Fundamentos de Sistemas Inteligentes Projeto 2: Experimentos de Classificação com Florestas Randômicas

Gabriel Correia de Vasconcelos
Departamento de Ciência da Computação
Universidade de Brasília
Campus Universitário Darcy Ribeiro, DF
gcvasconcelos98@gmail.com

Raphael Luís Souza de Queiroz
Departamento de Ciência da Computação
Universidade de Brasília
Campus Universitário Darcy Ribeiro, DF
rapha95@gmail.com

***Resumo*—Segundo projeto da matéria de Fundamentos de Sistemas Inteligentes com o objetivo de fazer experimentos de classificação com variações do algoritmo de Floresta Randômica, utilizando o conjunto de dados relacionados a espécies de folhas e seus atributos.**

***Index Terms*—classificação, floresta randômica, python**

I. INTRODUÇÃO

O projeto consiste em experimentos utilizando o algoritmo de Floresta Randômica para classificar espécies de plantas. Com o objetivo de entender melhor os parâmetros do algoritmo, serão aplicadas três variações, que terão suas acurácias avaliadas e comparadas, com o fim de se escolher o modelo que classifica corretamente a espécie de cada folha, a partir apenas de seus atributos.

O *dataset* possui informações de 40 espécies possíveis de folhas, cada uma com 14 diferentes atributos, como textura, comprimento, solidez, entre outros, o que o torna multivariado. Há várias observações de cada espécie, totalizando 340 observações. O *dataset* foi criado como um projeto de mestrado por Pedro Filipe Barros Silva com base nos espécimes coletados por Rubim Almeida da Silva na Universidade de Porto, Portugal.

Para a realização dos experimentos a equipe escolheu a linguagem *Python* devido a sua conhecida facilidade para manipular dados e sua ampla disponibilidade de bibliotecas para nos auxiliar na utilização do algoritmo. As principais bibliotecas usadas foram: pandas, para utilizar sua estrutura de dados *dataframe*, *matplotlib*, para a visualização de alguns gráficos relacionados ao experimento; *sklearn*, para utilizar o algoritmo de floresta randômica, funções de divisão do *dataframe* em treino e teste, normalização dos dados e validação cruzada; entre outras, para usos mais pontuais no código.

Apesar das bibliotecas possuírem funções que fazem a maior parte do trabalho de classificação, é extremamente necessário o conhecimento dos conceitos teóricos que rodam por trás do algoritmo, para entender o que são os parâmetros de cada função e como seu retorno é afetado através de alterações.

II. CONCEITOS TEÓRICOS

A. Classificação e classificadores

O problema da classificação consiste na identificação de qual grupo pré-definido uma observação pertence, baseado na similaridade de seus critérios com os do grupo. Esses critérios são um conjunto de propriedades quantitativas pertencentes a cada observação e podem ser categóricos (e.g. "*Quercus suber*", "*Salix atrocinera*", "*Populus nigra*"), ordinais (e.g "ruim", "médio", "bom") ou números, sejam eles valores inteiros ou reais.

Outro nome para a classificação é reconhecimento de padrões, pois o classificador (F(x)) é função matemática que diz a qual grupo pertence a observação (y) baseado em padrões identificados em seus critérios (x), que são organizados em grupos. Nesse modelo, o Y é sempre discreto.

$$y = F(x) \tag{1}$$

A classificação ainda pode ser divida entre binária ou multi-classe. Na binária, a observação só possui duas classificações, ou é ou não é, enquanto na multi-classe ela pode ter várias categorias.

Na linguagem de aprendizado de máquina, algoritmos de classificação necessitam de um aprendizado supervisionado, ou seja, é necessária a existência de um conjunto de dados de treino, com os valores dos critérios e da classificação da observação já conhecidos, para a predição da classificação de novos dados.

Por convenção, sempre uma pequena parte da base de dados, chamada de dados de teste, é separada para o teste da acurácia do classificador escolhido. Outra forma de visualizar a acurácia do modelo estatístico escolhido e ainda mais identificar mais facilmente falsos positivos e a distribuição das predições é a utilização de matrizes de confusão. Esse é uma matriz de valores reais e valores preditos onde a diagonal corresponde a predições acertadas e as outras posições foram falsos positivos, classificações que o classificador achou que estavam certas, mas ao comparar com o Y do dado de treino elas se mostraram erradas.

Análise de crédito em bancos dado os dados do cliente e proposta de empréstimo. Diagnóstico de doenças dado os sintomas e exames do paciente. Reconhecimento de faces, dado diversas fotos digitais da pessoa. Todos esses são problemas muito conhecidos no mundo do aprendizado de máquina e que suas soluções são algoritmos de classificação.

No caso do problema atual, os atributos da base de dados são categóricos, de classificação multi-classe, com um modelo de aprendizado supervisionado baseado em árvores de decisão.

B. *Árvores de Decisão*

A árvore, na ciência da computação, é um estrutura de dados formada por elementos que guardam alguma informação ou regra, chamados de nós, conectados entre si de forma hierárquica por galhos ou arestas. O primeiro nó é chamado de raiz, que tem seus filhos (próximos nós na escala hierárquica) até que cheguem aos últimos nós, que são chamados de folhas, ou seja, não possuem filhos.

Já árvore de decisão é um modelo que utiliza a estrutura da árvore para algoritmos de aprendizagem, muito utilizada para classificação e regressão. Nela, seus nós armazenam os atributos e suas folhas armazenam a classe predita para aquela observação. O caminho entre a raiz e a folha é definido pelo valor daquele atributo do nó, define qual será o nó filho destino.

Como a aprendizagem é supervisionada, define-se qual atributo cairá em qual nó, ou seja qual terá maior ordem na hierarquia de nós, baseado em um cálculo de ganho de informação. O ganho de informação é a redução esperada na entropia do conjunto de atributos, para medir qual dos atributos é o que pesa mais na classificação da observação. O ganho pode ser representado pela formula 2, onde E(x) é a entropia, representada pela fórmula 3, S é o conjunto de atributos, e X é o conjunto de possíveis classificações.

$$Ganho(S, X_i) = E(S) - \sum_j \frac{S_{x_i j}}{S} E(S_{x_i j}) \tag{2}$$

$$E(x) = \sum_{i=1}^{n} p(x_i) \log_2 p(x_i) \tag{3}$$

A figura 1 mostra um exemplo onde precisa classificar os clientes de um banco que terão seu aumento do limite de crédito aprovada ou não. Com os dados dos clientes do banco, o algoritmo calculou o atributo ”Idade”como o que trazia mais ganho de informação e por isso ele é a raiz da árvore. Os próximos nós definem os outros dois atributos e dependendo de seu valor, levam a uma folha que define a classe final do cliente, ou seja, se ele terá seu limite aprovado ou não.

C. *Floresta Randômica e Bagging*

Apesar do algoritmo de árvore de decisão funcionar para modelos simples, ela tem acurácia de classificação muito ruim para modelos mais complexos, pois elas podem sofrer alta variância. Se gerarmos dois subconjuntos aleatórios do mesmo dado de treinamento e aplicarmos um modelo de árvore de decisão em ambas, suas árvores resultantes podem ser muito

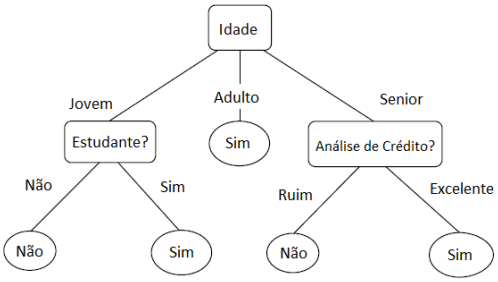

Figura 1: Ilustração de uma árvore de decisão simples, com três atributos

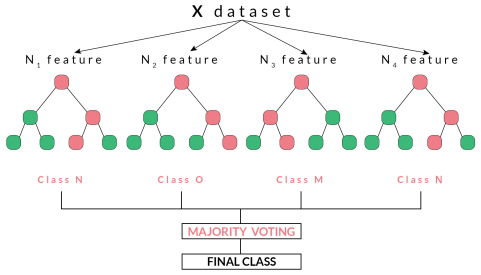

Figura 2: Ilustração de uma floresta randômica genérica

diferentes. Entretanto, queremos modelos com baixa variância, pois em classificações mais complexas trazem uma maior confiabilidade para o modelo.

Uma solução para reduzir a variância é aplicar a técnica do *bagging*, acrônimo para *bootstrap agregatting*, método proposto por Breiman em 1996. A ideia desta técnica é gerar novos conjuntos de dados a partir do conjunto original, de forma aleatória e com reposição, ou seja, uma observação pode ser escolhida novamente e ser repetida no mesmo subconjunto.

Em média, cada subconjunto faz uso de 2/3 das observações para treino, sobrando 1/3 para teste do modelo. O então modelo é aplicado em cada um desses subconjuntos e a predição final é dado pelo voto de maioria entre todos os resultados de classificação dos subconjuntos.

O algoritmo de Floresta Randômica utiliza esse método do *bagging* para gerar várias árvores aleatórias, onde os atributos/preditores em cada árvores *bagged* também são escolhidos aleatoriamente, usualmente um total de $\sqrt{N_{preditores}}$. Este método reduz um possível viés que possa existir de um preditor mais forte pois descorrelaciona as árvores e seus atributos, aumentando ainda mais a confiabilidade do modelo. A predição final usa o voto de maioria das árvores de decisão *bagged*.

O funcionamento do algoritmo pode ser observado na figura 2.

D. *Validação Cruzada*

No primeiro projeto, aprendemos a separar uma pequena parte da base de dados, chamada de dados de teste, para o

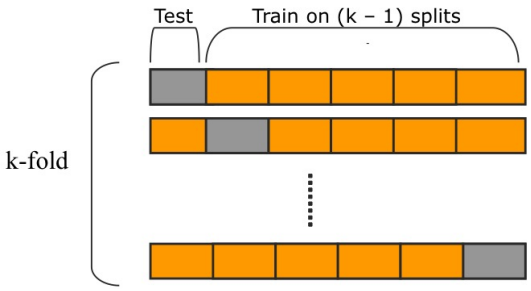

Figura 3: Ilustração do funcionamento da validação cruzada utilizando o k-fold

teste da acurácia do classificador escolhido, treinado com o resto dos dados, chamados de dados de treino.

No entanto, esta técnica é muito limitada pois limitamos o modelo apenas a aqueles dados de treino, o que aumenta a dependência do modelo nos dados e diminui sua robustez. Uma maneira de aumentar essa robustez é fazer uma validação cruzada, que cria várias combinações do mesmo conjunto de dados e aplica o modelo neles.

Uma das técnica técnicas de validação cruzada, chamada de *K-fold*, divide os dados em K subconjuntos (*folds*), treina o algoritmo com K-1 deles e testa sua acurácia com apenas um. esse processo é realizado K vezes, sempre alternando o subconjunto de teste, como pode ser observado na figura 3.

III. EXPERIMENTOS E RESULTADOS

A. Pré-processamento e escolha dos parâmetros

Para a execução do trabalho, foram importados os dados da url: https://archive.ics.uci.edu/ml/machine-learning-databases/00288/. Os dados são baixados em um arquivo csv e lidos durante o código e preparados para a classificação. Para que seja classificado sem que haja nenhum problema, os dados recebidos do arquivo csv são normalizados. Após a normalização, os dados estão prontos para serem classificados.

O processamento dos dados foi dividido em Treino e Teste, onde foi utilizado a classificação em $RandomForest$, e o KFold para validação da classificação, pois este método de validação diminui a variância obtida.

O processo de classifição foi realizado 3 (três) vezes utilizando alguns argumentos a mais para melhorar na classificação. Para a primeira classificação, foi utilizado os valores padrões da função $RandomForestClassifier()$. São listado todos os argumentos na tabela I:

Argumento	Valor Padrão
n_estimators	10
criterion	$'gini'$
max_features	$'auto'$
max_depth	None
min_samples_split	2
min_samples_leaf	1
min_weight_leaf	0
min_weight_fraction_leaf	0
max_leaf_nodes	None
min_impurity_decrease	0
bootstrap	True
oob_score	False
n_jobs	1
random_state	None
verbose	0
warm_start	False
class_weight	$'balanced'$

Tabela I: Tabela de Argumentos da função $RandomForestClassifier$.

Para a segunda classificação, foi alterado o argumento $min\_impurity\_decrease$ para o mínimo ($1e-7$). Esse argumento funciona na hora da separação dos nós filhos na hora da classificação. Onde, por meio da equação apresentado, os nós que possuírem uma diferença de impureza calculada com um valor acima de $1e-7$ serão separados na hora da formação da árvore de decisão. O valor deste argumento é inversamente proporcional ao valor de acurácia calculado após a classificação, ou seja, quanto maior for o valor mínimo de impureza, menor vai ser a acurácia do código. Para a terceira classificação, o valor de $min\_impurity\_decrease$ foi retornado ao padrão e foi realizado a classificação com a mudança de critério, onde foi substituído para $'entropy'$ (entropia). O critério padrão do $RandomForest$ é o $'gini'$ e utiliza o $GiniImpurity$. Este mede o quão frequente um elemento escolhido randomicamente do modelo seria classificado incorretamente se fosse classificado randomicamente de acordo com a distribuição das classificações no submodelo. O critério de entropia realiza a separação de nós baseado na informação obtida do modelo.

B. Resultados

Após a classificação explicada na seção acima, cada $RandomForest$ gerou uma acurácia diferente, devido aos argumentos modificados. Para obter uma classificação melhor do modelo, foi utilizado a função $StratifiedKFold$, gerando o que é chamado de $score$ da classificação. A função $StratifiedKFold$ foi utilizado para que fosse garantido que tenha todas as folhas no subconjunto classificado. Não era necessário a utilização deste método de validação, pois o próprio $RandomForest$ utiliza o método $bagging$.

Para as 3 (três) classificações efetuadas no código, foram obtidos valores diferentes. Com os valores padrões da função de $RandomForest$, a acurácia no momento executado foi de 0.8235, dando um score de $0.80(+/-0.09)$. Ao classificar mudando o argumento $min\_impurity\_decrease$, a acurácia caiu para 0.6470 e seu score foi de $0.79(+/-0.09)$. Ao classificar utilizando a terceira proposta, a acurácia foi de 0.7794 e o score foi de $0.78(+/-0.13)$.

IV. CONCLUSÃO

Este trabalho teve por objetivo a apresentação do classificador $RandomForest$ e a manipulação do mesmo. Por meio do problema apresentado, foi realizado diversos estudos tanto teóricos quantos práticos (código) do classificador indicado.

Foi observado que o classificador $RandomForest$ possui uma acurácia boa para modelos simples, porém ruim para modelos de complexidade alta. Esta explicação foi observada principalmente pelos argumentos de classificação utilizados na função do classificador. Ao alterar apenas um dos valores padrões da função, o modelo se torna mais complexo e o classificador resulta em uma acurácia significativamente menor. Outra observação para o classificador pode ser vista no momento da execução do código, onde a cada execução, o seed principal muda randomicamente, gerando uma classificação diferente da anterior. Os dados utilizados neste trabalho podem ser considerados simples, pois é verificado que a acurária obtida com os valores padrões do classificador são muito próximos.

O objetivo proposto pelo trabalho foi alcançado. Foi observado os conceitos teóricos sobre o $RandomForest$ e foi possível a observação prática do classificador.

REFERÊNCIAS

[1] BREIMAN, Leo. Bagging predictors. Machine learning, v. 24, n. 2, p. 123-140, 1996.

[2] KOHAVI, R. A study of cross-validation and bootstrap for accuracy estimation and model selection. In: International joint Conference on artificial intelligence. [S.l.: s.n.], 1995. v. 14, p. 1137–1145.

[3] HO, Tin Kam. Random decision forests. In: Proceedings of 3rd international conference on document analysis and recognition. IEEE, 1995. p. 278-282.

[4] Decision tree learning. https://en.wikipedia.org/wiki/Decision_tree_learning#Gini_impurity

[5] Plapinger, Thomas. In: Tuning a Random Forest Classifier. https://medium.com/@taplapinger/tuning-a-random-forest-classifier-1b252d1dde92