Boolean Combination Algorithms & Applications

(Code, Visualisation and Analysis)

Gunda Venkata Sai
MITACS Intern

Supervisor : Prof. Wahab Hamou-Lhadj

Concordia University
July 20th, 2022

Outline

- Context
- Research Objective and Deliverables
- Boolean Combination Algorithms and their Applications
- Evaluation & Conclusion

Context

What are Boolean Combinations?

A	B	~A	~B	A ∪ B	A ∩ B	A ⊕ B	~A ∪ B
0	0	1	1	0	0	0	1
0	1	1	0	1	0	1	1
1	0	0	1	1	0	1	0
1	1	0	0	1	1	0	1

$A =$	0	1	1	0	1	0	0	1
$B =$	1	1	0	1	0	0	0	1
$A \cup B =$	1	1	1	1	1	0	0	1
$A \cap B =$	0	1	0	0	0	0	0	1
$A \oplus B =$	1	0	1	1	1	0	0	0

Boolean Combination Algorithms

There are 5 Boolean Combination Algorithms :

- BBC2
- IBC
- PBC
- WPBC2
- WPIBC

BBC Algorithm

The Pair-wise Brute-force Boolean Combination (BBC2) fuses all possible pairs of crisp classifiers generated from all the available soft classifiers using all Boolean functions.

For Example, Consider 3 soft detector scores with 4 thresholds each, That would make 12 crisp detectors, Hence 1440 combinations (N*N*10) .

<u>Pros and Cons of BBC2:</u>

1. Exploits all Boolean functions using an exhaustive brute-force search to determine optimum points leads to an exponential number of combinations.
2. High Computation Complexity. O(N*N)

BBC Pseudo Code & Visualisation

$n_k \leftarrow$ number of decision thresholds of D_k using $\mathcal{V}$ // num. of vertices on ROC(D_k).

let $n = \sum_{k=1}^{K} n_k$

$BooleanFunctions \leftarrow$ $\{a \wedge b, \neg a \wedge b, a \wedge \neg b, \neg(a \wedge b), a \vee b, \neg a \vee b, a \vee \neg b, \neg(a \vee b), a \oplus b, a \equiv b\}$

allocate $\boldsymbol{C}$ an array of size: $[|\mathcal{V}|, n]$ // storage of all crisp detectors' decisons.

convert soft detectors to crisp detectors

for $i \leftarrow 1$ **to** K **do**

  for $j \leftarrow 1$ **to** n_i **do**

    $\boldsymbol{R} \leftarrow (D_i, t_j)$ // responses of D_i at decision threshold t_j using $\mathcal{V}$.

    push $\boldsymbol{R}$ onto $\boldsymbol{C}$

allocate $\boldsymbol{F}$ an array of size: $[2, U^2 \times size(BooleanFunctions)]$ // temporary storage of combination results.

foreach $bf \in BooleanFunctions$ **do**

  for $i \leftarrow 1$ **to** U **do**

    $\boldsymbol{R_1} \leftarrow \boldsymbol{C}_{\text{selected}}[i]$ // Retrieve Decision Vector

    for $j \leftarrow 1$ **to** U **do**

      $\boldsymbol{R_2} \leftarrow \boldsymbol{C}_{\text{selected}}[j]$

      $\boldsymbol{R_c} \leftarrow bf(\boldsymbol{R_1}, \boldsymbol{R_2})$ // combine responses using current Boolean func.

      compute (tpr, fpr) of $\boldsymbol{R_c}$ using $\mathcal{V}$ // map combination to ROC plane

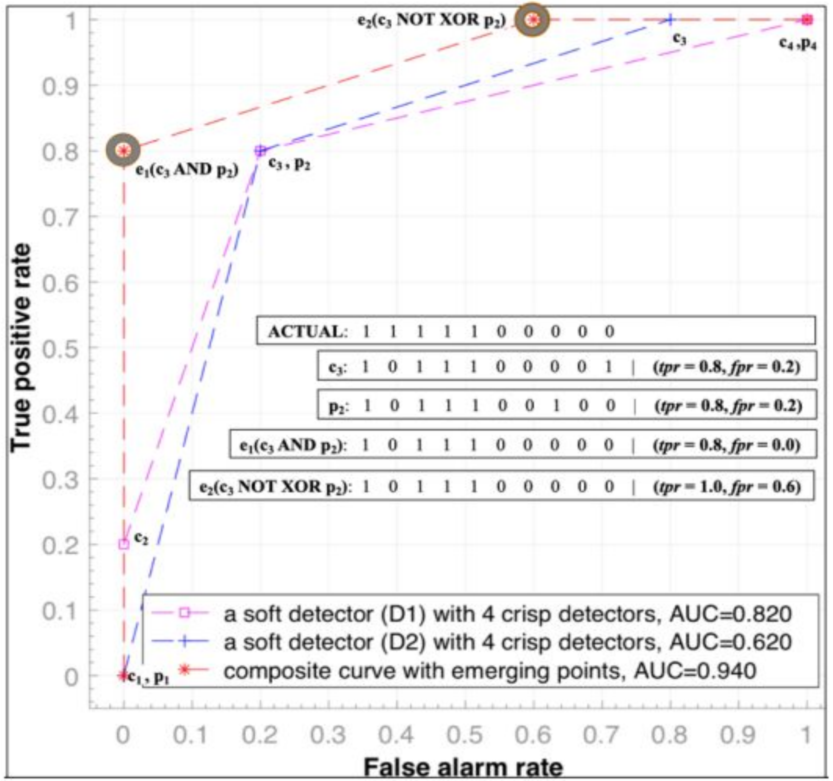
      push (tpr, fpr) onto $\boldsymbol{F}$

compute *ROCCH of all ROC points in* $\boldsymbol{F}$

$n_{ev} \leftarrow$ number of emerging vertices

$\mathcal{S} \leftarrow \{(D_1, t_i), (D_2, t_j), \ldots, (D_k, t_k), bf\}$ // set of selected decision thresholds from each detector and Boolean functions for emerging vertices.

store $\mathcal{S}$; **return** *ROCCH*



IBC

IBC avoids the impractical exponential explosion associated with the BBC2 by combining the emerging responses on a composite ROCCH sequentially. It first combines the first two ROC curves of the first two soft classifiers. Then, the combined ROCCH, particularly, the emerging points are combined with the next ROC curve, and so on until the K th ROC curve is combined. IBC repeats these sequential combinations iteratively until there are no further improvements or it reaches to a predefined maximum number of iterations.

IBC Pseudo Code

Algorithm 1. $BC_{ALL}(T_a, T_b, labels)$: Boolean combination of two ROC curves.

Input: Thresholds of ROC curves, T_a and T_b, and *labels* (of validation set)
Output: ROCCH and fused responses (Rab) of combined curves, where each point is the result of two fused thresholds along with the corresponding Boolean function (bf)

let $m \leftarrow$ number of distinct thresholds in T_a
let $n \leftarrow$ number of distinct thresholds in T_b
Allocate F an array of size: $[2, m \times n]$ // holds temporary results of fusions
$BooleanFunctions \leftarrow \{a \wedge b, \neg a \wedge b, a \wedge \neg b, \neg(a \wedge b), a \vee b, \neg a \vee b, a \vee \neg b, \neg(a \vee b), a \oplus b, a \equiv b\}$
Compute $ROCCH_{old}$ of the original curves
foreach $bf \in BooleanFunctions$ **do**
  for $i = 1, \ldots, m$ **do**
    $R_a \leftarrow (T_a \geq T_{a_i})$ // converting threshold of 1st ROC to responses
    for $j = 1, \ldots, n$ **do**
      $R_b \leftarrow (T_b \geq T_{b_j})$ // converting threshold of 2nd ROC to responses
      $R_c \leftarrow bf(R_a, R_b)$ // *combined responses with* bf
      Compute (tpr, fpr) using R_c and *labels*
      Push (tpr, fpr) onto F
  Compute $ROCCH_{new}$ of F
  Store thresholds and corresponding Boolean functions that exeeded the $ROCCH_{old}$,
  $S^*_{global} \leftarrow (T_{a_x}, T_{b_y}, bf)$ // *to be used during operations*
  Store the responses of these emerging points into R // *to be used with* BCM_{ALL} *and* IBC_{ALL}
  $ROCCH_{new} \leftarrow ROCCH_{old}$ // *Update ROCCH*
Return $ROCCH_{new}$, R, S^*_{global}

Algorithm 2. $BCM_{ALL}([T_1,\ldots,T_K],labels)$: Cumulative combination of multiple ROC curves based on BC_{ALL}.

Input: Thresholds of K ROC curves $[T_1,\ldots,T_K]$ and *labels*
Output: ROCCH of combined curves where each point is the result of the combination of combinations

```
[ROCCH_1,R_1] = BC_ALL(T_1,T_2,labels)        // combine the first two ROC curves
for k = 3,...,K do
    // combine the responses of the previous combination with those of the following ROC curve
    [ROCCH_{k-1},R_{k-1}] = BC_ALL(R_{k-2},T_k,labels)
Return ROCCH_{K-1},R_{K-1} and the stored tree of the selected responses/thresholds fusions along with their corresponding fusion functions
```

Algorithm 3. $IBC_{ALL}([T_1,\ldots,T_K],labels)$: Iterative Boolean combination based on BC_{ALL} or BC_{ALL}

Input: Thresholds of K ROC curves $[T_1,\ldots,T_K]$ and *labels*
Output: ROCCH of combined curves where each point is the result of the combination of combinations through several iterations

```
[ROCCH_OLD,R_OLD] = BCM([T_1,T_2,...,T_K],labels)
while (AUC(ROCCH_NEW) ≥ AUC(ROCCH_OLD) + ε) or (numberIterations ≤ maxIter) do
    [ROCCH_NEW,R_NEW] = BC(R_OLD,[T_1,T_2,...,T_K],labels)
return ROCCH_NEW,R_NEW and the stored tree of the selected responses fusions along with their corresponding fusion functions
```

PBC

PBC prunes all trivial and redundant crisp classifiers and, then, selects complementary crisp classifiers based on the kappa agreements between each crisp classifier’s decisions and the true labels (ground truth) on the validation set.

PBC Pseudo Code

Algorithm 2: PBC($D_1, D_2, \ldots, D_K, \mathcal{V}$): Pruned Boolean Combination

input : K soft detectors ($D_1, D_2, \ldots, D_K$) and a validation set $\mathcal{V}$ of size $|\mathcal{V}|$

output: ROCCH of combined detectors.
- Each vertex is the result of exact 2 combination of crisp detectors.
- Each combination selects the best decision thresholds from different detectors (D_i, t_j) and Boolean function (stored in the set $\mathcal{S}$)

$n_k \leftarrow$ number of decision thresholds of D_k using $\mathcal{V}$ // num. of vertices on ROC(D_k).

let $n = \sum_{k=1}^{K} n_k$

$BooleanFunctions \leftarrow$ $\{a \wedge b, \neg a \wedge b, a \wedge \neg b, \neg(a \wedge b), a \vee b, \neg a \vee b, a \vee \neg b, \neg(a \vee b), a \oplus b, a \equiv b\}$

allocate $\boldsymbol{C}$ an array of size: $[|\mathcal{V}|, n]$ // storage of all crisp detectors' decisons.

convert soft detectors to crisp detectors

for $i \leftarrow 1$ **to** K **do**
- **for** $j \leftarrow 1$ **to** n_i **do**
  - $\boldsymbol{R} \leftarrow (D_i, t_j)$ // responses of D_i at decision threshold t_j using $\mathcal{V}$.
  - **push** $\boldsymbol{R}$ onto $\boldsymbol{C}$

choose Pruning Technique {MinMax-Kappa, ROCCH-Kappa}

reduce n to U // $U \ll n$: is a user defined max number of detectors

return $\boldsymbol{C}_{\text{selected}} \leftarrow \boldsymbol{C}$ - Pruned Detectors

// Subset of size U detectors selected from all original detectors and returned for combination

allocate $\boldsymbol{F}$ an array of size: $[2, U^2 \times size(BooleanFunctions)]$

// temporary storage of combination results.

foreach $bf \in BooleanFunctions$ **do**
- **for** $i \leftarrow 1$ **to** U **do**
  - $\boldsymbol{R_1} \leftarrow \boldsymbol{C}_{\text{selected}}[i]$ // Retrieve Decision Vector
  - **for** $j \leftarrow 1$ **to** U **do**
    - $\boldsymbol{R_2} \leftarrow \boldsymbol{C}_{\text{selected}}[j]$
    - $\boldsymbol{R_c} \leftarrow bf(\boldsymbol{R_1}, \boldsymbol{R_2})$ // combine responses using current Boolean func.
    - **compute** (tpr, fpr) of $\boldsymbol{R_c}$ using $\mathcal{V}$ // map combination to ROC plane
    - **push** (tpr, fpr) onto $\boldsymbol{F}$

compute *ROCCH of all ROC points in* $\boldsymbol{F}$

$n_{ev} \leftarrow$ number of emerging vertices

$\mathcal{S} \leftarrow \{(D_1, t_i), (D_2, t_j), \ldots, (D_k, t_k), bf\}$ // set of selected decision thresholds from each detector and Boolean functions for emerging vertices.

store $\mathcal{S}$; **return** *ROCCH*

WPBC2

WPBC2 prunes all the soft detectors using linear weighted kappa and again prunes the resulting crisp detectors using MinMaxKappa - Pruning. The Pruned Detectors are now combined using BBC algorithm.

```
// Phase1-pruning soft detectors using weighted kappa
allocate an array AUC_all[1:K]   // temporary store auc of each S_k
for k ← 1 to K do
    compute auc of ROC(S_k, T_k)
    push auc onto AUC_all
allocate an empty array B = []   //store selected diverse soft detectors
while (K)
    select base soft detector: S_b ← max_k[AUC_all(k)]
    store S_b onto B   // store S_b as a base soft detector
    let n_b ← number of order/levels/thresholds in T_b
    update K ← K - S_b                    // remove S_b from K soft detectors
    update AUC_all ← AUC_all - AUC_all(S_b)      // remove auc for S_b
    let n ← the size of |K|
    for k ← 1 to n do
        compute linear weighted kappa kp between S_k and S_b using n_b
        if 0.80 < kp <=1
            update K ← K - S_k   // remove S_k as a redundant copy of S_b
            update AUC_all ← AUC_all - AUC_all(S_k)   // remove auc for S_k
// -----Phase2- pruning crisp detectors using unweighted kappa-----------
let L ← number of selected diverse base soft detectors in B
let m ← number of selected complementary crisp detectors from S_b ∈ B
allocate an empty array θ = []  //store thresholds of each complementary crisp
//detectors
for b ← 1 to L do
    let n_b ← number of crisp detectors or thresholds in T_b ∈ S_b
    allocate an array U[1:n_b]    // store temporary kappa coefficients
    allocate an array V[|lab|:n_b] //store temporary responses
    for j ← 1 to n_b do
        r ← S_b ≥ t_j  //temporary responses at decision threshold t_j ∈ T_b
        compute unweighted kappa kp between r and lab
        push kp onto U and r onto V
    filter U and V by removing trivial detectors
    select m complementary crisp detectors using MinMaxKappa(U,V)
    pruning technique
    map m selected complementary crisp detectors into θ_b thresholds
    store θ_b thresholds onto θ// store θ_b complementary crisp detectors of S_b
return  B < S_1, ... S_L > and θ < θ_1, ... , θ_L >
```

WPIBC

WPIBC prunes all the soft detectors using linear weighted kappa and again prunes the resulting crisp detectors using MinMaxKappa - Pruning. The Pruned Detectors are now combined using IBC algorithm.

// Phase1-pruning soft detectors using weighted kappa
allocate an array $AUC_{all}[1{:}K]$ *// temporary store auc of each S_k*
for $k \leftarrow 1$ **to** K **do**
  compute auc of $ROC(S_k, T_k)$
  push auc onto AUC_{all}
allocate an empty array $B = []$ *//store selected diverse soft detectors*
while (K)
  select base soft detector: $S_b \leftarrow max_k[AUC_{all}(k)]$
  store S_b onto B *// store S_b as a base soft detector*
  let $n_b \leftarrow$ number of order/levels/thresholds in T_b
  update $K \leftarrow K - S_b$ *// remove S_b from K soft detectors*
  update $AUC_{all} \leftarrow AUC_{all} - AUC_{all}(S_b)$ *// remove auc for S_b*
  let $n \leftarrow$ the size of $|K|$
  for $k \leftarrow 1$ **to** n **do**
    compute linear weighted kappa kp between S_k and S_b using n_b
    if $0.80 < kp <= 1$
      update $K \leftarrow K - S_k$ *// remove S_k as a redundant copy of S_b*
      update $AUC_{all} \leftarrow AUC_{all} - AUC_{all}(S_k)$ *// remove auc for S_k*
// -----Phase2- pruning crisp detectors using unweighted kappa-----------
let $L \leftarrow$ number of selected diverse base soft detectors in B
let $m \leftarrow$ number of selected complementary crisp detectors from $S_b \in B$
allocate an empty array $\theta = []$ *//store thresholds of each complementary crisp //detectors*
for $b \leftarrow 1$ **to** L **do**
  let $n_b \leftarrow$ number of crisp detectors or thresholds in $T_b \in S_b$
  allocate an array $U[1{:}n_b]$ *// store temporary kappa coefficients*
  allocate an array $V[|lab|{:}n_b]$ *//store temporary responses*
  for $j \leftarrow 1$ **to** n_b **do**
    $r \leftarrow S_b \geq t_j$ *//temporary responses at decision threshold $t_j \in T_b$*
    compute unweighted kappa kp between r and lab
    push kp onto U and r onto V
  filter U and V by removing trivial detectors
  select m complementary crisp detectors using $MinMaxKappa(U,V)$ pruning technique
  map m selected complementary crisp detectors into θ_b thresholds
  store θ_b thresholds onto θ *// store θ_b complementary crisp detectors of S_b*
return $B < S_1, \dots S_L >$ and $\theta < \theta_1, \dots, \theta_L >$

Research Objectives and Deliverables

- Open Source Boolean Combination Libraries Contribution
- Boolean Combination of Heterogeneous Classifiers

Open Source Boolean Combination Algorithm Libraries Contribution

BBC

```python
from BBC_Algorithm import BBC

original = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

soft_detectors = [[3.9, 4.3, 3.4, 3.2, 3.6, 3.1, 3.0, 3.0, 2.3, 2.6, 2.5, 2.2, 0.8, 1.4, 0.8, 1.9, 0.8, 1.1, 1.3, 0.8, 0.8, 0.8, 1.1, 10.5, 0.8 ],
                  [3.2, 3.4, 3.1, 3.0, 3.7, 3.0, 3.0, 3.0, 2.3, 2.5, 2.5, 2.2, 0.9, 1.2, 0.9, 1.8, 0.9, 1.0, 1.2, 0.9, 0.9, 0.9, 1.3, 10.8, 0.9 ],
                  [2.9, 2.8, 3.2, 3.0, 3.1, 2.9, 2.8, 2.9, 2.2, 2.5, 2.4, 2.3, 1.9, 2.7, 1.9, 1.9, 1.9, 2.4, 2.0, 1.9, 1.9, 1.9, 2.4, 11.2, 1.9 ]]

thresholds = 4

BBC(original, soft_detectors, thresholds)
```

```
([0.0, 0.0, 0.05, 0.1, 1.0], [0.0, 0.4, 0.8, 1.0, 1.0], 0.9750000000000001)
```

Open Source Boolean Combination Algorithm Libraries Contribution

IBC

```
from IBC_Algorithm import IBC

original = [1, 1, 1, 1, 1, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0, 0]

soft_detectors = [
    [3.9, 4.3, 3.4, 3.2, 3.6, 3.1, 3.0, 3.0, 2.3, 2.6, 2.5, 2.2, 0.8, 1.4, 0.8, 1.9, 0.8, 1.1, 1.3, 0.8, 0.8, 0.8, 1.1,
     10.5, 0.8],
    [3.2, 3.4, 3.1, 3.0, 3.7, 3.0, 3.0, 3.0, 2.3, 2.5, 2.5, 2.2, 0.9, 1.2, 0.9, 1.8, 0.9, 1.0, 1.2, 0.9, 0.9, 0.9, 1.3,
     10.8, 0.9],
    [2.9, 2.8, 3.2, 3.0, 3.1, 2.9, 2.8, 2.9, 2.2, 2.5, 2.4, 2.3, 1.9, 2.7, 1.9, 1.9, 1.9, 2.4, 2.0, 1.9, 1.9, 1.9, 2.4,
     11.2, 1.9]]

thresholds = 12

print(IBC(soft_detectors, original, thresholds)[0])
print(IBC(soft_detectors, original, thresholds)[1])
print(IBC(soft_detectors, original, thresholds)[2])
```

```
C:\Users\gunda\IdeaProjects\BooleanCombinationClassifiers\venv\Scripts\python.exe C:/Users/gunda/IdeaProjects/BooleanCombinationCl
[0.0, 0.0, 0.1, 1.0]
[0.0, 0.6, 1.0, 1.0]
0.98
```

Boolean Combination of Heterogeneous Classifiers

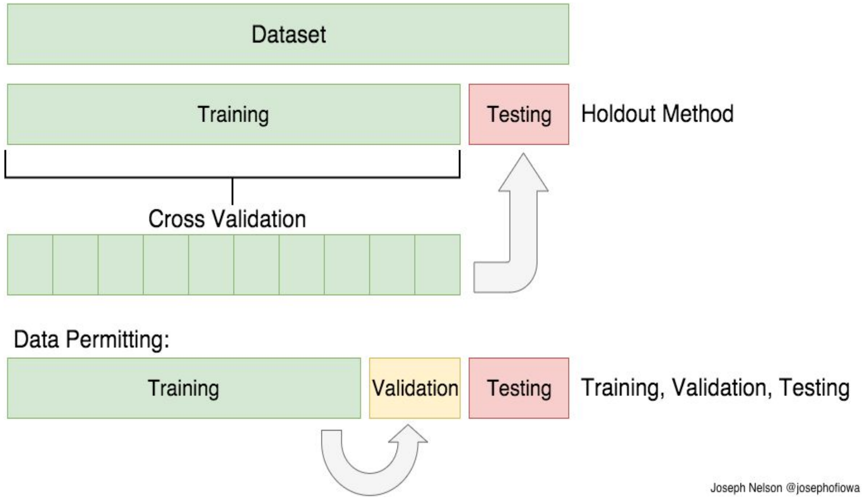
Table 2
Input metrics used in experiments

Input Metrics	# Independent of Variables	Metrics Type
CK	6	Static code metrics
NET	64	Network metrics
PROC	13	Process metrics
CK+NET	70	Combined metrics
CK+PROC	19	Combined metrics
NET+PROC	77	Combined metrics
CK+NET+PROC	83	Combined metrics

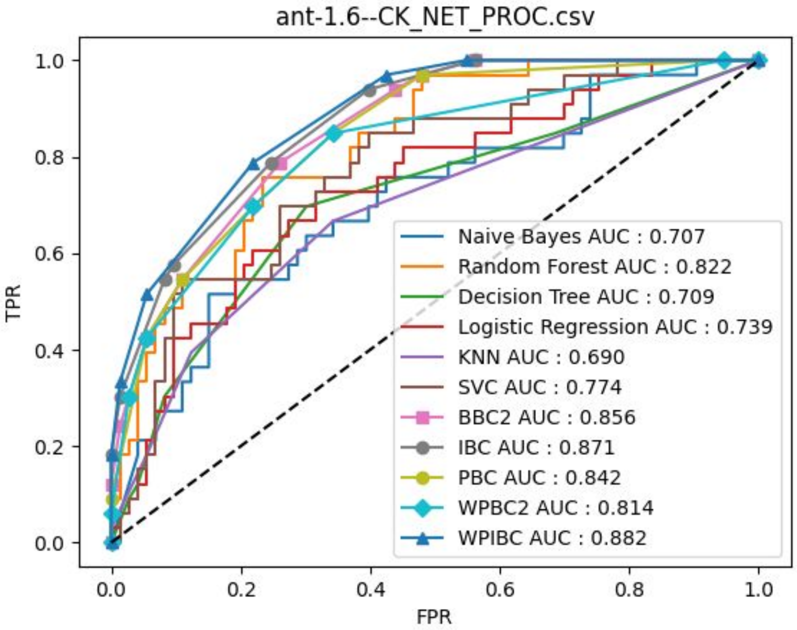
We have tested our approach on 162 different datasets metrics.

Training Heterogeneous Classifiers

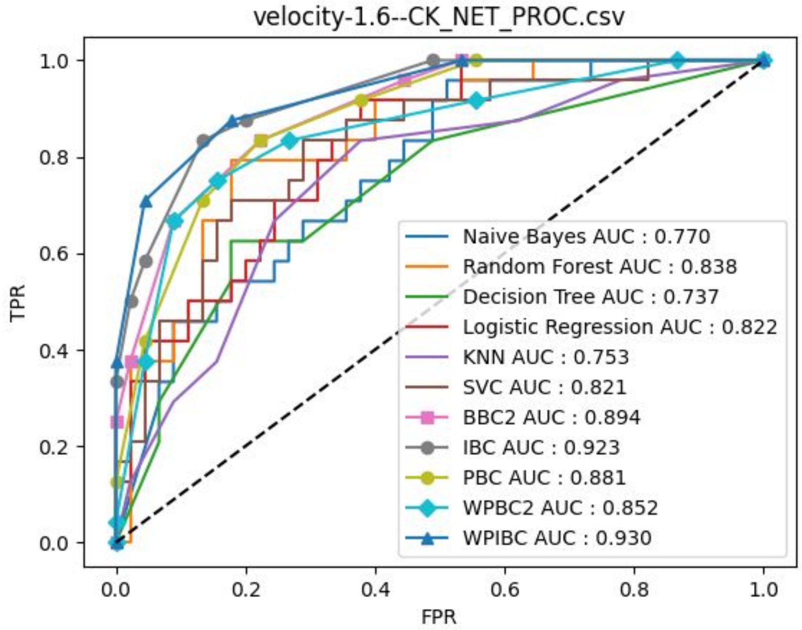
1. Firstly, We train 6 Models on all 162 datasets. They are Naive Bayes, RF, DT, SVM, LR, KNN.
2. For every training we split the dataset to 70 % train and 30 % test. Thereafter, perform a 5 fold cross validation using gridsearchCV to tune and obtain best parameters.
3. To compute the soft scores we predict the probabilities on the test data for every models. Hence Every dataset obtain 6 soft detectors with their length of Y_test.



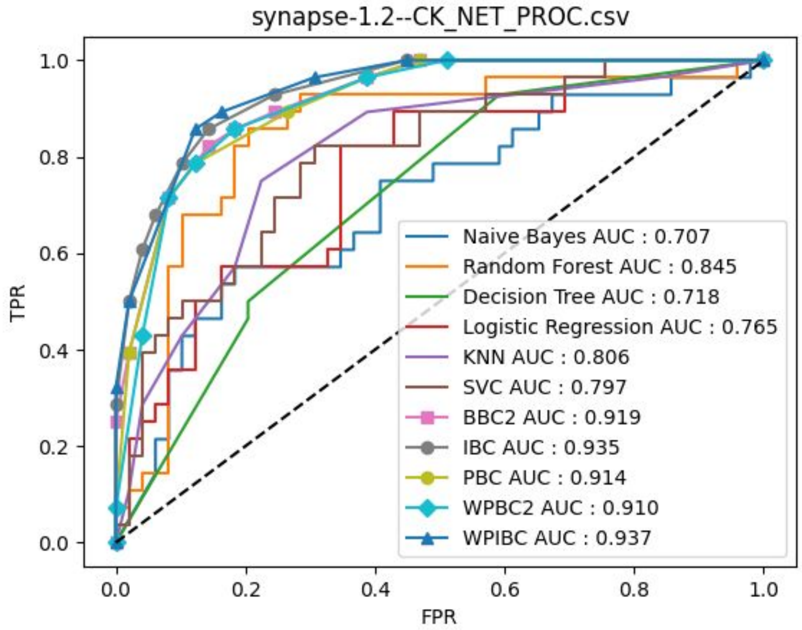
ROC and AUC Results



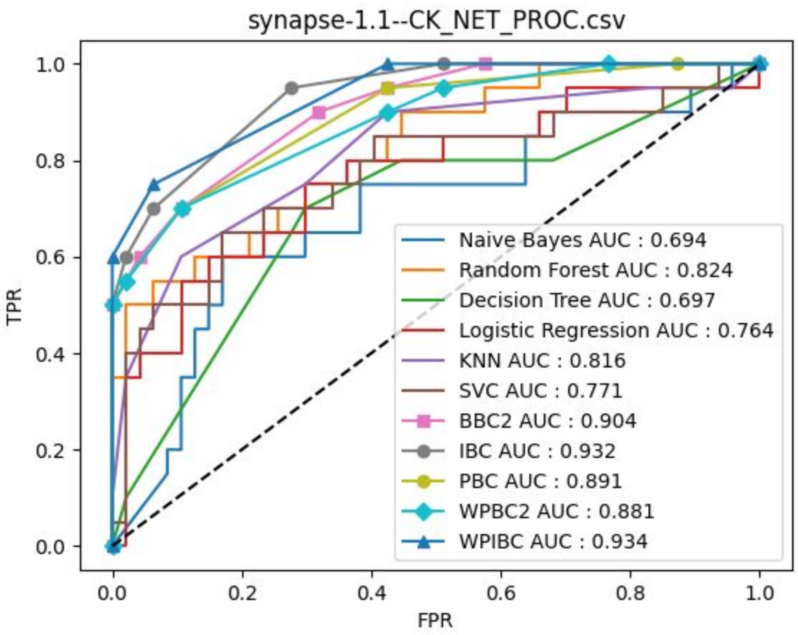
ROC and AUC Results



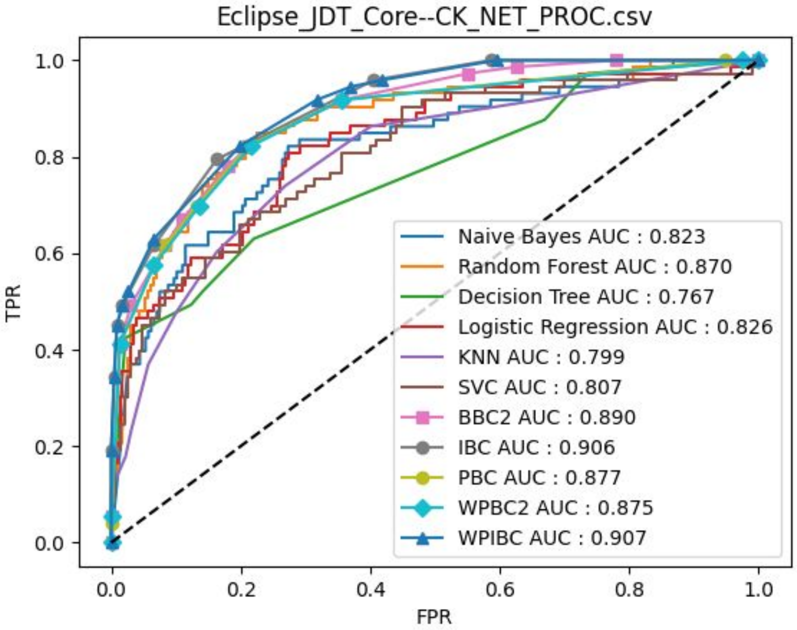
ROC and AUC Results



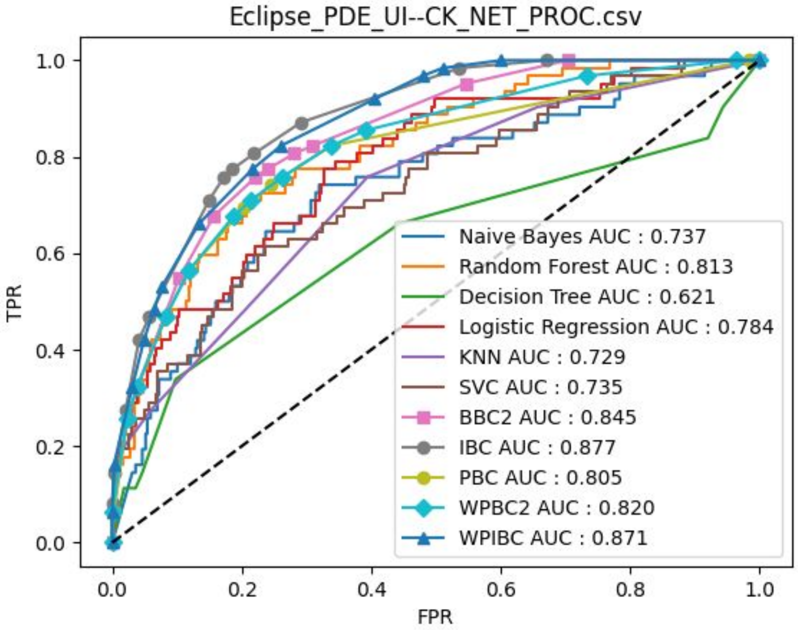
ROC and AUC Results



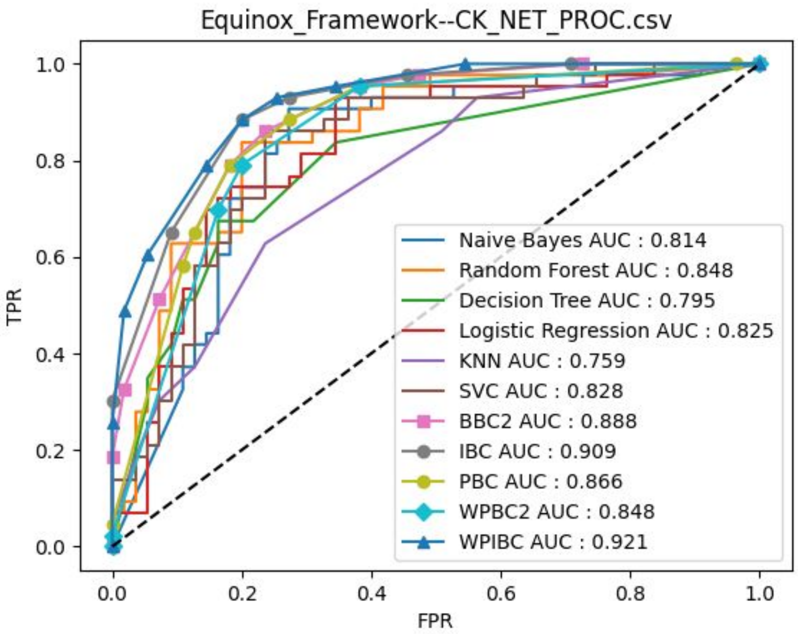
ROC and AUC Results



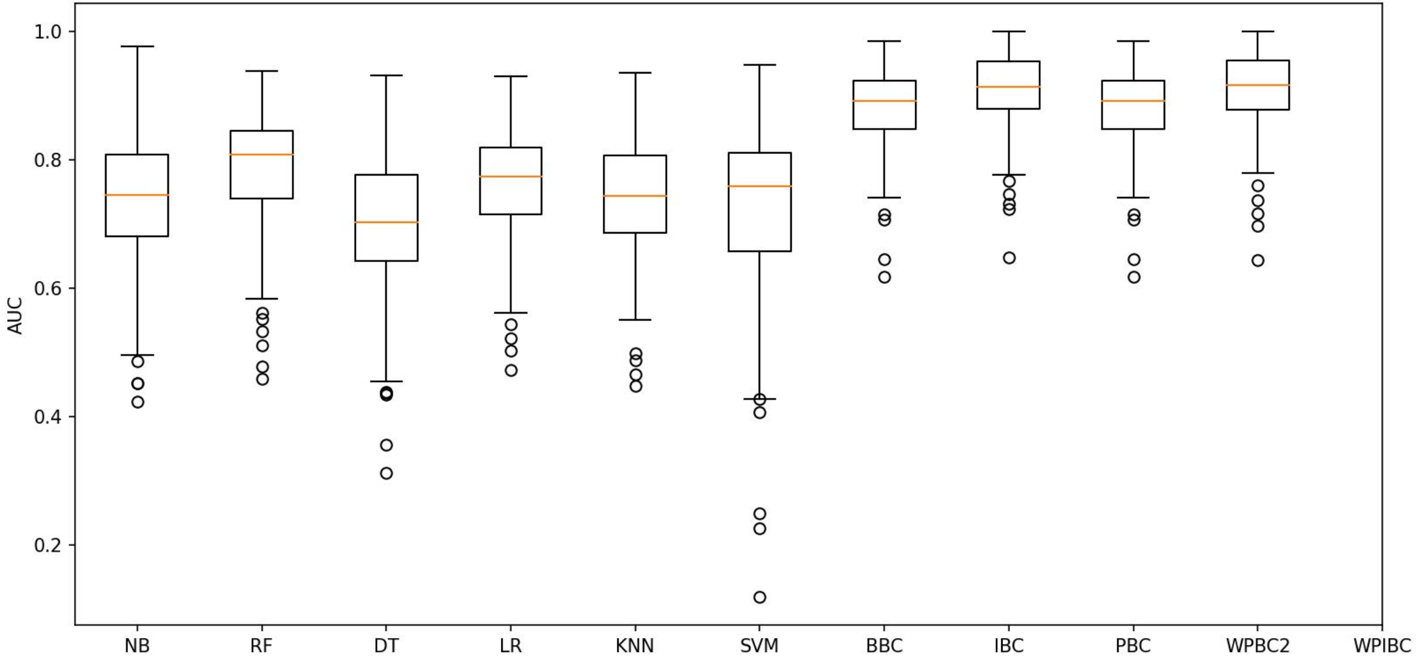
ROC and AUC Results



ROC and AUC Results



Evaluation of AUCs



Conclusions

1. We were able to create BBC, IBC libraries that can be used by anyone across the globe essentially contributing to the open source community
2. We analysed 162 datasets and either of these 4 i.e, BBC, IBC, WPBC2, WPIBC algorithms has yielded a better AUC score than all the heterogeneous models.

References :

1. https://users.encs.concordia.ca/~abdelw/papers/IEERel-Kappa.pdf
2. https://users.encs.concordia.ca/~abdelw/papers/QRS2015-Pruning.pdf
3. https://reader.elsevier.com/reader/sd/pii/S0031320310001263?token=E5FC421ACEB6BCCD05F21DAA19CA397B7B466985184599C6785E51635848E1B0229C8439989C287AA9A1B8384B5AD6BD&originRegion=us-east-1&originCreation=20220719225735
4. https://www.google.com/imgres?imgurl=https%3A%2F%2Fmiro.medium.com%2Fmax%2F948%2F1*4G__SV580CxFj78o9yUXuQ.png&imgrefurl=https%3A%2F%2Ftowardsdatascience.com%2Ftrain-test-split-and-cross-validation-in-python-80b61beca4b6&tbnid=5fK7WhZvCthDHM&vet=12ahUKEwj46Im5gIf5AhUErXIEHei5DMUQMygBegUIARDDAQ..i&docid=2CWqaWOK8QzhyM&w=948&h=493&q=ctrainign%20cross%20validation&ved=2ahUKEwj46Im5gIf5AhUErXIEHei5DMUQMygBegUIARDDAQ

Thank You