\tfrac{3}{4}]) + \tfrac{1}{2}H([\tfrac{3}{4},\tfrac{1}{4}])$$

$$\text{Gain(S;B)} = H([\tfrac{1}{2},\tfrac{1}{2}]) - (\tfrac{1}{2}H([\tfrac{1}{4},\tfrac{3}{4}]) + \tfrac{1}{2}H([\tfrac{3}{4},\tfrac{1}{4}]))$$

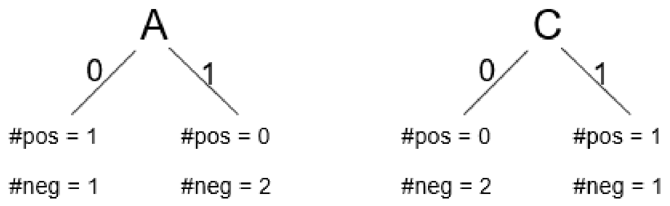
$$= 0.188722$$

Also, since the splitting on feature C is similar to B, the Gain in C is same as B. However, we select the ties lexicographically so the **first** splitting point is **B** as it has the maximum Gain.

Now, we look at the left subtree of B. This subtree has all the values of B as 0 and the following truth table represents the data going through in the left subtree:

| $A$ | $B$ | $C$ | $(A \wedge B) \vee (\bar{A} \wedge C)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 0 |

The following figure shows the split on the features of A and C:



$$\text{Original Entropy at branch} = H([\tfrac{1}{4},\tfrac{3}{4}])$$

$$\text{Entropy if branching on A}\Rightarrow H_{21A} = \tfrac{1}{2}H([\tfrac{1}{2},\tfrac{1}{2}]) + \tfrac{1}{2}H([0,1])$$

$$\text{Gain(S;A)} = H([\tfrac{1}{4},\tfrac{3}{4}]) - (\tfrac{1}{2}H([\tfrac{1}{2},\tfrac{1}{2}]) + \tfrac{1}{2}H([0,1]))$$

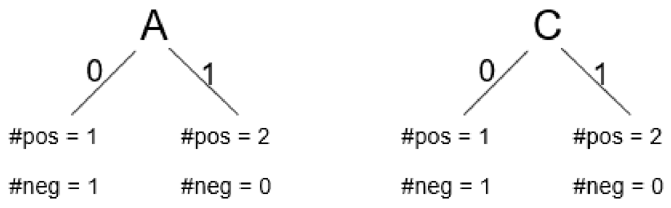
$$= 0.311278$$

Also, since the splitting on feature C is similar to A, the Gain in C is same as A. However, we select the ties lexicographically so the **next** splitting point is **A** on the left subtree of **B** as it has the maximum Gain.

Now, we look at the right subtree of B. This subtree has all the values of B as 1 and the following truth table represents the data going through in the right subtree:

| $A$ | $B$ | $C$ | $(A \wedge B) \vee (\bar{A} \wedge C)$ |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

The following figure shows the split on the features of A and C:



$$\text{Original Entropy at branch} = H([\tfrac{1}{4}, \tfrac{3}{4}])$$

$$\text{Entropy if branching on A=> } H_{22A} = \tfrac{1}{2}H([\tfrac{1}{2}, \tfrac{1}{2}]) + \tfrac{1}{2}H([1, 0])$$

$$\text{Gain(S;A)} = H([\tfrac{1}{4}, \tfrac{3}{4}]) - (\tfrac{1}{2}H([\tfrac{1}{2}, \tfrac{1}{2}]) + \tfrac{1}{2}H([1, 0]))$$

$$= 0.311278$$

Also, since the splitting on feature C is similar to A, the Gain in C is same as A. However, we select the ties lexicographically so the **next** splitting point is **A** on the right subtree of **B** as it has the maximum Gain.

The left subtree (A) of B is able to classify the negative points whenever its value is 1. So we need to split for the left subtree of A which is when A is equal to 0. The following truth table represents the data through the left subtree of A:

| $A$ | $B$ | $C$ | $(A \wedge B) \vee (\bar{A} \wedge C)$ |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |

Now we split on **C** as final value is the same as C.

Similarly, the right subtree (A) of B is able to classify the positive points whenever its value is 1. So we need to split for the left subtree of A which is when A is equal to 0. The following truth table represents the data through the left subtree of A:

| $A$ | $B$ | $C$ | $(A \wedge B) \vee (\bar{A} \wedge C)$ |
|---|---|---|---|
| 0 | 1 | 0 | 0 |
| 0 | 1 | 1 | 1 |

Again, we split on **C** as final value is the same as C.

The final decision tree is shown on the next page.

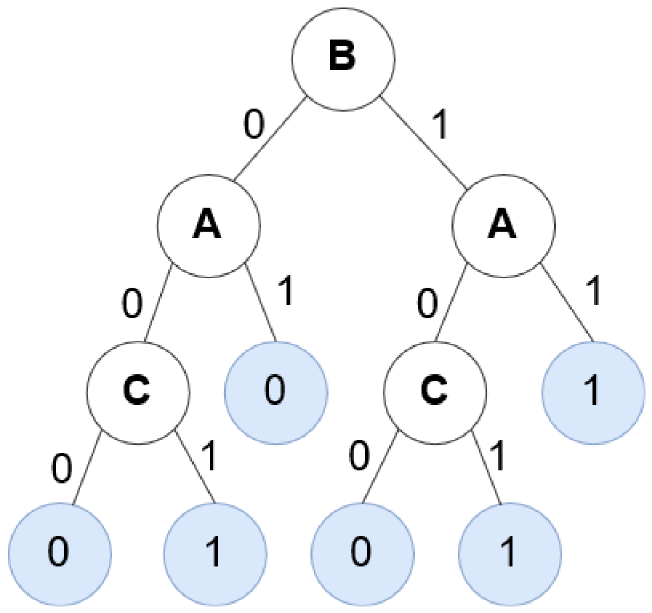


Figure 2: C4.5 Decision tree representing $(A \wedge B) \vee (\bar{A} \wedge C)$

**(c)** The decision tree in **(b)** is not the same as the tree in **(a)**. The C4.5 algorithm does a greedy search for optimal splitting point, however, it might not result in a global optimal tree. In other words, selecting the decision tree split (at each node as we move down the tree) that maximizes information gain will necessarily **not** guarantee an optimal decision tree.

## 3 Evaluation Metrics for Imbalanced Data

**(a)** Suppose,

m = Number of points in the resampled dataset
n = Number of points in the original dataset
P = Number of positives in the original dataset
N = Number of negatives in the original dataset
$(x_i, y_i)$ = Point in the original dataset where $y_i$ is either +1 or -1
$p_i$ = Probability of the data point in the resampled data
f = Classifier

$$E[Acc_{resampled}] := \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{n} p_i \mathbf{1}_{[f(x_i) == y_i]}$$

where $\mathbf{1}_{[f(x_i)==y_i]}$ is the Indicator function
Since, its a summation over finite elements, we can break it into two parts: one when $y_i = 1$ and the other when $y_i = -1$

$$E[Acc_{resampled}] := \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{P} p_i \mathbf{1}_{[f(x_i)==y_i]} \mathbf{1}_{[y_i==1]} + \frac{1}{m} \sum_{j=1}^{m} \sum_{i=1}^{N} p_i \mathbf{1}_{[f(x_i)==y_i]} \mathbf{1}_{[y_i==-1]}$$

Now, $p_i$ for positive sample is equal to $\frac{1}{2P}$ as probability for selecting a positive point $p_i * P = \frac{1}{2}$

Similarly, for negative sample $p_i$ is equal to $\frac{1}{2N}$

$$E[Acc_{resampled}] := \frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{P}\frac{1}{2P}\mathbf{1}_{[f(x_i)==y_i]}\mathbf{1}_{[y_i==1]} + \frac{1}{m}\sum_{j=1}^{m}\sum_{i=1}^{N}\frac{1}{2N}\mathbf{1}_{[f(x_i)==y_i]}\mathbf{1}_{[y_i==-1]}$$

$\sum_{i=1}^{P}\mathbf{1}_{[f(x_i)==y_i]}\mathbf{1}_{[y_i==1]}$effectively represents the Total Positives (TP) in the original sample

Similarly, $\sum_{i=1}^{N}\mathbf{1}_{[f(x_i)==y_i]}\mathbf{1}_{[y_i==-1]}$ represents the Total Negatives (TN) in the original sample

$$E[Acc_{resampled}] := \frac{1}{m}\sum_{j=1}^{m}\frac{TP}{2P} + \frac{1}{m}\sum_{j=1}^{m}\frac{TN}{2N}$$

Also, $\frac{TP}{P}$ is Total Positive Rate (TPR) and $\frac{TN}{N}$ is Total Negative Rate (TNR) of original sample

$$E[Acc_{resampled}] := \frac{1}{m}\sum_{j=1}^{m}\frac{TPR}{2} + \frac{1}{m}\sum_{j=1}^{m}\frac{TNR}{2}$$

$E[Acc_{resampled}] := \frac{TPR+TNR}{2} = Acc_{balanced}$ of original sample

Hence, proved that Expected accuracy of resampled data is same as the Balanced accuracy of original sample.

**(b)** Suppose that the predicted score are sorted in a descending order. In general, the ROC curve is constructed by moving the classifier threshold through all the points. We will calculate the AUC in such a way that whenever the threshold passes through a negative point, we calculate the TPR at that point and we sum up through all such encounters of negative points. Also, one thing to note is that whenever a positive point is encountered the ROC curve moves vertically up by $\frac{1}{P}$, where P are the total positives in the dataset and if a negative point is encountered, the graph moves right by $\frac{1}{N}$ , where N are the total negatives in the dataset. We can now show the calculation of AUC mathematically using Indicator functions as follows:

$$AUC := \sum_{i=1}^{n}\sum_{j=1}^{i}\frac{\mathbf{1}_{[y_i==-1]}}{N}\frac{\mathbf{1}_{[y_j==1]}}{P}$$

where n = total points in the dataset

In the above equation, the outer sum iterates through all the points in the dataset and whenever a negative point is ecountered, the inner sum calculates the TPR of that point by iterating through all the positives upto that point. Now, since N and P are constant we can take them out of the dual sum:

$$AUC := \frac{1}{NP}\sum_{i=1}^{n}\sum_{j=1}^{i}\mathbf{1}_{[y_i==-1]}\mathbf{1}_{[y_j==1]}$$

Also, j is always less than equal to i but when i is equal to j then the value of $\mathbf{1}_{[y_i==-1]}\mathbf{1}_{[y_j==1]}$ is equal to 0 since a point cannot be both positive and negative at the same time. So effectively, i is always greater than j. The above equation can be equivalently written as:

$$AUC := \frac{1}{NP}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{1}_{[y_i==-1]}\mathbf{1}_{[y_j==1]}\mathbf{1}_{[i>j]}$$

Now, since we have sorted the predicted scores in the descending order, $i > j$ means that $f(x_i) < f(x_j)$ . The equation becomes:

$$AUC := \frac{1}{NP}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{1}_{[y_i==-1]}\mathbf{1}_{[y_j==1]}\mathbf{1}_{[f(x_i)<f(x_j)]}$$

The probability that a positive sample has larger predicted score than a negative one is:

$$\mathbf{P}(f(x_i) < f(x_j)|y_i == -1, y_j == 1)$$

This can be equivalently written as :

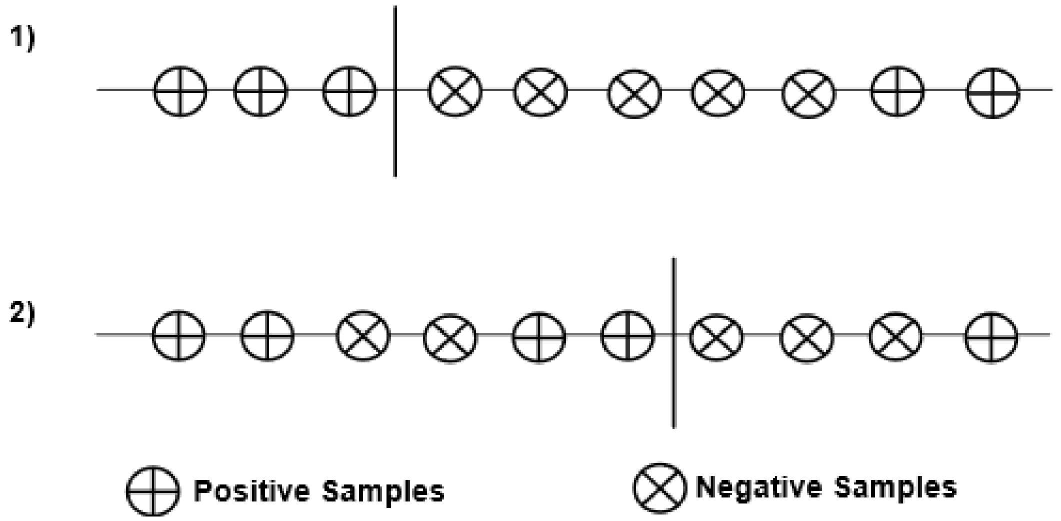
$$\frac{1}{NP}\sum_{i=1}^{n}\sum_{j=1}^{n}\mathbf{1}_{[y_i==-1]}\mathbf{1}_{[y_j==1]}\mathbf{1}_{[f(x_i)<f(x_j)]}$$

because in the above equation, the numerator calculates the number of times a positive point has a higher predicted score than a negative one, and the denominator represents all the possible combinations of positive-negative pairs from the dataset.

This equation is exactly the same as the one derived for AUC. Therefore we can say that, AUC equals the probability that a positive sample has larger predicted score than a negative one, where probability is calculated with respect to the draw of positive-negative pairs from the dataset

**(c)** Since AUC equals the probability that a positive sample has a larger predicted score than a negative one, where probability is calculated with respect to the draw of positive-negative pairs from the dataset, it does not matter whether the positive/ negative sample is in majority or minority. Moreover, since AUC is the area under ROC and ROC depends on the TPR and FPR which are ratios and not the absolute number of majority/minority class, the class imbalance does not affect the AUC. Only the distribution of data points ( predicted scores) determines the AUC and it is definitely better than pure accuracy in dealing with imbalanced data just like balanced accuracy.

**(d)** The fact that the balanced accuracy and AUC are different can be shown through the following two examples:



In 1), the balanced accuracy and AUC can be calculated as follows:

$TPR_1 = 3/5$ , $TNR_1 = 5/5$ , therefore Balanced Accuracy $BA_1 = 4/5 = 8/10$
$AUC_1 = 3/5 * 5/5 + 0 = 3/5 = 15/25$

In 2), the balanced accuracy and AUC can be calculated as follows:

$TPR_2 = 4/5$ , $TNR_2 = 3/5$ , therefore Balanced Accuracy $BA_2 = 7/10$
$AUC_2 = 2/5 * 2/5 + 4/5*3/5 + 0 = 16/25$

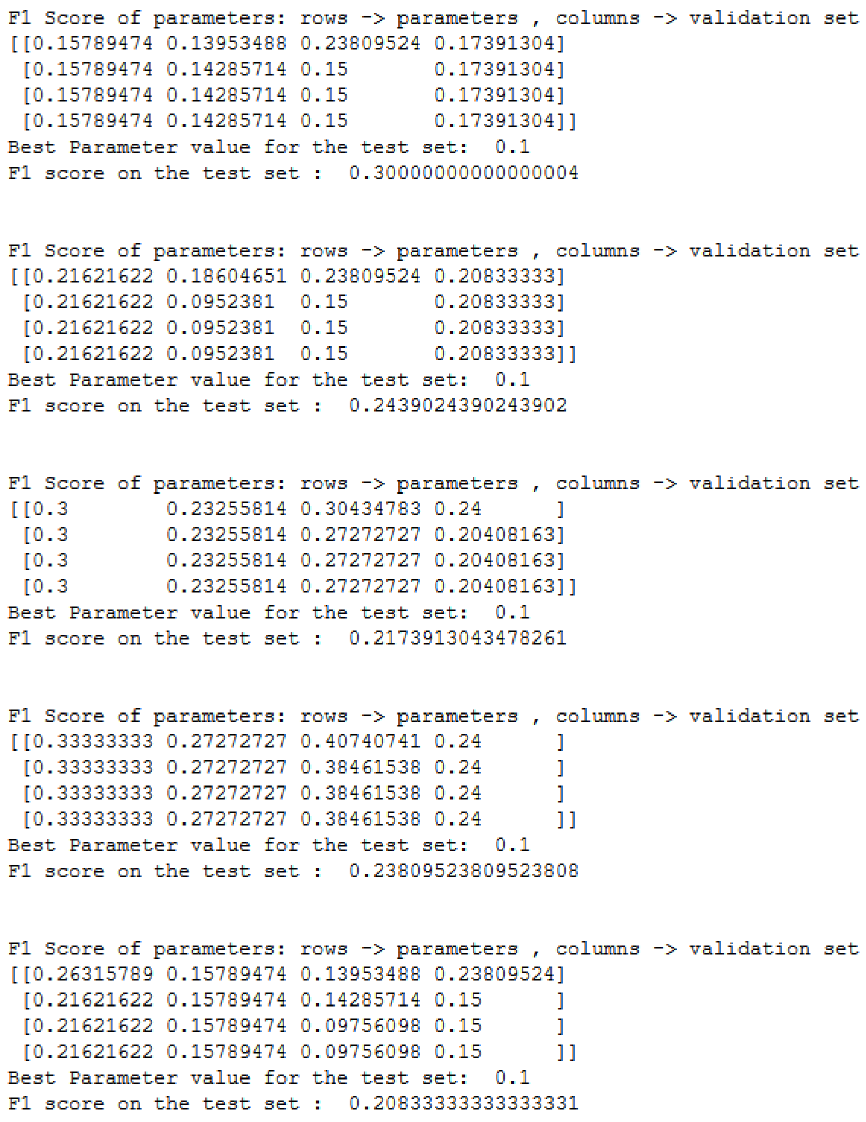
As we can see that $BA_1 > BA_2$ , however, $AUC_1 < AUC_2$ , the AUC and BA are different as they disagree with each other.

## 4 Validation and Testing

**(a)** The following code presents a k-fold nested cross validation algorithm:

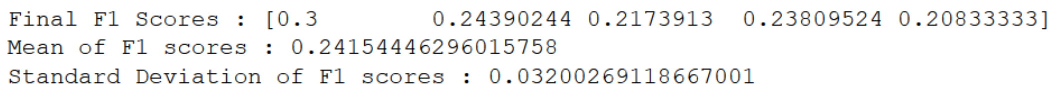
```python
import numpy as np
import csv
from sklearn.linear_model import LogisticRegression
from sklearn.metrics import f1_score
import math
import random
import warnings
warnings.filterwarnings('ignore')

def read_data():
    data = []
    with open('transfusion.data') as csvfile:
        reader = csv.reader(csvfile)
        headers = next(reader) # take the header out
        for row in reader: # each row is a list
            data.append(row)
    data  = np.array(data, dtype = np.int32)
    data=np.take(data,np.random.RandomState(seed=15).permutation(data.shape[0]),
    axis=0);
    # Randomizes the data before splitting into dependent and independent
    variables
    X = data[:,:-1]
    y = data[:,-1]
    return X, y


X, y = read_data()
records=X.shape[0] # Total number of rows
temp=math.ceil(records/5) # Number of rows in each fold
C = [0.1,1,10,100] # Hyperparameter for regularization
final_f1_value = np.zeros(5) # Array to store the final F1 scores for each test
    set

for i in range(0,5): #Start of test loop
    if(i!=4): # Splitting data into training and test for the first 4 test sets
        X_test= X[i*temp:(i+1)*temp,] #Test set of the current loop for
    independent variables
        y_test= y[i*temp:(i+1)*temp] #Test set of the current loop for dependent
    variable
        X_train_1=X[0:i*temp,]
        X_train_2= X[(i+1)*temp:,]
        X_train= np.concatenate((X_train_1,X_train_2),0)#Outer Training set of the
     current loop for independent variable
        y_train_1=y[0:i*temp]
        y_train_2= y[(i+1)*temp:]
        y_train= np.concatenate((y_train_1,y_train_2)) #Outer Training set of the
    current loop for dependent variable
    else: # Splitting data into training and test for the last test set since last
     fold might have fewer rows
        X_test=X[i*temp:,]
        X_train= X[0:i*temp,]
        y_test=y[i*temp:,]
        y_train= y[0:i*temp,]
    f1_matrix=np.zeros((4,4)) # 4 x 4 matrix to store the F1 scores of every parameter acroos the 4 validation folds
    for j in range(0,4): # Start of the validation loop
        if(j!=3):
            X_val = X_train[j*temp:(j+1)*temp,]#Validation set of the current loop for independent variables
            X_inner_train_1 = X_train[0:j*temp,]
            X_inner_train_2 = X_train[(j+1)*temp:,]
            X_inner_train= np.concatenate((X_inner_train_1,X_inner_train_2),0) # Inner Training set of the current loop for independent variable
            y_val = y_train[j*temp:(j+1)*temp] #Validation set of the current loop for dependent variables
            y_inner_train_1 = y_train[0:j*temp]
            y_inner_train_2 = y_train[(j+1)*temp:]
            y_inner_train= np.concatenate((y_inner_train_1,y_inner_train_2))#Inner Training set of the current loop for dependent variable
        else:
            X_val=X_train[j*temp:,]
            X_inner_train= X_train[0:j*temp,]
            y_val=y_train[j*temp:,]
            y_inner_train= y_train[0:j*temp,]
        l=0
        for k in C:# Iterate through all parameters
            model = LogisticRegression(C=k)
            model.fit(X_inner_train, y_inner_train)
            y_pred = model.predict(X_val)
            f1_matrix[l][j]=f1_score(y_val, y_pred)# Store the F1 score in the 4 x 4 matrix
            l=l+1
    print("F1 Score of parameters: rows -> parameters , columns -> validation set")
    print(f1_matrix)
    f1_matrix_sum=f1_matrix.sum(axis=1)# Sum(equivalent to comparing average as we divide by 4) the F1 score for the parameters across the validation loop
    max_val= np.amax(f1_matrix_sum) # Maximum value
    max_ind = np.where(f1_matrix_sum==max_val) # Selecting the index of parameter with max value
    print("Best Parameter value for the test set: ",C[max_ind[0][0]])# Output the best parameter selected for the current test set
    model = LogisticRegression(C=C[max_ind[0][0]])
    model.fit(X_train, y_train)
    y_pred = model.predict(X_test)
    final_f1_value[i]= f1_score(y_test, y_pred) # Storing the F1 score using the best paramterer for current test set
    print("F1 score on the test set : ",final_f1_value[i])
    print("\n")

print("Final F1 Scores :",final_f1_value)
print("Mean of F1 scores :",np.mean(final_f1_value))# Mean of F1 scores
print("Standard Deviation of F1 scores :" , np.std(final_f1_value))# Standard Deviation of F1 scores
```

The following shows the output at the end of nested cross validation:



```
F1 Score of parameters: rows -> parameters , columns -> validation set
[[0.15789474 0.13953488 0.23809524 0.17391304]
 [0.15789474 0.14285714 0.15       0.17391304]
 [0.15789474 0.14285714 0.15       0.17391304]
 [0.15789474 0.14285714 0.15       0.17391304]]
Best Parameter value for the test set:  0.1
F1 score on the test set :  0.30000000000000004


F1 Score of parameters: rows -> parameters , columns -> validation set
[[0.21621622 0.18604651 0.23809524 0.20833333]
 [0.21621622 0.0952381  0.15       0.20833333]
 [0.21621622 0.0952381  0.15       0.20833333]
 [0.21621622 0.0952381  0.15       0.20833333]]
Best Parameter value for the test set:  0.1
F1 score on the test set :  0.2439024390243902


F1 Score of parameters: rows -> parameters , columns -> validation set
[[0.3        0.23255814 0.30434783 0.24      ]
 [0.3        0.23255814 0.27272727 0.20408163]
 [0.3        0.23255814 0.27272727 0.20408163]
 [0.3        0.23255814 0.27272727 0.20408163]]
Best Parameter value for the test set:  0.1
F1 score on the test set :  0.2173913043478261


F1 Score of parameters: rows -> parameters , columns -> validation set
[[0.33333333 0.27272727 0.40740741 0.24      ]
 [0.33333333 0.27272727 0.38461538 0.24      ]
 [0.33333333 0.27272727 0.38461538 0.24      ]
 [0.33333333 0.27272727 0.38461538 0.24      ]]
Best Parameter value for the test set:  0.1
F1 score on the test set :  0.23809523809523808


F1 Score of parameters: rows -> parameters , columns -> validation set
[[0.26315789 0.15789474 0.13953488 0.23809524]
 [0.21621622 0.15789474 0.14285714 0.15      ]
 [0.21621622 0.15789474 0.09756098 0.15      ]
 [0.21621622 0.15789474 0.09756098 0.15      ]]
Best Parameter value for the test set:  0.1
F1 score on the test set :  0.20833333333333331
```

The final F1 scores along with the mean and Standard Deviation are also shown:



```
Final F1 Scores : [0.3        0.24390244 0.2173913  0.23809524 0.20833333]
Mean of F1 scores : 0.24154446296015758
Standard Deviation of F1 scores : 0.03200269118667001
```

As we can see, the value of hyerparameter that is selected every time is **0.1** (This depends on the randomization of dataset though, it is not necessary that this value of hyperparameter will be selected every time). The **Mean** of the evaluation measure (F1 score) is **0.2415444** and the **Standard Deviation** is **0.03200269**

**(b)** I would select $C_1$ in practice if it outperforms $C_2$ in validation even though $C_2$ performs better on the test set. Suppose we are doing a k-fold nested cross validation, then for any outer loop (testing loop), if $C_1$ is performing better in the validation set, then it means essentially it performs better on k-1 folds than $C_2$. Even if $C_2$ performs better on a test set, it cannot be generalized for the test sets of other iterations. But, if you select $C_1$ , it has more probability of performing better on the test sets of other iterations as validation ensures the best parameter over k-1 folds. Moreover, you cannot use your test set to essentialy select a hyperparameter as it defeat the entire purpose of validation process.

**(c)** Let the dataset has J features and n rows, then for the outer loop training set we use the following formula:

For every feature j,

$$\mu_j(TrainingSet) = \frac{\sum_{i=1}^{n'} value_{ij}}{n'}$$

$$\sigma_j(TrainingSet) = \sqrt{\frac{\sum_{i=1}^{n'} (value_{ij} - \mu_j)^2}{n'}}$$

$$newvalue_{ij} = \frac{value_{ij} - \mu_j}{\sigma_j}, \forall i \in 1, ..., n', \forall j \in 1, ..., J$$

For the outer loop training set, the value of $n'$ will be rows in k-1 folds (training set). In other words, every training set in the outer loop is normalized by its own mean and own standard deviation.

To normalize the test set in every loop, we use the mean and standard deviation of the training set of that loop:

For every feature j,

$$TestNewValue_{ij} = \frac{TestValue_{ij} - \mu_j(TrainingSet)}{\sigma_j(TrainingSet)}, \forall i \in 1, ..., n'', \forall j \in 1, ..., J$$

where the value of $n''$ is the number of rows in the test set/ test fold. This normalization is done at every iteration of the test loop (outer loop). This is done because in the real world scenariao, we assume that we do not know the actual data distribution of the test set and therefore use the mean and standard deviation of training set on which the model is trained to best predict our results.

Since the validation set is a part of the outer loop training set, it automatically is normalized. Moreover, we do not need to explicitly perform a normalization on the validation set because it is just used to select the best hyperparameter. Performing a normalization on that using the inner loop training set would not give accurate results because the mean and standard deviation of the inner loop training set will start taking part in it and hence the validation set may not necessarily give the best hyperparameter.