# Projeto Final de Programação

Matheus Suknaic

Dezembro, 2020

## 1 Especificação

O projeto de programação final em que nós trabalhamos neste semestre leva o título de : **Approximated born again tree ensembles**. Este projeto é uma extensão do trabalho proposto por Vidal, Schiffer e Pacheco [1] e baseado no trabalho de Breiman e Shang [2].

A proposta desse trabalho é reconstruir uma random forest como uma decision tree, aproveitando a acurácia do método random forest, com a interpretabilidade de uma decision tree. Em seu trabalho, Vidal, Schiffer e Pacheco demonstram que esse problema é NP-hard e propõem algumas soluções exatas e uma solução heurística. Entretanto, a solução proposta exata não é escalável, crescendo exponencialmente no número de features.

Diante desse problema, um possível caminho a ser considerdo é o de reproduzir o feature space de maneira aproximada, e não completamente.Esta foi a motivação usada na implementação do trabalho, através de um algoritmo de programação dinâmica e algumas extensões sobre o código anterior (que pode ser encontrado no seguinte link: https://github.com/vidalt/Born-Again-Random-Forests).

De maneira sucinta, o escopo do trabalho pode ser descrito como a **construção de uma árvore de decisão de uma profundidade máxima K, que maximize a importância (dada por uma função V), e que consiga reproduzir aproximadamente uma random forest recebida como entrada.**

Os requisitos funcionais do programa são:

1. Gerar um arquivo de output contendo a árvore de decisão - que adiante será chama de BA (Born-Again) tree - a partir de uma random forest recebida como entrada.

2. Permitir ao usuário decidir o número de árvores a ser avaliadas na random forest e a seed, caso se escolha a opção heurística.

3. Oferecer ao usuário cinco opções de objetivos ao se construir a born again decision tree: minimizar profundidade, minimizar número de folhas, minimizar número de folhas e profundidade, maximizar a importância dada uma profundidade máxima K e a versão heurística.

4. Quando o objetivo escolhido for maximizar a importnância dada uma profundidade máxima K, o programa deve permitir ao usuário escolher a profundidade máxima desejada e oferecer duas possibilidades de avaliação de células: todas as células com o mesmo valor ou cada célula tendo como valor o número de pontos do data set em seu interior.

5. Avisar ao usuário que para determinadas escolhas de função objetivo e quantidade de features, que não é possível gerar a BA tree e sugerir então a versão heurística do problema nesses casos.

6. Gerar logs indicando as estatísticas essenciais quando a decision tree for gerada, tais como: número de folhas, profundidade da árvore, valor da função objetivo encontrada, entre outros.

Os requisitos não funcionais do programa são:

1. Robustez

2. Manutenibilidade

3. Portabilidade em diferentes sistemas operacionais: Windows, Linux e Mac (Unix).

# 2 Projeto

## 2.1 Estrutura geral do algoritmo

O algoritmo começa recebendo como entrada os seguintes dados: nome da instância, nome do arquivo que será gerado como saída, função objetivo, valor da célula, profundidade desejada e seed.

A função objetivo permite ao usuário escolher entre cinco algoritmos diferentes a partir dos quais a árvore será construída. As três primeiras opções conseguem construir uma árvore que representa fidedignamente o feature space, porém com diferentes objetivos. A primeira opção minimiza a profundidade da árvore, a segunda minimiza o número de folhas e a terceira usa ambos estes critérios simultaneamente. A quarta e quinta opções constroem essa árvore de maneira aproximada, ou seja, não representam exatamente todo o feature space. A quarta opção é uma versão heurística; e a quinta, uma em que o usuário passa de antemão a profundidade da árvore desejada e o algoritmo então retorna a melhor árvore encontrada de no máximo essa profundidade.

Para esta última opção mencionada, o usuário também tem que escolher o valor da célula, além da profundidade da árvore desejada. O valor das células se refere ao modo como as células serão avaliadas, existindo duas possibilidades: todas as células têm o mesmo valor ou o valor de cada célula é definido pelo número de pontos do data set dentro de cada célula.

Após receber esses dados, o programa os armazena e constrói sua representação interna da random forest. Esta random forest, junto com o objetivo selecionado pelo usuário, são utilizados como entrada para a programação dinâmica que constrói a BA tree.

Antes de entender como a construção da árvore é feita, é necessário determinar a definição de dois termos: célula e região. Uma célula é uma unidade que representa o espaço delimitado entre as interseções dos hiperplanos, enquanto uma região é um conjunto de células, descrito sempre pela célula mais à esquerda e abaixo e pela célula mais à direita e acima.

A construção dessa nova árvore tem duas possibilidades: a versão exata e a heurística. Caso seja escolhida a versão exata (objetivos 1, 2, 3 e 5), o algoritmo começa enumerando todas as células que constituem o feature space. Caso o usuário escolha a versão heurística, o algoritmo, em vez de enumerar todas as células, seleciona um grupo para representar a região avaliada na iteração. Para cada célula, o algoritmo atribui uma classe e, se o objetivo escolhido for o quinto, atribui também a ela um valor.

Após todas as células estarem com suas classes e valores definidos, o algoritmo filtra os hiperplanos do feature space que não são utilizáveis, desta maneira reduzindo o número de células e regiões. A intuição por trás desse processo é remover hiperplanos que separam células vizinhas da mesma classe, uma vez que não faz sentido separar essas células por um hiperplano quando elas podem ser classificadas no mesmo grupo. O número reduzido de regiões e células permite também ao algoritmo funcionar de maneira mais eficiente. Depois da separação, as regiões do feature space são criadas e o processo de construção da BA tree é inicializado.

Os objetivos 1, 2 e 3 recebem como entrada todas as células do feature space e utilizam uma programação dinâmica para recursivamente percorrer as células e armazenar as regiões. Para cada feature e para cada hiperplano, a região é dividida em duas sub-regiões: uma da esquerda, delimitada pelo o hiperplano mais à esquerda da região original e pelo hiperplano considerado no momento; e umaa região da direita, delimitada pelo hiperplano considerado no momento e pelo hiperplano mais à direita da região original. À partir disso, o algoritmo com os objetivos 1, 2 ou 3 permanece realizando a recursão até que as sub-regiões da esquerda e da direita sejam a mesma célula ou que a região esteja armazenada na memória da programação dinâmica. Quando o algoritmo chega nesse caso em que as sub-regiões da esquerda e da direita são a mesma célula, ele confere se as células são da mesma classe, agrupando-as, neste caso, em uma folha, e nao o fazendo caso não sejam. Quando a recursão retorna deste caso base, escolhe-se entre as duas sub-regiões a que tem o melhor valor. O valor é então atribuído para a região original.

O quinto objetivo funciona de maneira similar, possuindo também ambos os casos base mencionados anteriormente. Porém, quando as recursões retornam do caso base , os valores da sub-região da esquerda e da direita são somados. Além disso, ele possui outro caso base em que quando a profundidade limite é atiginda, o algoritmo itera nas células daquela região, vendo qual das duas classes possui o maior valor e atribuindo essa classe e valor à região.

O objetivo 4, que representa a versão heurística do problema, funciona de uma maneira diferente. Um conjunto de células é selecionado para representar uma região e escolhe-se o hiperplano que maximiza o ganho de informação. Se todas as células pertencerem à mesma classe, um solver de programas inteiros é utilizado para averiguar se essa região é homogênea ou não. Caso seja, todas essas células são agrupadas em uma folha, caso contrário, o algoritmo continua dividindo a região em duas sub-regiões até que elas sejam homogêneas. Por fim, com o conjunto de regiões definido, o algoritmo constrói a árvore de maneira recursiva recebendo todas as células do feature space e consultando na memória do DP para cada região o valor ótimo obtido.

## 2.2 Classes

O programa é constituído de seis classes:

1. **Commandline**: Lê e verifica todos os argumentos passados como entrada para o programa.
2. **RandomForest**: Cria a estrutura de dados que representa a random forest, além da estrutura de dados auxiliar que armazena as informações sobre os hiperplanos. Disponibiliza métodos para manipulação da estrutura de dados que representa a random forest.
3. **BornAgainDecisionTree**: Constrói a BA tree à partir da random forest e da função objetivo escolhida. Além disso, gera os dados estatísticos relacionados à construção desta árvore.

4. **FSpace (Feature Space)**: Representa o feature space da random forest passada como entrada e cria a estrutura de dados que representa as células utilizadas para a construção da BA tree. Disponibiliza métodos para manipulação da estrutura de dados que representa as células.

5. **Parameters**: Lê os argumentos passados como entrada e os armazena para serem utilizados ao longo do algoritmo.

6. **MIPCertificate**: Constrói e executa um programa inteiro linear misto que auxilia na construção da BA tree, quando a função objetivo escolhida é a heurística.

## 2.3 Modelo UML

Abaixo encontra-se o modelo UML ilustrando a relação entre as classes:

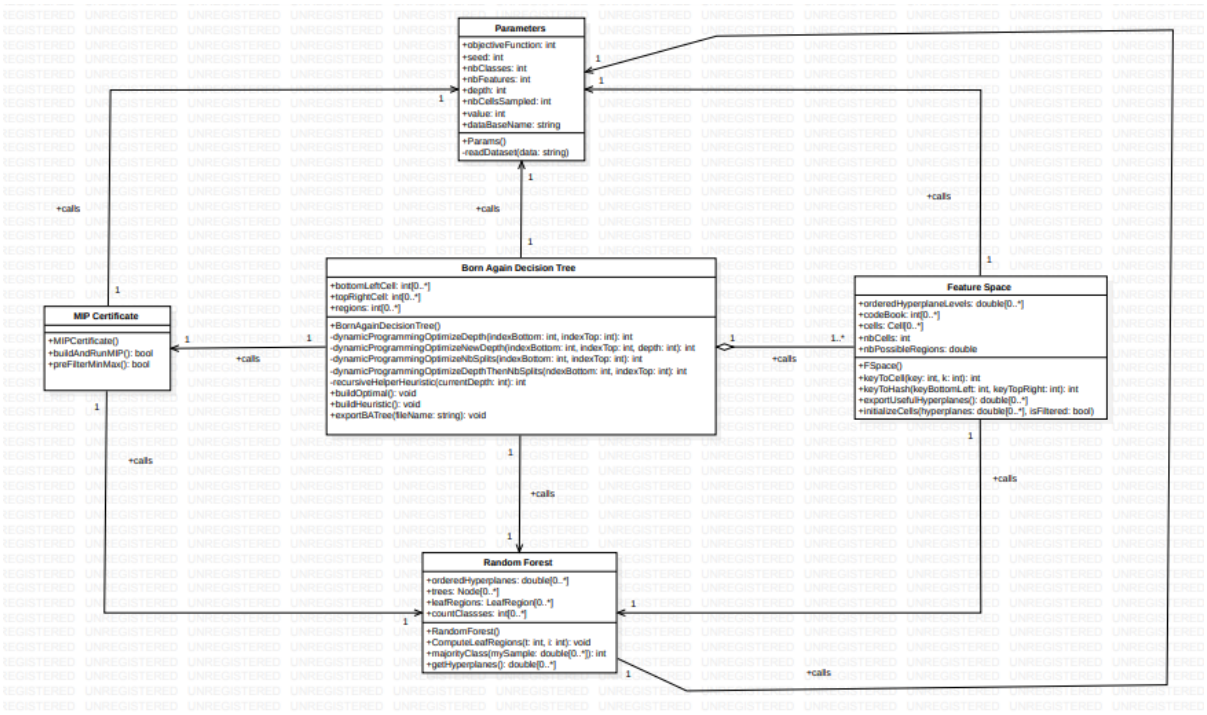
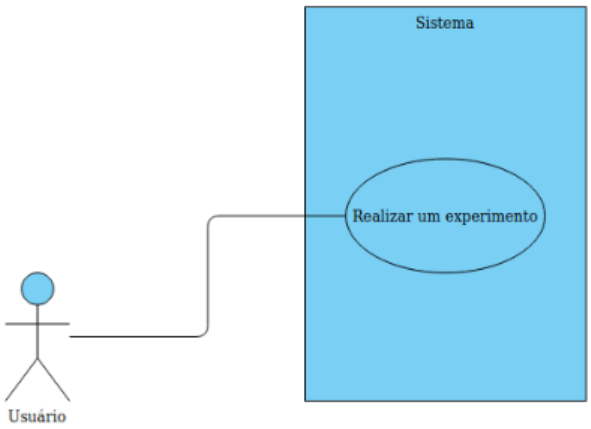



Figure 1: Modelo UML

Para uma melhor visualização, consultar o pdf **model** incluso na documentação.

## 2.4 Casos de Uso

Abaixo encontram-se os casos de uso do programa:




| | |
|---|---|
| Nome | Realizar um experimento |
| Atores | Usuário do framework |
| Objetivo | Utilizar o algoritmo born Again Decision Tree para realizar um experimento |
| Fluxo Principal | 1. O usuário inicializa o framework.<br>2. O usuário escolhe os parâmetros desejados.<br>3. O usuário recebe uma árvore que representa a random forest passada como entrada pro algoritmo. |

Figure 2: Casos de uso

# 3 Experimentos computacionais

Os objetivos dos testes realizados foram os seguintes:

1. Provar a corretude da nova programação dinâmica, que tem como objetivo maximizar a importância da BA tree gerada, dada uma profundidade limite K.

2. Analisar como a redução desde a profundidade "ideal" da árvore até zero (a raiz é uma folha) impacta nas métricas de acurácia e F1 measure.

O algoritmo de programação dinâmica foi implementado em C++ e compilado com GCC usando a flag -O3, os scripts de teste e as random forests usadas como input foram geradas em python (usando a biblioteca scikit - learn v.0.22.1). Todos os experimentos rodaram em uma thread em um computador com as seguintes especificações: CPU Intel(R) Core(TM) i7-3537U 2.00GHz e memória RAM de 8GB.

Os testes realizados foram comparados aos resultados gerados no trabalho de Vidal, Schiffer e Pacheco. Para isso, utilizamos os mesmos datasets desse trabalho e as random forests geradas. Para mais informações sobre a geração das random forests e dos datasets, recomendamos uma leitura aprofundada do artigo [1].

## 3.1 Testes sobre a corretude

O primeiro teste realizado foi comparar se as árvores encontradas utilizando o algoritmo de Vidal, Schiffer e Pacheco para o objetivo da minimização da altura da BA tree seriam idênticos

a utilizar a nova programação dinâmica implementada. A comparação foi feita utilizando o objetivo em que todas as células têm o mesmo valor e a profundidade limite passada foi a encontrada pelo algoritmo quando utilizada como objetivo a minimização da profundidade da BA tree. A motivação por trás deste experimento foi confirmar a corretude da implementação desse novo objetivo, pois com os paramêtros escolhidos ele é idêntico ao objetivo que minimiza a profundidade da BA tree.

As BA trees foram geradas através de um shell script, onde para cada um dos três datasets(HTRU2,COMPAS-ProPublica,FICO), utilizaram-se todas as 10 random forests disponíveis. Para cada uma das random forests, escolheu-se o número de árvores igual a 10, o objetivo igual a 5 (novo objetivo a ser testado), os dois tipos de valores para célula (0 e 2) e proufnidade ideal de cada árvore para cada instância (de acordo com a saída quando utilizado o objetivo de minimização da profundidade). No total, foram geradas **60** instâncias.

Para realizar esse teste, um script foi feito em python. Nós conseguimos executar esses script apenas para três (HTRU2, COMPAS-ProPublica, FICO) dos seis datasets utilizados. As instâncias dos três outros datasets demonstraram ser mais complexas e o algoritmo não conseguiu encontrar uma solução dentro de um limite aceitável de tempo.

As entradas desse algoritmo são : a random forest, a BA tree gerada através do objetivo de minimização da profundidade e a BA tree gerada através do objetivo de minimização da profundidade com uma profundidade limite. A saída do programa é um arquivo excel que contém o método utilizado: random forest, objetivo de minimização da profundidade da BA tree ou o novo objetivo implementado; a acurácia e F1 measure no conjunto de treinamento e teste. Esses resultados vêm em triplas, cada um com um dos três métodos mencionados anteriormente. Abaixo podemos ver um exemplo:

| | A | B | C | D | E | F | G |
|---|---|---|---|---|---|---|---|
| 1 | | | **Train-Acc** | **Train-F1** | **Test-Acc** | **Test-F1** | |
| 2 | **0** | RandomForest | 0,6632882883 | 0,6597350191 | 0,6772793054 | 0,6748660214 | COMPAS-Probublica.RF1 |
| 3 | **1** | BornAgain | 0,6632882883 | 0,6597350191 | 0,6772793054 | 0,6748660214 | |
| 4 | **2** | BornAgainNew | 0,6632882883 | 0,6597350191 | 0,6772793054 | 0,6748660214 | |
| 5 | **3** | RandomForest | 0,6547619048 | 0,6488587713 | 0,6700434153 | 0,6624097593 | COMPAS-Probublica.RF2 |
| 6 | **4** | BornAgain | 0,6547619048 | 0,6488587713 | 0,6700434153 | 0,6624097593 | |
| 7 | **5** | BornAgainNew | 0,6547619048 | 0,6488587713 | 0,6700434153 | 0,6624097593 | |
| 8 | **6** | RandomForest | 0,6676319176 | 0,6664189137 | 0,6599131693 | 0,660663975 | COMPAS-Probublica.RF3 |
| 9 | **7** | BornAgain | 0,6676319176 | 0,6664189137 | 0,6599131693 | 0,660663975 | |
| 10 | **8** | BornAgainNew | 0,6676319176 | 0,6664189137 | 0,6599131693 | 0,660663975 | |
| 11 | **9** | RandomForest | 0,6616795367 | 0,6597528977 | 0,6584659913 | 0,6547620015 | COMPAS-Probublica.RF4 |
| 12 | **10** | BornAgain | 0,6616795367 | 0,6597528977 | 0,6584659913 | 0,6547620015 | |
| 13 | **11** | BornAgainNew | 0,6616795367 | 0,6597528977 | 0,6584659913 | 0,6547620015 | |
| 14 | **12** | RandomForest | 0,6595881596 | 0,6529291599 | 0,6034732272 | 0,5941902441 | COMPAS-Probublica.RF5 |
| 15 | **13** | BornAgain | 0,6595881596 | 0,6529291599 | 0,6034732272 | 0,5941902441 | |
| 16 | **14** | BornAgainNew | 0,6595881596 | 0,6529291599 | 0,6034732272 | 0,5941902441 | |
| 17 | **15** | RandomForest | 0,6642535393 | 0,6594519682 | 0,6584659913 | 0,6556462253 | COMPAS-Probublica.RF6 |
| 18 | **16** | BornAgain | 0,6642535393 | 0,6594519682 | 0,6584659913 | 0,6556462253 | |
| 19 | **17** | BornAgainNew | 0,6642535393 | 0,6594519682 | 0,6584659913 | 0,6556462253 | |
| 20 | **18** | RandomForest | 0,6557271557 | 0,6496587637 | 0,6526772793 | 0,6384220026 | COMPAS-Probublica.RF7 |
| 21 | **19** | BornAgain | 0,6557271557 | 0,6496587637 | 0,6526772793 | 0,6384220026 | |
| 22 | **20** | BornAgainNew | 0,6557271557 | 0,6496587637 | 0,6526772793 | 0,6384220026 | |

Figure 3: Exemplo do arquivo excel de saída

Como podemos observar, a cada tripla, todos os resultados possuem os mesmos valores para a acurácia e F1 measure no conjunto de treino e de teste. A única instância onde isso não acontece é a HTRU2.RF6. Neste caso, temos uma divergência nos valores. Essa divergência é, entretanto, facilmente explicada: no caso em que duas ou mais classes têm o mesmo valor para determinada região, o algoritmo sempre escolhe a primeira delas. Podemos, portanto, estar classificando algum sample incorretamente, pois qualquer uma dessas classes tem o mesmo valor

e estamos considerando apenas a primeira. Como podemos observar abaixo, onde alteramos o algoritmo para que ele selecione a última classe em vez da primera. Quando fazemos isso, os resultados coincidem.

| | | | | | | |
|---|---|---|---|---|---|---|
| 75 | RandomForest | 0,9770921281 | 0,9766453641 | 0,9770949721 | 0,9770548791 | |
| 76 | BornAgain | 0,9770921281 | 0,9766453641 | 0,9770949721 | 0,9770548791 | |
| 77 | BornAgainNew | 0,9767817234 | 0,9762562097 | 0,9770949721 | 0,9770548791 | HTRU2.RF6 |

Figure 4: Instância HTRU2.RF6 com resultados diferentes

| | | | | | | |
|---|---|---|---|---|---|---|
| 75 | RandomForest | 0,9770921281 | 0,9766453641 | 0,9770949721 | 0,9770548791 | |
| 76 | BornAgain | 0,9770921281 | 0,9766453641 | 0,9770949721 | 0,9770548791 | |
| 77 | BornAgainNew | 0,9770921281 | 0,9766453641 | 0,9770949721 | 0,9770548791 | HTRU2.RF6 |

Figure 5: Instância HTRU2.RF6 com resultados iguais

Por fim, para comprovarmos a corretude do nosso método, podemos comparar a média da acurácia e F1 measure no conjunto de testes e ver se todas são idênticas. Neste caso, a única diferença ocorre no dataset HTRU2 pela razão explicada acima. Abaixo, podemos observar os valores obtidos quando colocamos todos os valores células iguais (value = 0) e quando colocamos os valores de cada célula para ser igual ao número de pontos em seu interior (value = 2).

```
Average RF Accuracy and F1 in COMPAS-ProPublica with value 0 : 0.6607306889255578 0.65479892532933
Average BA Accuracy and F1 in COMPAS-ProPublica with value 0 : 0.6607306889255578 0.65479892532933
Average BANew Accuracy and F1 in COMPAS-ProPublica with value 0 : 0.6607306889255578 0.65479892532933

Average RF Accuracy and F1 in FICO with value 0 : 0.7143874338024896 0.6964763064389234
Average BA Accuracy and F1 in FICO with value 0 : 0.7143874338024896 0.6964763064389234
Average BANew Accuracy and F1 in FICO with value 0 : 0.7143874338024896 0.6964763064389234

Average RF Accuracy and F1 in HTRU2 with value 0 : 0.9770986383687994 0.9769636304901447
Average BA Accuracy and F1 in HTRU2 with value 0 : 0.9770986383687994 0.9769636304901447
Average BANew Accuracy and F1 in HTRU2 with value 0 : 0.9770675978920176 0.9769636304901447

Average RF Accuracy and F1 in COMPAS-ProPublica with value 2 : 0.6607306889255578 0.65479892532933
Average BA Accuracy and F1 in COMPAS-ProPublica with value 2 : 0.6607306889255578 0.65479892532933
Average BANew Accuracy and F1 in COMPAS-ProPublica with value 2 : 0.6607306889255578 0.65479892532933

Average RF Accuracy and F1 in FICO with value 2 : 0.7143874338024896 0.6964763064389234
Average BA Accuracy and F1 in FICO with value 2 : 0.7143874338024896 0.6964763064389234
Average BANew Accuracy and F1 in FICO with value 2 : 0.7143874338024896 0.6964763064389234

Average RF Accuracy and F1 in HTRU2 with value 2 : 0.9770986383687994 0.9769636304901447
Average BA Accuracy and F1 in HTRU2 with value 2 : 0.9770986383687994 0.9769636304901447
Average BANew Accuracy and F1 in HTRU2 with value 2 : 0.9770986383687994 0.9769636304901447
```

Figure 6: Média de cada método em relação à acurácia e F1 measure.

```
Average RF Accuracy and F1 in HTRU2 with value 0 : 0.9770986383687994 0.9769636304901447
Average BA Accuracy and F1 in HTRU2 with value 0 : 0.9770986383687994 0.9769636304901447
Average BANew Accuracy and F1 in HTRU2 with value 0 : 0.9770986383687994 0.9769636304901447
```

Figure 7: Média com HTRU2.RF6 coincidindo.

### 3.2 Testes sobre a redução da profundidade da BA tree

O segundo teste realizado foi reduzir a profundidade da BA tree obtida para cada instância, partindo da profundidade "ideal" (encontrada utilizando o objetivo que minimiza a altura da BA tree) até zero. Fizemos isso para ver o quanto a profundidade da árvore impactava nas métricas acurácia e F1 measure.

As BA trees foram geradas através de um shell script, onde para cada um dos três datasets(HTRU2,COMPAS-ProPublica,FICO), utilizaram-se todas as 10 random forests disponíveis. Para cada uma das random forests, escolheu-se o número de árvores igual a 10, o objetivo igual a 5 (novo objetivo a ser testado), os dois tipos de valores para célula (0 e 2) e profundidade começando com a profundidade ideal de cada árvore para cada instância (de acordo com a saída quando utilizado o objetivo de minimização de altura) até a profundidade se igualar a 0. No total, foram geradas **514** instâncias.

Para realizar o teste, um script foi feito em python. Nós apenas conseguimos rodar também para os três datasets mencionados anteriorment. A entrada desse algoritmo é a BA tree gerada através do novo objetivo. A saída do programa é um arquivo .txt no qual linha a linha temos: a altura da árvore; a acurácia e F1 measure no conjunto de testes; e um arquivo .png contendo um gráfico, onde o eixo horizontal representa a altura da árvore e o eixo vertical representa o percentual da acurácia e da F measure no conjunto de testes. Abaixo, temos um exemplo dos arquivos de saída para uma instância:

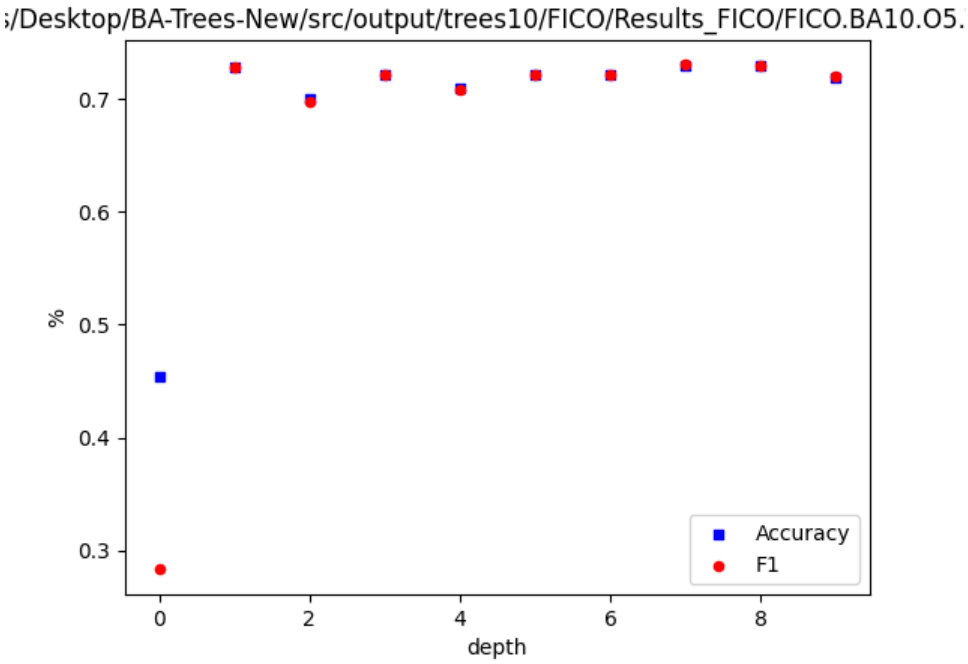



Figure 8: Arquivo .png gerado para instância FICO.BA10.T10.O5.V0

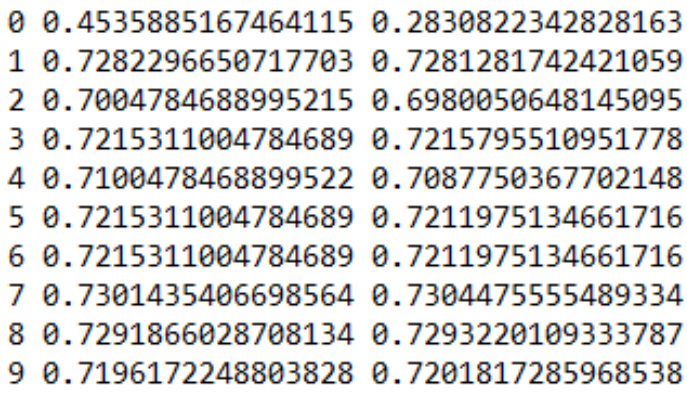



Figure 9: Arquivo .png gerado para instância FICO.BA10.T10.O5.V0

A partir desses resultados, conseguimos observar dois tipos de comportamento:

1. Reduzir a profundidade da BA tree resultante em alguns casos faz com que a acurácia e F1 measure diminuam. A perda dessas duas métricas não é tão significativa, a menos que essa redução seja considerável, como por exemplo, ao permitir que a BA tree tenha profundidade de apenas um ou zero.

2. Reduzir a profundidade da BA tree resultante, em alguns casos e até certo ponto, aumenta a acurácia e F1 measure. Se reduzimos um pouco o valor da profundidade ideal, conseguimos uma aumento pequeno nos valores da F measure e acurácia. Isso nos indica que as BA tree geradas com essa profundidade "ideal" na verdade estão com o problema de overfit.

   Mais análises ainda precisam ser feitas para se ter uma conclusão mais aprofundada de ambos esses comportamentos, porém os resultados preliminares nos indicam que é possível construir árvores menores sem ter grandes perdas de valores nessas métricas, como podemos observar nos dois exemplos abaixo:

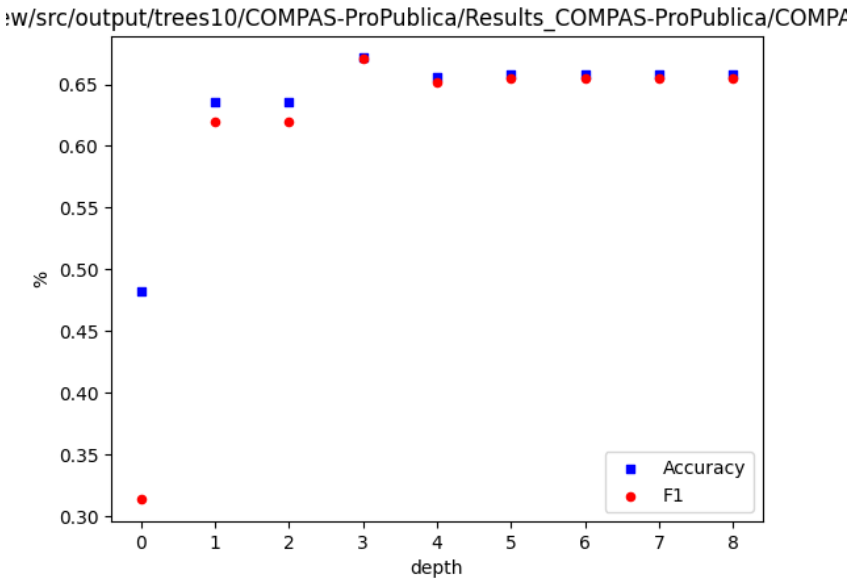



Figure 10: Instância COMPAS-ProPublica.BA4.T0.V0

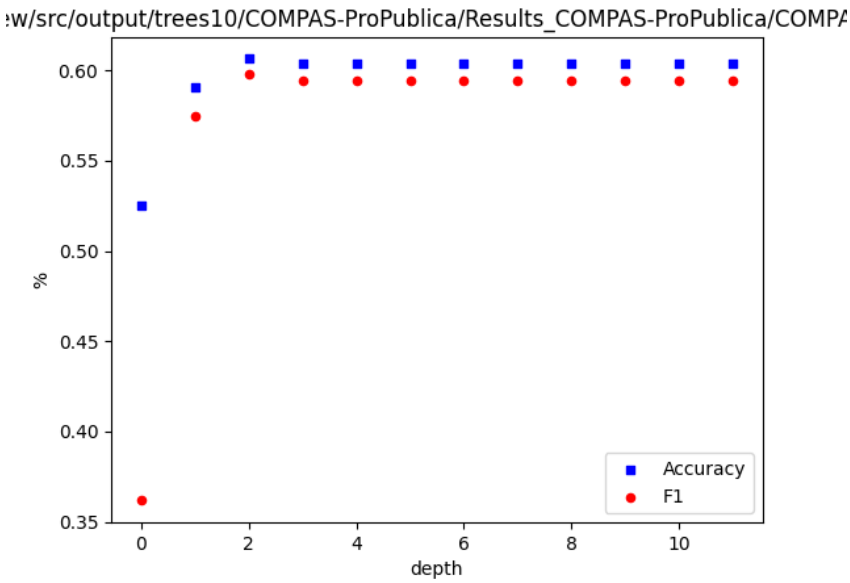



Figure 11: Instância COMPAS-ProPublica.BA5.T10.V2

# 4 Manual do usuário

Para executar o programa primeiro é necessário utilizar o comando **make** no diretório **born again dp** para que os executavéis sejam criados.

Após isso se o usuário desejar rodar uma instância específica ele deve utilizar: ./bornAgain (arquivo da instancia de entrada) (arquivo da saída) -trees (número de árvores a serem avaliadas da random forest) -obj (número da função objetivo) -seed (número da seed) -value (número do tipo de avaliação da célula) e -depth (limite de profundidade da árvore).

Um exemplo de chamada contendo todos esses parâmetros seria:./bornAgain ../resources /forests/HTRU2/HTRU2.RF8.txt ../output/trees10/HTRU2/HTRU2.BA8.O5.T10.V0 /HTRU2.BA8.T10.V0.D6 -trees 10 -obj 5 -seed 0 -value 0 -depth 6.

Entretanto, não é necessário que o usuário informe todos os parâmetros. Os únicos necessários são o arquivo de entrada e arquivo de saída. Os outros parâmetros já começam com um valor inicial default caso o usuário não os utilize.

Além disso, existem os seguintes scripts dentro desse diretório:

1. **runAllTrees10.sh**: Cria o diretório trees10, os diretórios para cada um dos três datasets (COMPAS-ProPublica, FICO e HTRU2) e gera para cada random forest uma BA tree utilizando 10 árvores e como função objetivo a minimazação da profundidade da árvore. As BA trees tem dois tipos de arquivo: o .tree possui a árvore gerada e suas informaçãoes e o arquivo .out contém as informações referentes a execução do programa como: número de folhas da árvore, altura, função objetivo e etc.

2. **runNewDepthExact.sh**: Cria o diretório ExactDepth dentro de cada um dos diretórios dos datasets, para cada random forest gera uma BA tree utilizando 10 árvores, função objetivo que maximize a importância da BA tree dada uma profundidade máxima, ambos os valores de avaliação da célula e a profundidade "ideal".

3. **runNewDepth.sh**: Similar ao script anterior, porém executa da profundidade ideal de uma árvore até a profundidade zero. Cria dentro de cada diretório de dataset, um diretório para cada random forest, com o número de árvores, função objetivo e valor da célula escolhidos.

4. **runAllDataSets.sh**: Executa todas as instâncias para todos os datasets, utilizando as funções objetivo de minimização da profundidade, minimização do número de folhas e minimização do número de folhas e profundidade da BA tree.

5. **extracttrees10.sh**: Coleta todas as profundidades ideais dos três datasets mencionados anteriormente e armazena em um arquivo chamado "depths.txt", para o caso em que a função objetivo escolhida foi a minimização da altura. É necessário executar esse script antes de executar o script runNewDepthExact.sh e runNewDepth.sh.

   Para rodar os testes mencionados na sessão de experimentos computacionais, primeiro execute os scripts: runAlltrees10.sh, extracttrees10.sh, runNewDepthExact.sh e runNewDepth.sh. Após isso, basta executar o arquivo main.py disponibilizado no diretório **src**, apenas sendo necessário trocar nas linhas 18 e 19 os caminhos onde se encontram o diretório **/BA-Trees-New/src** e **/BA-Trees-New/src/output** e a linha 66 na classe Params.cpp para indicar o caminho dos datasets.

# 5 Referências

[1] - T. Vidal, T. Pacheco, and M. Schiffer, "Born-again tree ensembles,"arXiv preprint arXiv:2003.11132,2020

[2] - Breiman, L. and Shang, N. Born again trees. Technical report, University of California Berkeley, 1996