Fundamentos de Sistemas Inteligentes
Projeto 2 - Classificação de espécies de plantas com base em folhas

Hugo Nascimento Fonseca
Universidade de Brasília - UnB
Engenharia de Computação
16/0008166
hugonfonseca@hotmail.com

José Luiz Gomes Nogueira
Universidade de Brasília - UnB
Engenharia de Computação
16/0032458
joseluizgnogueira@live.com

***Resumo*—Este artigo tem como objetivo apresentar o relatório do Projeto 2 da disciplina de Fundamentos de Sistemas Inteligentes, semestre 2019/1 - Prof. Díbio Borges, na Universidade de Brasília.**

I. Introdução

O projeto consiste em aplicar o algoritmo Florestas Randômicas para classificação supervisionada de espécies de plantas. O conjunto de dados conta com 40 espécies de plantas e 14 parâmetros de suas folhas, dos quais 7 deles descrevem seu formato, e os outros 7 descrevendo sua textura, tendo como objetivo avaliar o algoritmo realizando variações em seus hiperparâmetros, tais como poda, mudança de função de ganho, entre outros.

II. Fundamentação teórica

A. Descrição do algoritmo

Uma floresta randômica é um conjunto de árvores de decisão que votam de forma unitária para eleger a classe majoritária, a classe que representa um dado input, cada árvore cresce utilizando como base um conjunto de vetores gerados aleatoriamente.

De forma mais detalhada, temos que, cada árvore é criada a partir de um subconjunto de dados aleatórios obtidos a partir do conjunto de dados fornecidos na entrada, formando assim o conjunto de teste daquela árvore, esse conjunto é um vetor de atributos de tamanho arbitrário e de tamanhos iguais. Dentro do conjunto de treino de uma árvore é realizado uma seleção aleatória dos atributos que serão utilizado para crescer a árvore nó à nó. Por fim com todos as árvores devidamente criadas, dado uma entrada, a classificação deste é definido pelo voto majoritário entre as árvores.

Seguindo a dissertação de Leo Breiman sobre Random Forests, podemos definir uma floresta randômica como um classificador supervisionado estruturado em árvores de decisão regidos pelo crescimento dado pelas entradas $h(x, \theta_k), k = 1, ...,$ onde θ_k são vetores aleatórios independentes distribuídos identicamente, ou seja, possuem baixa colinearidade e possuem o mesmo tamanho, e cada árvore contribui com seu voto unitário para classe mais popular x. [1]

O algoritmo possui execução rápida, considerando casos médios ($O(MKNlogN)$ M randomizações de árvores, K atributos utilizados por nó e N número de exemplos para teste) [2] e alta acurácia para os conjuntos de treino, entretanto essa alta acurácia para os conjuntos de treino sacrifica a acurácia do teste devido ao overfitting, pois o algoritmo facilita a criação de uma árvore muito engessada para o conjunto de dados, esta característica é ainda mais presente utilizando uma única árvore de decisão, entretanto a floresta randômica ainda pode apresentar este comportamento.

Para evitar o sobreajuste, algumas abordagens podem ser utilizadas, dentre elas a poda (que consiste em remover algumas subárvores que possuem baixo ganho ou ganho negativo), ou alteração nos parâmetros de crescimento, tais como:

1) Número mínimo de amostras para uma divisão de nós:
   Ex: Se um nó possui 5 amostras, para serem divididos, e o mínimo for 6, o nó é descartado.
   O uso de um numero mínimo muito grande, entretanto pode causar subajuste.
2) Número mínimo de amostras para uma folha:
   Ex: Se uma folha possui 2 amostras que levam a ela, e o número mínimo é 3, essa folha é descartada.
   Novamente, o uso de um número muito grande, pode levar a subajuste.
3) Profundidade máxima:
   Define o tamanho máximo que cada árvore pode ter, caso seja utilizado um valor muito baixo pode causar subajuste também.
4) Ganho mínimo para uma divisão de nós:
   Ex: Se a escolha de dividir um nó em 2, reduzir a impureza em x, e for necessário x+1 para divisão, o nó é descartado.
5) Número usado de atributos usados para escolha da melhor divisão em um nó:
   Os atributos que serão utilizados, são escolhidos aleatoriamente nó à nó de maneira independente. O uso de um número muito pequeno pode causar subajuste e um

número muito grande, pode causar sobreajuste. Segundo indicado em arquivos de estudo o melhor valor para este hiperparâmetro é de $\sqrt{N}$ onde N é a quantidade de atributos.

III. Projeto

A. *Software*

Para executar essa tarefa, escrevemos um projeto em Python utilizando os seguintes componentes, com suas respectivas versões, abaixo:

- Python 3.6.7
- NumPy 1.16.2
- Matplotlib 3.0.3
- Pandas 0.24.2
- OpenCV 4.0.0.21
- sklearn 0.20.3

B. *Execução*

Com o objetivo de analisar o algoritmo foi utilizado a implementação da biblioteca sklearn em Python, que possui em seu algoritmo de treinamento de florestas randômicas, a possibilidade de variar os diversos hiperparâmetros para controle de crescimento da floresta randômica. Juntamente a execução do treino usando a floresta randômica, foi utilizado também o módulo model_selection [4] da biblioteca sklearn, para geração das 10 subseções dos dados para utilização da estrutura da validação cruzada.

Dito isso, foi construído um algoritmo capaz de executar em o treino utilizando floresta randômica e validação cruzada, podendo facilmente variar quais quer parâmetros escolhidos, e salvando as métricas de cada teste em um arquivo json.

Uma vez construído este algoritmo, foi realizado o teste do funcionamento do algoritmo variando-se diversos hiperparâmetros de forma isolada, criando assim um arquivo json para cada conjunto de valores de um hiperparâmetro.

Por fim, foi realizado a construção de um algoritmo capaz de ler este json e construir um gráfico exibindo as métricas em função das variações dos valores do hiperparâmetro, o gráfico pode exibe o valor da métrica mínima, máxima ou media obtido em cada validação cruzada, conforme parâmetro passado para função.

Com isto esperamos conseguir observar o predito em teoria quanto à variação de hiperparâmetros, podendo assim escolher os melhores valores de hiperparâmetros para o conjunto de dados utilizado.

Por fim, possuindo em mãos os melhores hiperparâmetros, apresentar a melhor acurácia e precisão, bem como a matriz de confusão.

IV. Avaliação/Resultados

A. *Máquina utilizada nos testes*

- Processador: i3 - 3217u - 1.8GHz
- Memória: 8GB - DDR3 - 1066MHz
- Sistema Operacional: Ubuntu 18.04 LTS

Ao realizar o procedimento descrito na seção anterior, obtivemos 7 gráficos, um para cada hiperparâmetro testado. Foi realizado também essas mesmas medidas utilizando como função de agregação dos testes de validação cruzada média, valor mínimo e máximo, entretanto para fins de simplificação será apresentado aqui somente os gráficos utilizando agregação via média e comentado a comparação com os demais.

Para fins de comparação ao final também será apresentado tabelas com os melhores valores obtidos em todas as curvas de todos os gráficos, utilizando tanto média, como valor mínimo e máximo para a agregação dos valores obtidos nos testes da validação cruzada.

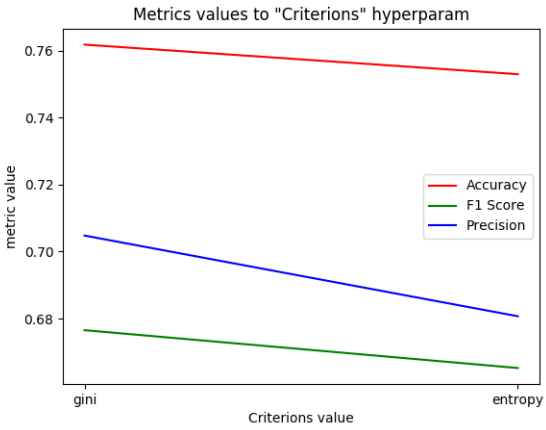

Figura 1: Comportamento das métricas em função dos critérios de pureza e ganho usada para divisão dos nós

Como podemos observar na figura 1, podemos constatar que o uso da índice de Gini, se sobressai por uma pequena diferença de valor, entretanto realizando diversos treinos e comparando os gráficos, observamos que essa diferença é menor que a flutuação encontrada, logo para cada treino completo realizado, uma se sobressai em relação à outra, logo para o conjunto de dados utilizado não é possível inferir qual seria superior, uma vez que depende diretamente dos subconjuntos criados para o treino de cada árvore e de cada subconjunto utilizado na validação cruzada.

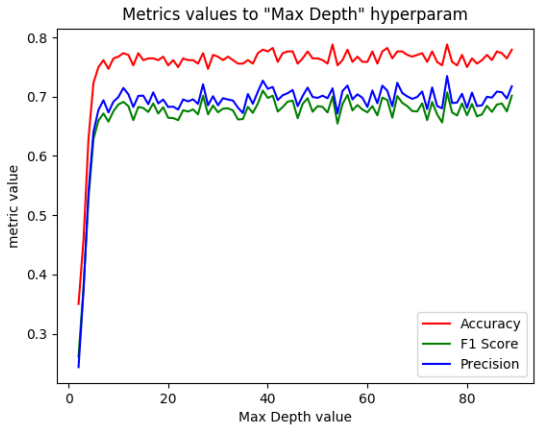

Figura 2: Comportamento das métricas em função da profundidade máxima

A figura 2 apresenta a flutuação das métricas em função da profundidade, novamente a variação se deve a randomização na criação dos subconjuntos, porém é visível que um valor muito pequeno causa subajuste, assim como previsto em teoria, e um comportamento assintótico, entretanto para este conjunto utilizado este valor de subajuste e o valor assintótico são muito próximos, acreditamos que isso se deva a baixa necessidade da árvore necessitar de múltiplos níveis o que torna o algoritmo mais leve, conforme indicado na complexidade do algoritmo apresentado na seção teórica, logo o melhor valor a se escolher é o valor imediato à assintota algo em torno de 13 níveis. Analisando este mesmo tipo de gráfico para os demais agregadores observamos um mesmo comportamento entretanto com maior flutuação ponto à ponto.

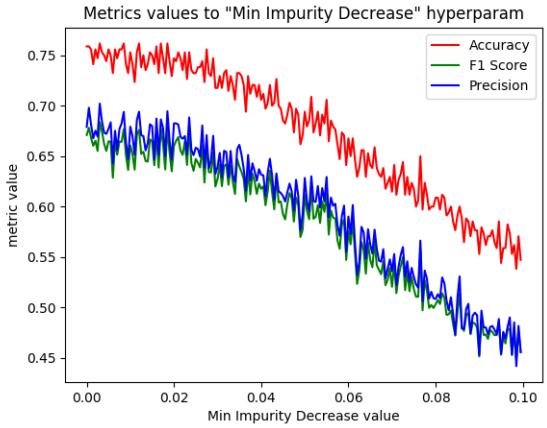

Figura 3: Comportamento das métricas em função da quantidade mínima de redução de impureza para divisão dos nós

Novamente na figura 3 podemos observar grande flutuação, contudo o comportamento decrescente é visível, bem como esperado por teoria, definir um mínimo de redução na impureza para a divisão dos nós torna a possibilidade de divisão mais escassa causando subajuste, entretanto pode ser interessante para garantir uma árvore menor e mais rápida, com menor sobreajuste, entretanto optamos por utilizar este hiperparâmetro com o valor 0.0, deixando assim para divisão do nó a melhor subárvore que maximizar a redução da impureza, sem restrição mínima.

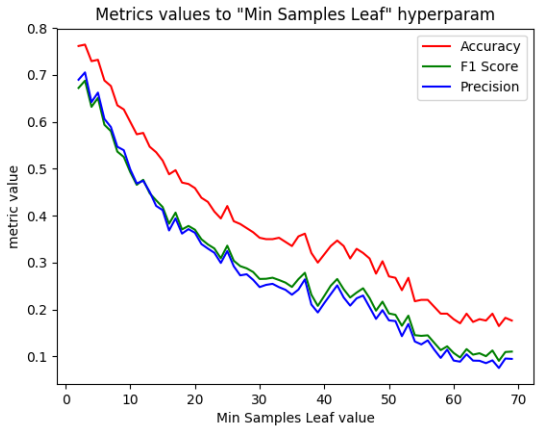

Figura 4: Comportamento das métricas em função da quantidade de amostras para definir uma folha

A figura 4 mostra que a classificação é bastante sensível a esse parâmetro, isso se deve ao fato que estamos analisando uma base de dados pequena, logo, qualquer restrição no quantidade e amostrar para definição de uma folha é crucial. Por exemplo, sabemos que a árvore não é muito profunda, logo possui poucas ramificações, nosso conjunto total de amostras é de 340 amostras, logo utilizando validação cruzada, temos para treino 306 amostras, utilizando todas as amostras nas árvores e dividindo igualmente entre todas as 40 possíveis folhas temos que cada folha terá aproximadamente 7 amostras, obviamente no bagging não utilizamos toda as amostras, e por serem escolhidas de forma aleatória teremos folhas com muito menos do que 7 amostras, logo se o nosso melhor caso possui 7 amostras por folha, concluímos que limitar a criação de folhas por quantidade de amostras pode não ser uma boa ideia, bem como é visível no gráfico, talvez se utilizássemos um conjunto muito maior, seria mais interessante a utilização de um limiar, entretanto para o conjunto de dados usados qualquer valor maior que 3 causa em perca de precisão das predições.

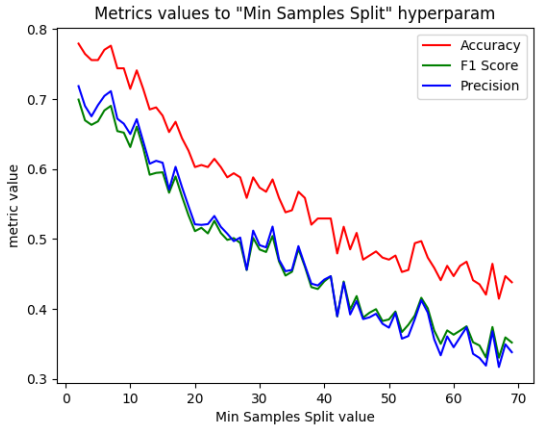

Figura 5: Comportamento das métricas em função da quantidade de amostras para divisão dos nós

Bem como apresentado nas figuras 4 e 5 temos o mesmo comportamento decrescente, afinal ambas os hiperparâmetros

de restrição de crescimento são bastante parecidos, porém, podemos observar nas formas das curvas geradas que, utilizar uma quantidade mínima de amostras para uma divisão do nó suaviza mais a curva, e consequentemente poupa tempo de processamento, entretanto o valor é mais suscetível à subajuste, por isso ocorre maior flutuação no gráfico.

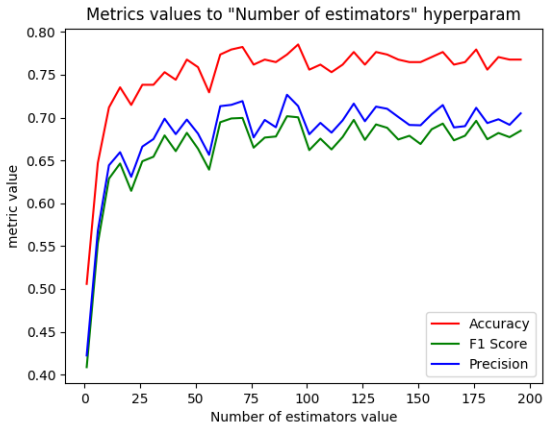

Figura 6: Comportamento das métricas em função da quantidade de árvores

Na figura 6 observamos uma das mais importantes curvas aqui geradas, ela diz respeito a quantidade de árvores utilizadas para predição, seguindo a teoria quanto maior a quantidade de árvores mais votos teremos e menor influência de uma única árvore, logo, melhor a predição, entretanto uma floresta randômica com infintas árvores é no mínimo inviável, e como podemos observar no gráfico, desnecessário, uma vez que este hiperparâmetro possui influência assintótica nas predições, logo basta utilizar um número grande o suficiente para estarmos próximo a linha assintótica. Observando a curva chegamos a conclusão de que este número deve ser algo entre 75 e 100, escolheremos como melhor ajuste o valor de 100 para possuirmos uma margem de erro, assim deixando o algoritmo ainda mais rápido, se comparada com o uso de muito mais árvores, como indicado na ordem de complexidade apresentada na seção teórica.

Uma vez que conseguimos entender um pouco melhor a influência de cada hiperparâmetro nas predições, criamos as tabelas abaixo, ilustrando os melhores valores, de mínimo, máximo e de média para cada hiperparâmetro, apresentando também na tabela o respectivo valor do parâmetro para aquela métrica. E assim podermos decidir qual o melhor conjunto de hiperparâmetros e valores para obter o melhor resultado possível para o conjunto de dados obtido.

O objetivo de apresentar todos estes valores utilizando agregações médias, máxima e mínimas é poder escolher um conjunto de hiperparâmetros que seja bom tanto para as piores distribuições, para as melhores distribuições quanto para todas as distribuições em geral.

B. Análise

Após análise dos gráficos e dos melhores valores destes gráficos, disponíveis na tabela, chegamos a conclusão que os melhores valores a se utilizar para este conjunto de dados são:

- Criterion: entropy | gini
- Max Depth: 13
- Min Impurity Decrease: 0
- Min Samples Leaf: 1 | 2 | 3
- Number of Estimators: 100

Após os testes destes valores obtivemos na prática a seguinte matriz de confusão e sua respectiva tabela de métricas:

Hyperparameter	Accuracy	Param Value	F1 Score	Param Value	Precision	Param Value
Number of estimators	0.785	96	0.702	91	0.727	91
Min Samples Split	0.779	2	0.699	2	0.719	2
Criterion	0.762	gini	0.677	gini	0.705	gini
Max Depth	0.788	53	0.71	39	0.735	76
Min Samples Leaf	0.765	3	0.688	3	0.705	3
Min Impurity Decrease	0.762	0.003	0.684	0.003	0.702	0.003

Tabela I: Tabela com as melhores métricas para cada hiperparâmetro isolado, utilizando como função agregadora a média (média entre os valores obtido treino à treino na validação cruzada)

Hyperparameter	Accuracy	Param value	F1 Score	Param value	Precision	Param value
Number of estimators	0.912	166	0.851	66	0.864	166
Min Samples Split	0.912	2	0.848	2	0.864	2
Criterion	0.912	entropy	0.848	entropy	0.864	entropy
Max Depth	0.912	9	0.865	27	0.886	10
Min Samples Leaf	0.882	2	0.833	3	0.847	3
Min Impurity Decrease	0.912	0.017	0.864	0.0185	0.886	0.0185

Tabela II: Tabela com as melhores métricas para cada hiperparâmetro isolado, utilizando como função agregadora o valor máximo (máximo entre os valores obtido treino à treino na validação cruzada)

Hyperparameter	Acurracy	Param Value	F1 Score	Param Value	Precision	Param Value
Number of estimators	0.676	36	0.55	46	0.58	156
Min Samples Split	0.676	3	0.564	6	0.583	6
Criterion	0.676	gini	0.517	gini	0.555	gini
Max Depth	0.676	39	0.579	39	0.609	76
Min Samples Leaf	0.647	3	0.517	3	0.577	3
Min Impurity Decrease	0.676	0.0145	0.533	0.01	0.571	0.0005

Tabela III: Tabela com as melhores métricas para cada hiperparâmetro isolado, utilizando como função agregadora o valor mínimo (mínimo entre os valores obtido treino à treino na validação cruzada)

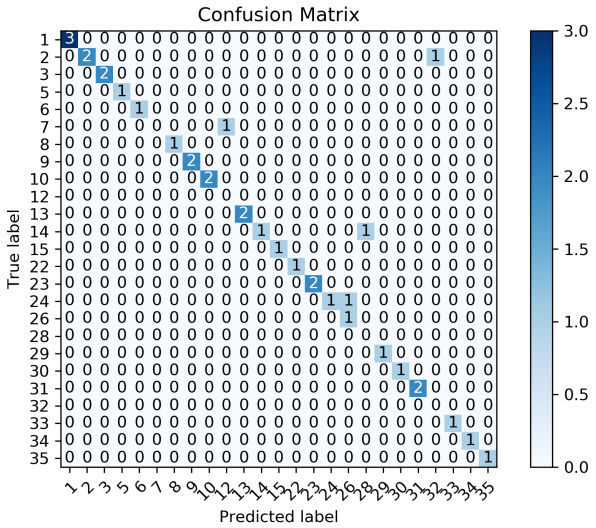

Figura 7: Matriz de confusão para os seguintes hiperparâmetros: criterion = entropy, max_depth = 13, min_impurity_decrease = 0, min_sample_leaf = 2, number_of_estimators = 100

Com essa configuração dos parâmetros, o algoritmo apresentou resultados consistentes e próximos aos valores médios. Considerando a melhor distribuição da validação cruzada apresenta resultados próximos as valores máximos, mesmo com os problemas de overfitting causados pela baixa quantidade de dados.

Métrica	**Valor**
Accuracy	0.883
F1 Score	0.792
Precision	0.821

Tabela IV: Tabela com as métricas para os seguintes hiperparâmetros: criterion = entropy, max_depth = 13, min_impurity_decrease = 0, min_sample_leaf = 2, number_of_estimators = 100

V. Conclusão

Por fim, concluímos que o conjunto de dados é bastante pequeno e impossibilita grandes variações nos valores dos hiperparâmetros, possuindo inclusive muitos casos onde o conjunto de treino não possui nenhum tipo de amostra para uma dada classe. Entretanto mesmo assim, o algoritmo se apresentou eficiente, sendo também bastante rápido tanto no treino quanto nas predições, porém com o grande problema de facilitar o overfitting, tal como podemos observar analisando a diferença entre o mínimo e o máximo na agregação das validações cruzadas, sendo muito sensível ao subconjunto utilizado para treino.

Referências

[1] Gilles L.: Understanding random forests, disponível em: https://arxiv.org/pdf/1407.7502.pdf, acessado em: 8 de maio de 2019.

[2] Breiman: Random forests, disponível em: https://arxiv.org/pdf/1407.7502.pdf, acessado em: 8 de maio de 2019.

[3] Sklearn: Random forest classifier, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html, acessado em: 7 de maio de 2019.

[4] Sklearn: Sklearn: KFold, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.model_selection.KFold.html, acessado em: 7 de maio de 2019.

[5] Sklearn: Confusion matrix Module, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.confusion_matrix.html, acessado em: 7 de maio de 2019.

[6] Sklearn: Plot Confusion matrix Function, disponível em: https://scikit-learn.org/0.17/auto_examples/model_selection/plot_confusion_matrix.html#example-model-selection-plot-confusion-matrix-py, acessado em: 7 de maio de 2019.

[7] Sklearn: Accuracy Module, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.accuracy_score.html, acessado em: 7 de maio de 2019.

[8] Sklearn: F1 Score, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html, acessado em: 7 de maio de 2019.

[9] Sklearn: Precision, disponível em: https://scikit-learn.org/stable/modules/generated/sklearn.metrics.precision_score.html, acessado em: 7 de maio de 2019.