Aprendizado Ensemble - Florestas Aleatórias
Trabalho 1 – INF01017 Aprendizado de Máquina – 2019/1

Bruno S. M. de Lima, Lucas N. Alegre e Pedro S. Perrone

Maio de 2019

1 Introdução

Este trabalho tem como objetivo a documentação da implementação e da avaliação de desempenho do algoritmo de Florestas Aleatórias (Ramdom Forests) para tarefas de aprendizado supervisionado seguindo o paradigma de ensemble, i.e., o uso de múltiplos modelos. O treinamento das árvores se baseia no método do ganho de informação.

2 Implementação

Nossa implementação foi feita com a linguagem Python3. Usamos a bilioteca **pandas** para armazenamento e manipulações básicas dos datasets, a biblioteca **numpy** para cálculos estatísticos básicos como média e desvio padrão, a biblioteca **graphviz** para gerar visualizações gráficas das árvores treinadas, a biblioteca **tqdm** para visualizar o progresso e o tempo de treinamento e a biblioteca **seaborn** para gerar os gráficos aqui utilizados. Todas as funcionalidades relativas ao algoritmo de Árvores Aleatórias foram implementadas pelo grupo.

Para validar a corretude da implementação, geramos a imagem da árvore de decisão gerada através do treinamento dos dados no benchmark disponibilizado (Figura 1).

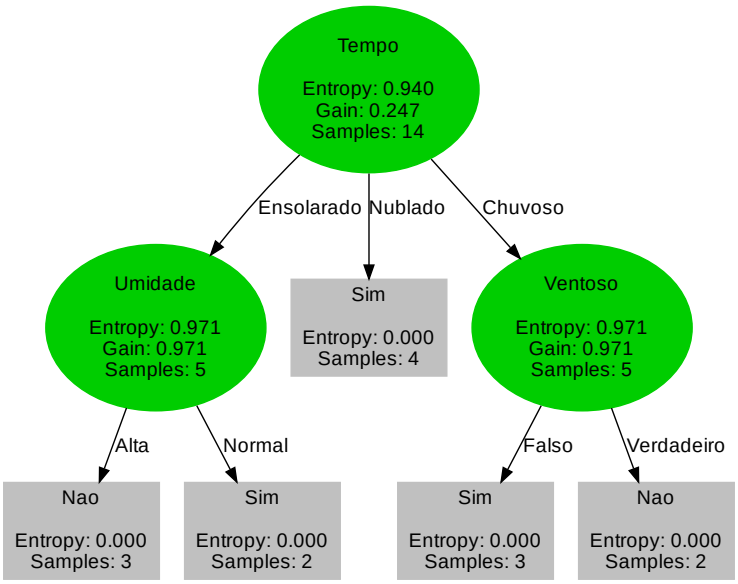

Figura 1: Árvore de Decisão gerada através do dataset de Benchmark

2.1 Arquitetura da Implementação

A implementação feita é composta de três classes (*RandomTreeNode*, *RandomTree* e *RandomForest*) e um módulo de funções auxiliares (*util.py*).

Essa estrutura pode ser vista em detalhes no diagrama de classes da Figura 2.

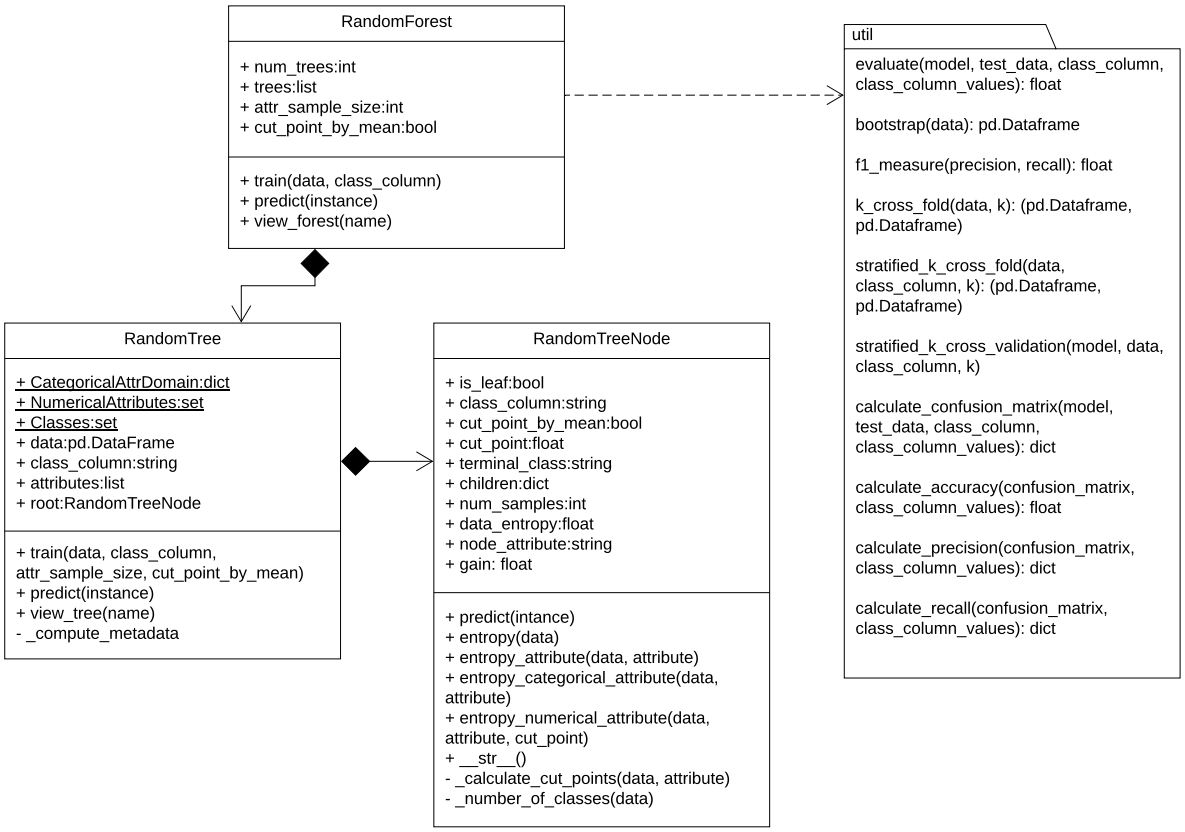

Figura 2: Diagrama de Classes

2.2 Estruturas de Dados

O conjunto de instâncias de teste e treinamento foram armazenadas numa tabela representada por um *DataFrame* da biblioteca **pandas**, que fornece uma implementação eficiente para manipulação de grandes quantidades de dados.

Os metadados foram representados pelas seguintes variáveis estáticas da classe *RandomTree*:

- *CategoricalAttrDomain* - dicionário que para cada atributo categórico armazena a lista de possíveis valores.
- *NumericalAttributes* - conjunto que guarda o nome de todos os atributos numéricos do dataset.
- *Classes* - conjunto de todas as classes que podem ser preditas no dataset.

O uso de dicionários e conjuntos possibilitam consultas em complexidade assintótica constante, sendo muito mais eficaz que o uso de uma lista nesse caso.

Para acelerar a navegação (e consequentemente o processo de predição), os nodos filhos de um certo nodo são armazenados num dicionário. Cada chave é um possível valor do atributo do nodo e os valores são os respectivos nodos filhos de cada divisão do atributo.

2.3 Funcionalidades Principais

Demonstramos nessa seção duas das principais funcionalidades implementadas: a predição de uma nova instância e a validação cruzada estratificada. É possível visualizar as outras funcionalidades relacionadas à construção das árvores no código completo da implementação.

2.3.1 Predição

Abaixo mostramos como a predição de uma nova instância acontece. A *RandomForest* realiza uma votação majoritária, onde cada *RandomTree* chama o método *predict* no nodo raíz, que propaga a decisão até encontrarmos um nodo folha.

Listing 1: Predição de uma nova instância

```
# Random Forest
def predict(self, instance):
    # Majority Voting
    trees_predictions = [tree.predict(instance) for tree in self.trees]
    forest_prediction = max(set(trees_predictions), key=trees_predictions.count)
    return forest_prediction

# Random Tree
def predict(self, instance):
    return self.root.predict(instance)

#Random Tree Node
def predict(self, instance):
    if self.is_leaf:
        return self.terminal_class
    if self.node_attribute in RandomTree.NumericalAttributes:
        return self.children[instance[self.node_attribute] <= self.cut_point].predict(instance)

    return self.children[instance[self.node_attribute]].predict(instance)
```

2.3.2 Validação Cruzada Estratificada

Abaixo podemos ver o código utilizado para realizar a validação cruzada estratificada. Inicialmente, é gerado um novo dataframe para cada classe possível, contendo todas as suas intâncias. A partir disso, cada dataframe gerado é subdividido em k subconjuntos de mesmo tamanho, que posteriormente são combinados com subconjuntos de outras classes para gerar um fold estratificado.

Listing 2: Validação cruzada estratificada

```
def stratified_k_cross_fold(data, class_column, k=10):
    data = data.sample(frac=1).reset_index(drop=True) # Shuffle DataFrame rows

    folds_per_class = []
    for _, g in data.groupby(class_column):
        folds_per_class.append(k_cross_fold(g, k=k))  # create fold for each class

    for _ in range(k):
        train = pd.DataFrame()
        test = pd.DataFrame()
        for c in range(len(folds_per_class)):         # append the next fold for each class
            train_c, test_c = next(folds_per_class[c])
            train = train.append(train_c)
            test = test.append(test_c)
        yield train, test
```

3 Técnicas escolhidas

O enunciado do trabalho abriu espaço para a escolha da técnica empregada em duas partes da implementação: na amostragem de atributos e no cálculo do valor de corte em atributos numéricos.

3.1 Amostragem de atributos

Para realizar a amostragem de atributos, decidimos utiliazar o método mais recorrente na literatura: seleção aleatória da raiz quadrada da quantidade total de atributos disponíveis. A realização da amostragem foi parametrizada para fins de teste de desempenho e tempo de treinamento.

3.2 Valor de corte de atributos numéricos

Fizemos duas implementações para a escolha do valor de corte e parametrizamos a sua escolha: ele pode ser simplesmente a média aritmética entre os valores das instâncias de treinamento ou o método discutido em aula e mencionado no enunciado do trabalho, o qual trata como possíveis pontos de corte a média aritmética entre os valores de duas instâncias consecutivas que não pertençam à mesma classe. O ponto de corte escolhido é o que gera o maior ganho de informação.

4 Parâmetros do algoritmo

Para a geração das árvores os seguintes valores foram parametrizados:

- **num_trees:** representa a quantidade de árvores geradas na floresta;
- **not sample:** se verdadeiro, o algoritmo deixa de utilizar amostragem de atributos na construção das árvores. Caso o contrário, ele faz a amostragem de $\sqrt{m}$ atributos, sendo m o número total atributos disponíveis naquele nó;
- **cut_by_mean:** se verdadeiro, o algoritmo utiliza a média aritmética das instâncias para o cálculo do ponto de corte de um atributo numérico. Caso o contrário, gera uma lista de possíveis pontos de corte ao calcular a média aritmética de duas instâncias consecutivas que não pertençam à mesma classe e escolhe o com maior ganho de informação;
- **k_folds:** quantidade de folds que serão gerados na validação cruzada estratificada. Fixamos o valor 10 para esse parâmetro em todos os experimentos como foi recomendado pelos professores.

5 Parâmetro *cut_by_mean*

Para analisar o parâmetro *cut_by_mean* realizamos experimentos sobre uma única árvore. Optamos por não utilizar amostragem, ou seja, atribuir ao parâmetro *not_sample* o valor *true*, uma vez que amostragem visa trazer diversidade ao *ensemble* e não melhorar o desempenho de um único modelo. Os conjuntos utilizados foram *wine.csv*, *wdbc.csv* e *ionosphoere.csv* e os resultados da validação cruzada estratificada com 10 *folds* se encontram na Figura 3.

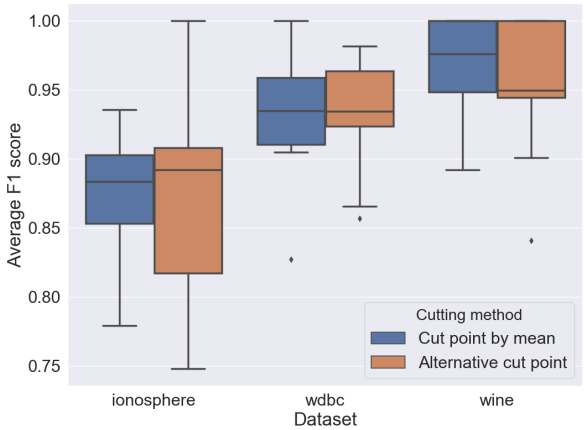

Figura 3: Desempenho de diferentes métodos de escolha de ponto de corte em uma árvore.

A utilização deste novo método de definição do ponto de corte não aumentou significativamente o desempenho das árvores isoladamente, porém aumentou em cerca de 7 vezes o tempo de construção de uma árvore de decisão. Portanto, analisando somente estes dados, não é justificável o uso do método alternativo para geração das florestas. Partindo disso, decidimos observar diretamente as árvores geradas, buscando algo que possa justificar a escolha de algum dos métodos. O que fizemos foi gerar uma pequena floresta com três árvores e ver as representações gráficas das estrututras geradas. As Figuras 4 e 5 foram os resultados.

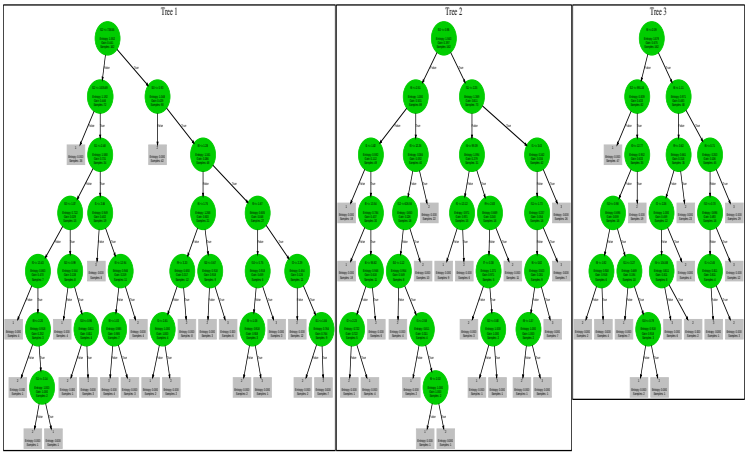

Figura 4: Floresta com ponto de corte pela média gerada a partir do dataset *wine*.

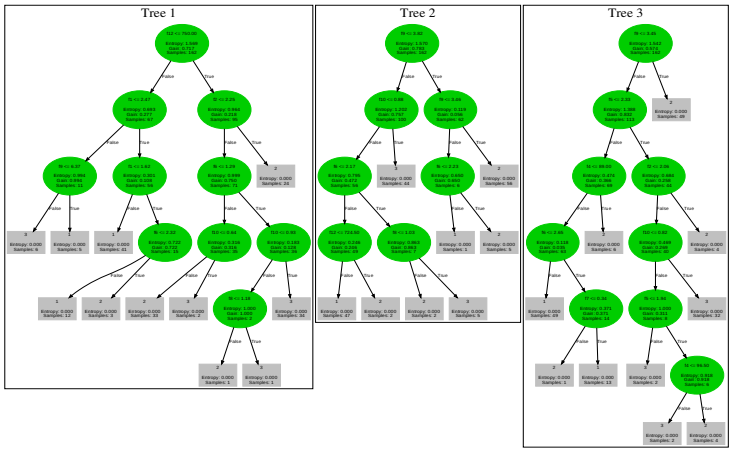

Figura 5: Floresta com ponto de corte alternativo (visto em aula) gerada a partir do dataset *wine*.

Observamos que o método alternativo gerou árvores mais simples consistentemente, reduzindo as chances de ocorrer *overfitting* ao treina-las. Sendo assim, o escolhemos para treinar as florestas em outros experimentos.

6 Variação do Número de Árvores na Floresta Aleatória

Os experimentos sobre florestas aleatórias trazem um novo parâmetro: *ntree*, a quantidade de árvores na floresta. Para os outros parâmetros, mantivemos os valores necessários para realizar amostragem e o método de definição de ponto de corte escolhido na Seção 5.

Além dos datasets fornecidos, também avaliamos dois datasets alternativos: *pima* (`https://github.com/EpistasisLab/penn-ml-benchmarks/tree/master/datasets/classification/pima`), que possui

8 atributos númericos, 768 exemplos e 2 classes e *car* (`https://archive.ics.uci.edu/ml/datasets/car+evaluation`), que possui 6 atributos categóricos, 1728 exemplos e 4 classes.

As Figuras 6, 7, 8, 9 e 10 apresentam o impacto do número de árvores na performance da Floresta Aleatória. É apresentado o boxplot para cada valor de número de árvores, onde cada ponto é o F1 score da avaliação em um dos k folds de teste (portanto 10 pontos para cada boxplot).

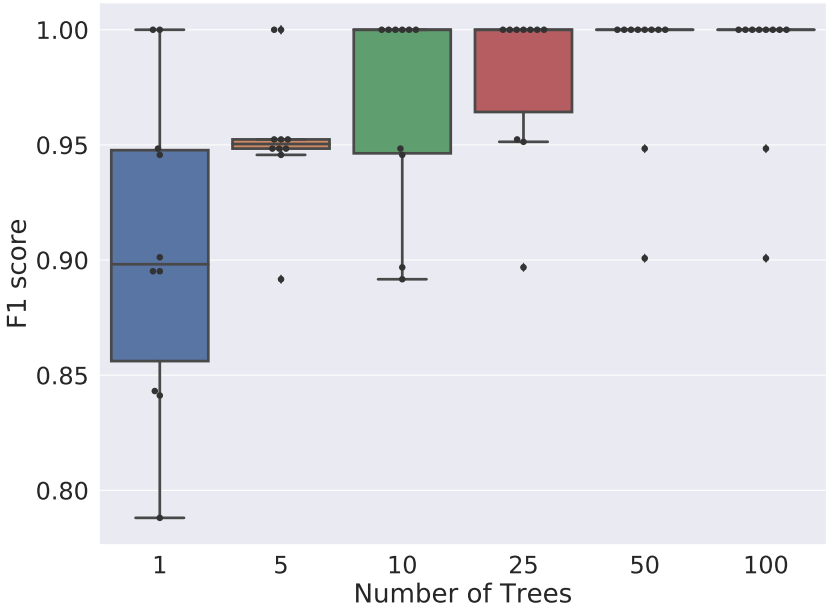

Figura 6: Desempenho de florestas aleatórias no conjunto de treinamento *wine*.

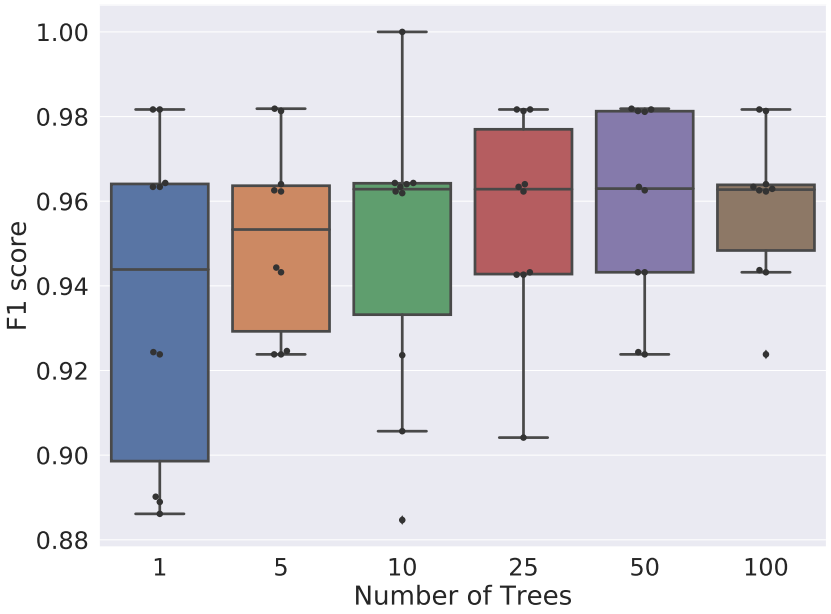

Figura 7: Desempenho de florestas aleatórias no conjunto de treinamento *wdbc*.

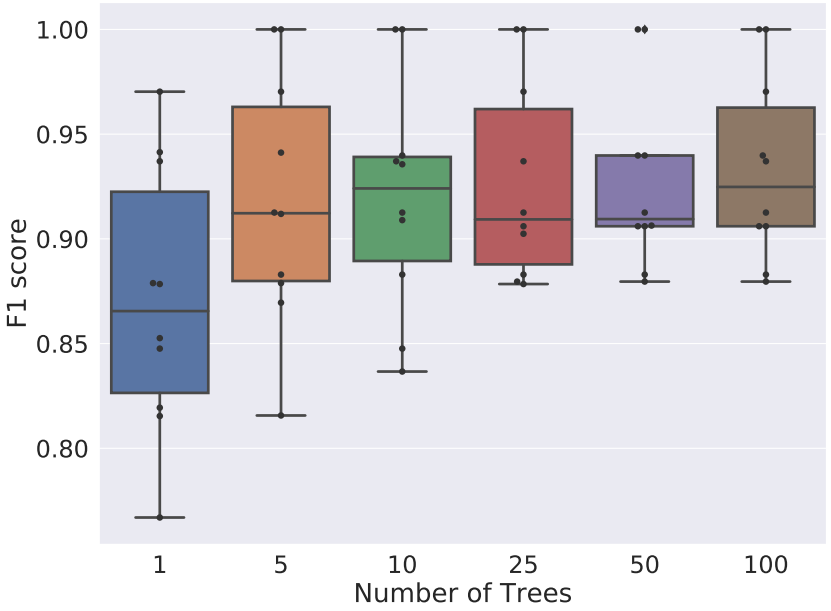

Figura 8: Desempenho de florestas aleatórias no conjunto de treinamento *ionosphere*.

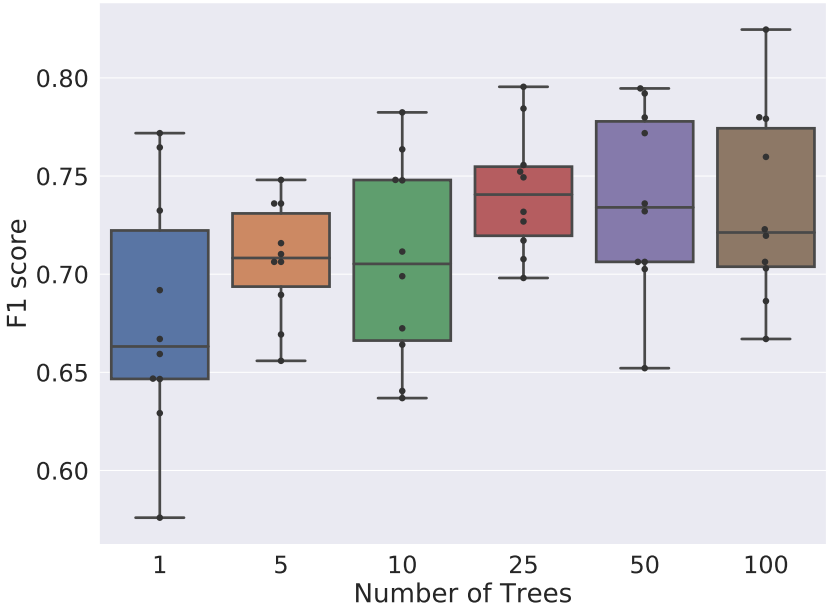

Figura 9: Desempenho de florestas aleatórias no conjunto de treinamento *pima*.

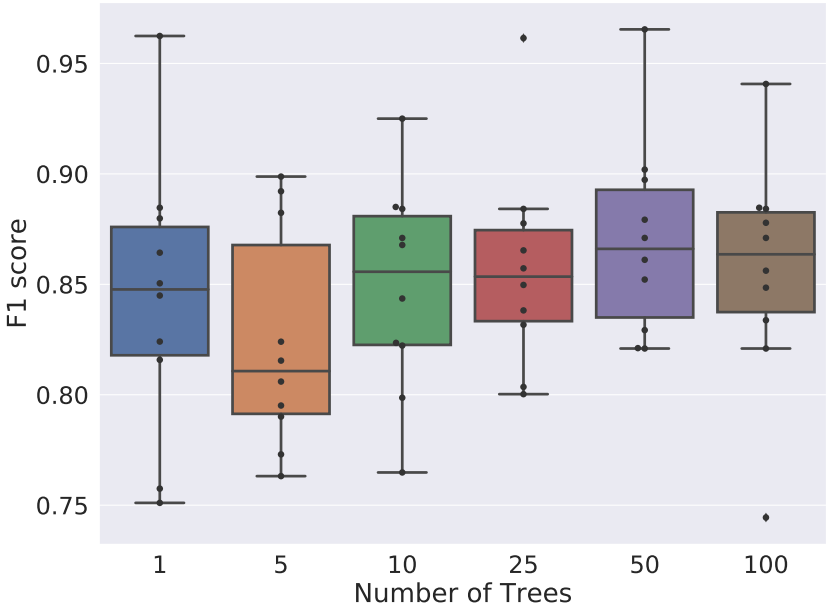

Figura 10: Desempenho de florestas aleatórias no conjunto de treinamento *car*.

Como esperado, os resultados obtidos mostram a tendência de uma melhor performance quanto maior o número de árvores. Esse resultado é mais evidente nos datasets *wine* (Figura 6), *pima* (Figura 9) e *ionosphere* (Figura 8). Nos datasets restantes a performance parece não sofrer melhoria significativa com mais de 10 árvores.

Vale notar que datasets com uma quantidade pequena de atributos podem não se beneficiar tanto do aumento do número de árvores em comparação com o aumento do tempo de treinamento. Tal fenômeno pode ser observado na Figura 10, cujo dataset analisado tem somente seis atributos. Já para *datasets* com uma grande quantidade de atributos, o aumento no número de árvores utilizadas reduz a variância mais significativamente. Isso pode ser observado principalmente na Figura 7, com informações sobre *wdbc*.

Além disso, pode-se observar a capacidade que os métodos *ensemble* têm para reduzir a variância se comparado ao uso de um único modelo. Analisando os resultados obtidos em todos os datasets, percebe-se que a versão que só treina uma única Árvore de Decisão apresenta sempre a maior variância. Entretanto, os resultados apresentados possuem alta variância mesmo com alta quantidade de árvores. Acreditamos que isso se deve, além do fato de Árvores de Decisão serem conhecidas na literatura por sua alta variância, ao fato de que temos apenas 10 medidas para cada valor do número de árvores. Se fosse feito validação cruzada repetida os resultados teriam maior significância estatística. Porém, não obtivemos recursos computacionais o suficiente, visto que o treinamento de 100 árvores pode levar mais de 5 horas em alguns dos datesets.

7 Conclusão

Florestas Aleatórias se mostraram uma ferramenta de predição bastante poderosa. Contudo, elas podem exigir bastante recursos computacionais, principalmente utilizando o método de definição de ponto de corte escolhido para atributos numéricos. Pode-se observar melhora no F1 score medido na validação cruzada estratificada para um maior número de árvores na Floresta Aleatória.

No geral, a implementação do grupo obteve desempenho satisfatório na maioria dos datasets e exemplificou as vantagens de métodos *ensemble* em relação aos métodos de modelo único.