Floresta de árvores aleatórias no contexto da estrutura de dados e algoritmos

Ewerton L. Costadelle[1,2], **Rafael N. de Carvalho**[1,2]

[1]Instituto de Computação (UFF)
Av. Gal. Milton Tavares de Souza, s/nº, São Domingos - Niterói, RJ – Brasil

[2]Instituto Federal de Educação, Ciência e Tecnologia de Rondônia
Av. Lauro Sodré, 6500 - Censipam - Aeroporto, Porto Velho,RO – Brazil

ecostadelle@id.uff.br, rafaelnink@id.uff.br

Resumo. *A popularidade das árvores de decisão é atribuída a sua estrutura simples e bastante compreensível. Porém o problema do sobreajuste exige que outros métodos sejam empregados. A utilização de algoritmos híbridos tem sido bem significativos, principalmente pela boa acurácia na previsão, velocidade de computação e ampla disponibilidade de software. Algoritmos, como o* Random Forest *combinam outros algoritmos, a exemplo do* Bagging *e do subespaço aleatório para classificação ou regressão em aprendizado de máquina. Esse trabalho apresenta um exemplo de uso do Random Forest como classificador, bem como detalha alguns aspectos importantes da implementação. O código fonte e as fontes de dados estão disponíveis no GitHub.*

1. Introdução

A pesquisa em árvores de classificação e regressão teve um rápido crescimento e as aplicações aumentaram a uma taxa ainda maior. A popularidade das árvores é atribuída a sua estrutura simples e bastante compreensível. Outro fator é que possuem razoavelmente boa acurácia na previsão, velocidade de computação e ampla disponibilidade de software [Loh 2014]. O ponto negativo das árvores é que elas podem se ajustar quase perfeitamente aos dados de treinamento e ficar pouco eficientes para a generalização. Esse é um conhecido problema de sobreajuste e para evitá-lo, costuma-se fazer a ”poda”da árvore [Grus 2019].

Outra possibilidade que tem ganhado destaque é a utilização de algoritmos híbridos. Recentemente, devido a evolução do poder computacional, os algoritmos *ensembles* despertaram o interesse de pesquisadores da área da aprendizagem de máquina. Essa significância deve-se ao fato de que o híbridos combinam vários outros algoritmos para produzir uma única previsão. Dentre os quais, pode-se destacar o *Random Forest*, ou Floresta de Árvores Aleatórias. Que é um híbrido do algoritmo *Bagging* e do método de subespaço aleatório e usa árvores de decisão como classificador [Buhmann et al. 2011].

O objetivo deste trabalho é apresentar um exemplo de uso do *Random Forest* como classificador, bem como detalhar alguns aspectos importantes da implementação. Pretende-se que o leitor possa entender o funcionamento dos algoritmos que compões esse hibrido.

2. Antes da floresta havia a árvore

Árvores de decisão são estruturas muito simples e com alto grau de interpretabilidade. Esse método, utilizado tanto para classificação quanto para regressão, foi proposto há quase 60 anos por [Morgan and Sonquist 1963] e desde então teve muitas variações.

Basicamente, uma árvore é uma estrutura hierárquica que particiona dados de modo a torná-los mais homogêneos possível. Ela subdivide um conjunto amostral com base nos seus atributos. É mais intuitiva a seleção de atributos categóricos, porém, atributos numéricos carecem de uma definição do melhor ponto para fazer o corte. No exemplo da Figura 1 o nó raiz particiona o conjunto de flores pela largura de suas pétalas. Ainda no exemplo, serão direcionadas ao nó da esquerda aquelas que tiverem largura menor que 2.45. Consequentemente, larguras maiores ou iguais a 2.45 serão direcionadas ao nó da direita, de modo que novos particionamentos podem ser feitos, até que nós folhas sejam tão homogêneos quanto o desejado. E é majoritariamente, no modo como o algoritmo faz esse particionamento é que define qual é o seu modelo.

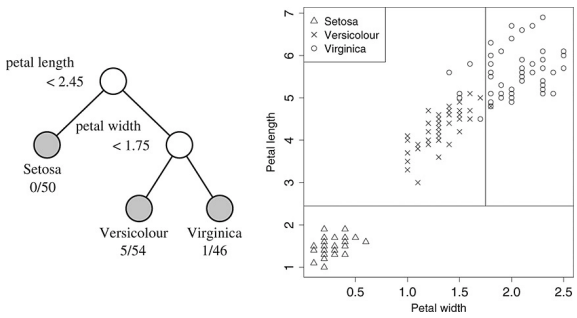

Figura 1. Exemplo de uma árvore de decisão. Fonte [Loh 2014]

O modelo AID (*Automatic Interaction Detection*) proposto por [Morgan and Sonquist 1963], varre o vetor do atributo sempre separando o conjunto em dois. A cada particionamento é calculado o erro pelo método dos mínimos quadrados. O ponto de partição que obtiver menor erro é o selecionado. A seleção da raiz de uma subárvore também é baseada no erro quadrático médio, após obter o ponto de corte e o erro de todos os atributos, seleciona-se a raiz. O processo é iterado até que se atinja condições de seleção definidas previamente.

Outros modelos acrescentaram padrões utilizados até os dias de hoje, o THAID (*THeta Automatic Interaction Detection*) [Messenger and Mandell 1972] introduziu os conceitos de entropia e índice de Gini na seleção do nó raiz. E o CART (*Classification And Regression Trees*), proposto por [Breiman et al. 1984], utilizou divisões em combinações lineares de variáveis, por busca estocástica e resolveu problemas de valores de dados ausentes em um nó.

O algoritmo *Iterative Dichotomiser* 3 (ID3) introduzido por [Quinlan 1986] é elaborado a partir de uma sistemas de inferência, gerando uma árvore da raiz às folhas usando o ganho de informação para definir a melhor raiz, de forma recursiva encontra-se o melhor filho, ou seja, o atributo que melhor divide o conjunto de dados, seguindo até as folhas. Possui limitação quanto aos tipos de dados, sendo necessário que este sejam categóricos não ordinais, desta forma o C4.5 apresenta-se como uma evolução trabalhando com dados

contínuos, além de tratar dados desconhecidos, ou seja, não presente nos dados de testes, ainda apresenta um método de pós-poda para avaliar desempenho.

2.1. Das métricas

Diversas métricas são comumente utilizadas na construção das árvores de decisão, como: grau de entropia, ganho de informação

O grau de entropia, dada uma amostra(S) com resultados positivos (+) e negativos (–) de algum conceito alvo, para uma classificação booleana definido pela expressão é

$$Entropia(S) \equiv -p_+ \log_2 p_+ - p_- \log_2 p_- \tag{1}$$

tendo como resultados no intervalo $[0, 1]$, quanto mais próximos de 1, menor a dúvida na classificação.

De forma genérica, se atributo alvo aceitar c diferentes valores, a entropia de S relativa a esta classificação c–classes é definida como:

$$Entropia(S) \equiv \sum_{i=1}^{c} -p_i \log_2 p_i \tag{2}$$

sendo p_i a proporção de S pertencendo a classe i.

Neste contexto a entropia refere-se a aleatoriedade de uma variável prever a classe, enquanto a métrica ganho de informação representa a redução da entropia causada pela divisão dos exemplos de acordo com os valores do atributo, sendo expressa:

$$Gain(S, A) \equiv Entropia(S) - \sum_{v \in valores(A)} \frac{|S_v|}{|S|} Entropia(S_v) \tag{3}$$

redução esperada na entropia causada gerada pela divisão dos exemplos de acordo com este atributo A.

2.2. Submétodos combinados para formar a floresta

O *Random Forest*, é um algoritmo híbrido, que além de árvores de decisão, utiliza de métodos de agregação e a subdivisão do espaço amostral. Utiliza o *Bagging*, que é um método de estimação de parâmetros com alto poder de extrair informação dos dados. Nesse contexto, seria um erro traduzir *bagging* com qualquer variação do verbo empacotar, principalmente porque pouco teria a ver com a ideia que o motiva. De fato, o autor [Breiman 1996] o descreveu como o acrônomio de ***b**ootstrap **agg**regat**ing***, ou seja, é um agregador de preditores baseados na técnica *bootstrap* de reamostragem.

Por sua vez, o *bootstrap* é uma técnica de reamostragem proposta por [Efron and Tibshirani 1979] que consiste em remover e repor aleatoriamente uma porcentagem de amostras de um conjunto de dados para estimar alguns parâmetros. Com base no novo conjunto de dados, problemas como a falta de normalidade são contornados, permitindo a inferência estatística e determinação de parâmetros, como intervalos de confiança, de modo analítico. É um método estatístico que ficou popular com o aumento do poder computacional, principalmente a partir da década de 1990.

No código, abaixo, um conjunto de dados com tamanho amostral $n = 6$ foi reamostrado $N_{boot} = 400$ vezes. O sorteio foi feito utilizando o gerador de números pseudo-aleatórios da biblioteca *NumPy*[Harris et al. 2020]. De modo que, x armazena os índices (randomizados) que compuseram o novo conjunto de dados. Nesse sentido, a complexidade assintótica do método depende de $O(N_{boot}) \times O(n)$, ou simplesmente, $O(n)$.

```
import numpy as np
porosity = [30.3, 21.0, 19.2, 29.1, 21.9, 23.1]
Nboot = 400
def bootstrap(x, Nboot, statfun):
    x = np.array(x)
    resampled_stat = []
    for b in range(Nboot):
        index = np.random.randint(0,len(x),len(x))
        sample = x[index]
        bstatistic = statfun(sample)
        resampled_stat.append(bstatistic)
    return np.array(resampled_stat)
porosity_bootstrap = bootstrap(porosity, Nboot, np.mean)
```

Código 1: Exemplo de *bootstrapping* em Python

No caminho de explicar a origem do nome, os autores do método de *bootstrap* [Efron and Tibshirani 1979] o referenciaram a um conto alemão do século XVIII denominado As Aventuras do Barão de Münchhausen (em tradução livre), de Rudolph Erich Raspe. Em um trecho do livro, o barão escapa de um pântano puxando (a si próprio) pela cinta das próprias botas. É uma alegoria que tenta retratar algo, ou alguém, que consegue evoluir partindo do nada e pelos próprios meios.

Uma outra alegoria que circunda o tema está na na conclusão do artigo denominado *Bagging Predictors*. O autor [Breiman 1996] utiliza o contrassenso de um provérbio dizendo que utilizar o método de *Bagging* em preditores, é uma maneira de fazer ”uma bolsa de seda a partir da orelha de uma porca”e completa, ”especialmente se a orelha da porca estiver trêmula”.

Na prática, o que o método de *Bagging* faz é produzir varias versões do conjunto de amostras para depois criar conjuntos de treino $\mathcal{L}$ e teste $\mathcal{T}$. Para se ter noção, no artigo que apresentou o método, o autor transformou um conjunto de 200 amostras em 2000, de modo que, 200 foram utilizadas como treinamento de uma árvore de decisão e 1800 como teste.

Nesse sentido, um conjunto de árvores com *bagging* (*Bagging Trees*) é um método em que na construção de cada árvore é feita uma seleção aleatória a partir dos exemplos do conjunto de treinamento.

Ampliando, surge a possibilidade de escolha aleatória de n diferentes subconjuntos de preditores, a partir do espaço de preditores original, para treinar n classificadores, com isso, cada elemento do conjunto será treinado com apenas uma porção das preditores, permitindo classificar uma amostra com preditores ausentes ao selecionar os elementos do conjunto treinados somente com os preditores disponíveis no dado a ser classificado, esse método é chamado de subespaço aleatório [Polikar et al. 2010].

2.3. Floresta de árvores aleatórias

Nesse sentido, em uma floresta de árvores aleatórias (*Random Forest*) há uma combinação de preditores de árvores de modo que cada árvore depende dos valores de um vetor aleatório amostrado independentemente e com a mesma distribuição para todas as árvores da floresta [Breiman 2001].

O Scikit-learn [Pedregosa et al. 2011] é uma das mais populares bibliotecas para fazer aprendizado de máquina em Python [Grus 2019]. Seus pacotes contêm diversos módulos, dos quais pode-se citar o `sklearn.tree`, onde estão os módulos necessários para árvores de decisão e, para florestas de árvores, o `sklearn.ensemble`.

A complexidade assintótica do *Random Forest*, sem limitações é $O(M \cdot N \cdot \log N)$, onde M é o número de árvores e N é o número de amostras. É possível reduzir a complexidade alterando parâmetros que limitem o número mínimo de amostras para que o nó seja particionado (`min_samples_split`), o número máximo de nós folhas (`max_leaf_nodes`), a altura das árvores (`max_depth`) e o número mínimo de amostras nas folhas (`min_samples_leaf`).

A utilização é facilitada e basta inserir alguns parâmetros e a biblioteca encarrega-se do restante. Na sequência tem-se o código-fonte que carrega um conjunto de dados, seleciona-se os preditores e alvo, separa a base de teste, executa o algoritmo do Random Florest e por fim realiza a impressão da análise.

```
import pandas as pd
from sklearn.model_selection import train_test_split
from sklearn.metrics import accuracy_score
from sklearn.metrics import recall_score
from sklearn.metrics import confusion_matrix
from sklearn.ensemble import RandomForestClassifier

github = (
    'https://raw.githubusercontent.com/ecostadelle/'+
    'RandomForestTex/main/db/transforma.csv')
df = pd.read_csv(github, sep = ';',decimal=",")
X = df[['ira_nb1', 'ira_mr1', 'ira_nb3', 'rec_s1']]
y = y=df[['resultado_final']]

X_train, X_test, y_train, y_test = train_test_split(X,
                                                    y,
                                                    test_size=0.25,
                                                    random_state=123)
rfc = RandomForestClassifier(bootstrap=True,
                             n_estimators=10,
                             max_depth=None,
                             min_samples_split=2,
                             random_state=0)
rfc.fit(X_train, y_train.values.ravel())
y_pred=rfc.predict(X_test)
```

Código 2: Exemplo de *Random Forest* utilizando o Scikit-learn

No código-fonte anterior o parâmetro *n_estimators* define o número de árvores que

serão geradas e a impressão destas pode ser feito pelo código-fonte que segue:

```
from dtreeviz.trees import dtreeviz
from sklearn import tree
from matplotlib import pyplot as plt
plt.rcParams.update({'figure.figsize': (12.0, 8.0)})
plt.rcParams.update({'font.size': 14})
plt.figure(figsize=(20,20))
_ = tree.plot_tree(rfc.estimators_[0],
                   feature_names=X.columns,
                   filled=True)
```

Código 3: Exemplo de Impressão de *Random Forest* utilizando o Scikit-learn

Ao executar o código-fonte supracitado gera-se uma árvore escolhida no parâmetro `n_estimators`, sendo um exemplo, a árvore a seguir.

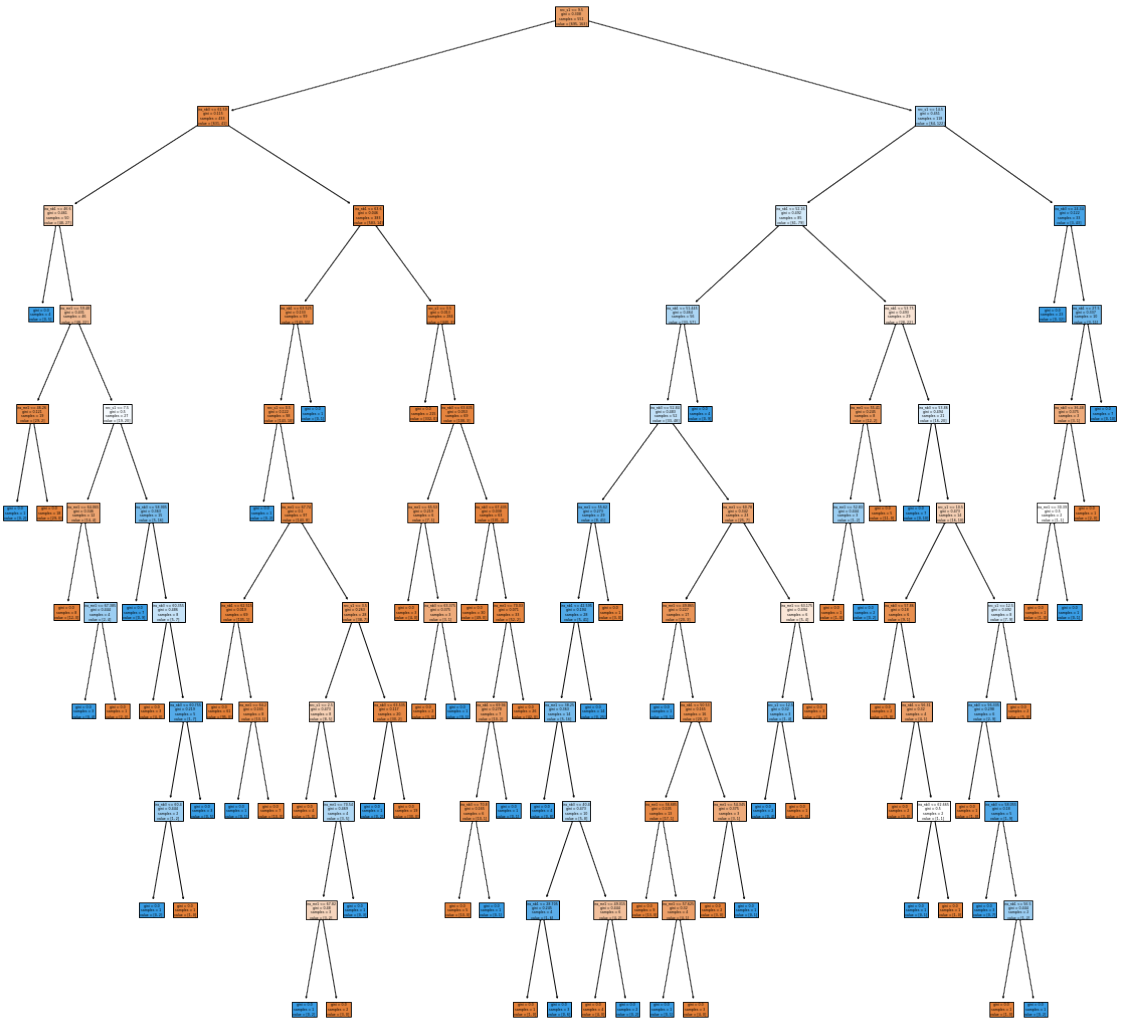

Figura 2. Arvore 0 [0,9] do modelo treinado

Referências

[Breiman 1996] Breiman, L. (1996). Bagging predictors. *Machine Learning*, 24(2):123–140.

[Breiman 2001] Breiman, L. (2001). Random forests. *Machine Learning*, 45(1):5–32.

[Breiman et al. 1984] Breiman, L., Friedman, J. H., Olshen, R. A., and Stone, C. J. (1984). *Classification And Regression Trees*. Routledge.

[Buhmann et al. 2011] Buhmann, M. D., Melville, P., Sindhwani, V., Quadrianto, N., Buntine, W. L., Torgo, L., Zhang, X., Stone, P., Struyf, J., Blockeel, H., Driessens, K., Miikkulainen, R., Wiewiora, E., Peters, J., Tedrake, R., Roy, N., Morimoto, J., Flach, P. A., and Fürnkranz, J. (2011). Random forests. In *Encyclopedia of Machine Learning*, pages 828–828. Springer US.

[Efron and Tibshirani 1979] Efron, B. and Tibshirani, R. J. (1979). *An Introduction to the Bootstrap*. Springer US.

[Grus 2019] Grus, J. (2019). *Data Science do Zero: Primeiras Regras com o Python*. Alta Books.

[Harris et al. 2020] Harris, C. R., Millman, K. J., van der Walt, S. J., Gommers, R., Virtanen, P., Cournapeau, D., Wieser, E., Taylor, J., Berg, S., Smith, N. J., Kern, R., Picus, M., Hoyer, S., van Kerkwijk, M. H., Brett, M., Haldane, A., del Río, J. F., Wiebe, M., Peterson, P., Gérard-Marchant, P., Sheppard, K., Reddy, T., Weckesser, W., Abbasi, H., Gohlke, C., and Oliphant, T. E. (2020). Array programming with NumPy. *Nature*, 585(7825):357–362.

[Loh 2014] Loh, W.-Y. (2014). Fifty years of classification and regression trees. *International Statistical Review*, 82(3):329–348.

[Messenger and Mandell 1972] Messenger, R. and Mandell, L. (1972). A modal search technique for predictibe nominal scale multivariate analys. *Journal of the American Statistical Association*, 67(340):768.

[Morgan and Sonquist 1963] Morgan, J. N. and Sonquist, J. A. (1963). Problems in the analysis of survey data, and a proposal. *Journal of the American Statistical Association*, 58(302):415–434.

[Pedregosa et al. 2011] Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., and Duchesnay, E. (2011). Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12:2825–2830.

[Polikar et al. 2010] Polikar, R., DePasquale, J., Mohammed, H. S., Brown, G., and Kuncheva, L. I. (2010). Learn++. mf: A random subspace approach for the missing feature problem. *Pattern Recognition*, 43(11):3817–3832.

[Quinlan 1986] Quinlan, J. R. (1986). Induction of decision trees. *Machine learning*, 1(1):81–106.