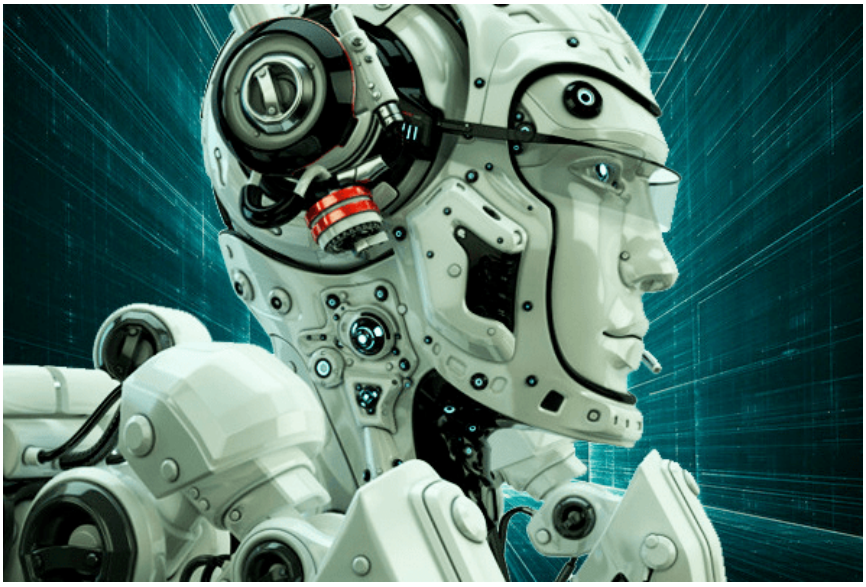
Um tutorial completo sobre modelagem baseada em árvores de decisão (códigos R e Python)

adminvooo 12 de dezembro de 2016 2



Traduzido de: A Complete Tutorial on Tree Based Modeling from Scratch (in R & Python)

Por Analytics Vidhya Content Team

Pesquisa

Posts recentes

Conheça a ValeCard e seus produtos

Policard e Up Brasil – Conheça a história e produtos

Pix: transformando os meios de pagamento

Tudo o que você ainda não sabe sobre a Ticket

GetNet – conheça mais sobre as soluções

Arquivos

- março 2021
- fevereiro 2021
- janeiro 2021
- dezembro 2020

Introdução

Os algoritmos de aprendizagem baseados em árvores de decisão são considerados um dos melhores e mais utilizados métodos de aprendizagem supervisionada. Os métodos baseados em árvores nos dão modelos preditivos de alta precisão, estabilidade e facilidade de interpretação. Ao contrário dos modelos lineares, eles mapeiam muito bem relações não-lineares. E podem ser adaptados para resolver vários tipos de problema (classificação ou regressão).

Métodos como árvores de decisão, floresta aleatória ("random forest") e "gradient boosting" estão sendo muito usados em todos os tipos de problemas de data science. Assim, para os interessados no assunto, é útil aprender esses algoritmos e usá-los em suas modelagens.

Este tutorial destina-se a ajudar no aprendizado da modelagem baseada em árvores. Após a conclusão deste tutorial, espera-se que fique fácil usar algoritmos baseados em árvore e construir modelos preditivos.

Nota: Este tutorial não requer nenhum conhecimento prévio de "machine learning". No entanto, conhecimento básico de R ou Python será útil.

Índice

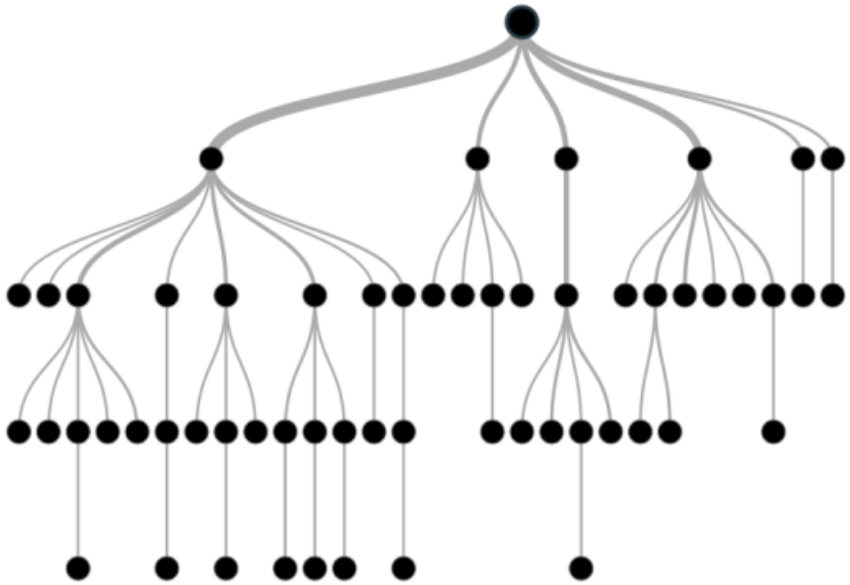
modelagem de árvores e como podemos evitar o sobreajuste nas árvores de decisão?

5. Modelos baseados em árvores são melhores do que modelos lineares?
6. Trabalhando com árvores de decisão em R e Python
7. O que são os métodos de 'ensemble' na modelagem baseada em árvores?
8. O que é "Bagging"? Como funciona?
9. O que é a Floresta Aleatória ("Random Forest")? Como funciona?
10. O que é "Boosting"? Como funciona?
11. Qual é mais poderoso: GBM ou Xgboost?
12. Trabalhando com GBM em R e Python
13. Trabalhando com Xgboost em R e Python
14. Onde praticar?

- setembro 2016
- agosto 2016
- julho 2016
- maio 2016
- abril 2016

1. O que é uma árvore de decisão? Como funciona?

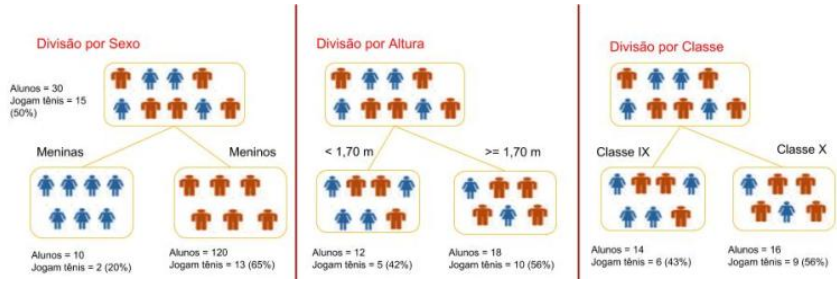
Árvore de decisão é um tipo de algoritmo de aprendizagem supervisionada (com uma variável alvo pré-definida), muito utilizada em problemas de classificação. Ele funciona para ambas as variáveis categóricas e contínuas de entrada e de saída. Na árvore de decisão, dividimos a população ou amostra em dois ou mais conjuntos homogêneos (ou sub-populações) com base nos divisores/diferenciadores mais significativos das variáveis de entrada.



Exemplo:

Digamos que uma amostra de 30 alunos tem três variáveis: Sexo (menino ou menina), Classe (IX ou X) e Altura (160 cm a 180 cm). Digamos também que 15 destes 30 jogam tênis no recreio. A partir disso, como podemos criar um modelo para prever quem vai jogar tênis durante o recreio? Neste problema, precisamos segregar os alunos que jogam tênis no recreio com base nas três variáveis à nossa disposição.

Nesse ponto entra a árvore de decisão. Ela segregará os alunos com base nos valores das três variáveis e identificará a variável que cria os melhores conjuntos homogêneos de alunos (que são heterogêneos entre si). No quadro abaixo, você pode ver que a variável Sexo é capaz de identificar os melhores conjuntos homogêneos em comparação com as variáveis Altura e Classe.



Como mencionado acima, a árvore de decisão identifica a variável mais representativa e os valores que retornam os conjuntos de população mais homogêneos. Agora, a questão é: como identificar essa variável e a sua divisão? Para fazer isso, a árvore de decisão usa vários algoritmos, que discutiremos na próxima seção.

Tipos de Árvores de Decisão

Os tipos de árvore de decisão baseiam-se no tipo de variável de destino que temos. São dois tipos:

1. **Árvore de decisão de variável categórica:** árvore de decisão que tem a variável de destino categórica, chamada assim de árvore de decisão de variável categórica. Exemplo: – No cenário acima do problema dos alunos, onde a variável alvo era “O aluno joga tênis ou não”, os valores são SIM ou NÃO.

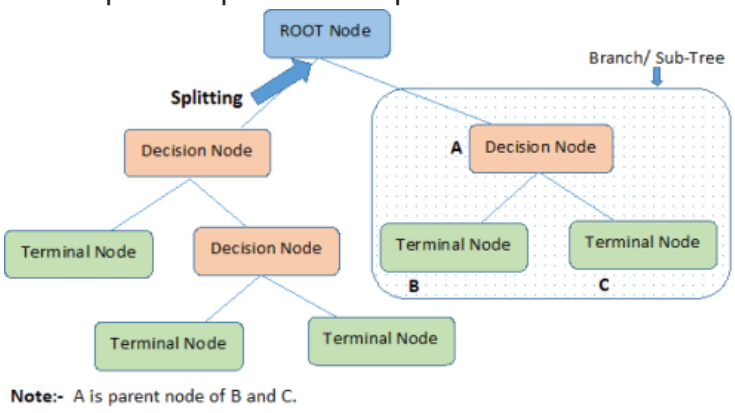
1. **Árvore de decisão de variável contínua:** cuja variável alvo é contínua. Exemplo: – Digamos que o problema seja prever se um cliente vai renovar o prêmio de seguro que paga a uma companhia de seguros (Sim ou Não). Nesse problema, sabemos que a renda do cliente é uma variável significativa, mas a companhia de seguros não conhece a renda de todos seus clientes. Agora, como sabemos que esta

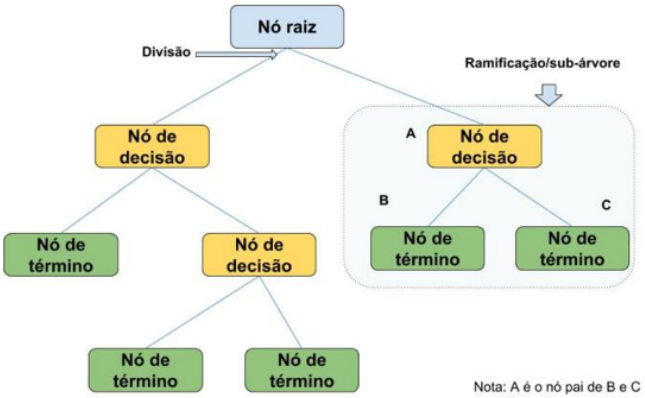
é uma variável importante, então podemos construir uma árvore de decisão para prever a renda do cliente com base na sua ocupação, no produto segurado e várias outras variáveis. Neste caso, estamos prevendo valores para uma variável contínua.

Terminologias relacionadas às árvores de decisão

Vejamos a terminologia básica usada em árvores de decisão:

1. **Nó Raiz:** Representa a população inteira ou amostra, sendo ainda dividido em dois ou mais conjuntos homogêneos.
2. **Divisão:** É o processo de dividir um nó em dois ou mais sub-nós.
3. **Nó de Decisão**: Quando um sub-nó é dividido em sub-nós adicionais.
4. **Folha ou Nó de Término:** Os nós não divididos são chamados Folha ou Nó de Término.
5. **Poda:** O processo de remover sub-nós de um nó de decisão é chamado poda. Podemos dizer que é o processo oposto ao de divisão.





6. **Ramificação/Sub-Árvore:** Uma sub-seção da árvore inteira é chamada de ramificação ou sub-árvore.
7. **Nó pai e nó filho:** Um nó que é dividido em sub-nós é chamado de nó pai. Os sub-nós são os nós filhos do nó pai.

Estes são os termos mais comuns usados para árvores de decisão. Como sabemos que cada algoritmo tem vantagens e desvantagens, abaixo estão alguns fatores importantes para levar em consideração.

Vantagens

1. **Fácil de entender:** A visualização de uma árvore de decisão torna o problema fácil de compreender, mesmo para pessoas que não tenham perfil analítico. Não requer nenhum conhecimento estatístico para ler e interpretar. Sua representação gráfica é muito intuitiva e permite relacionar as hipóteses também facilmente.
2. **Útil em exploração de dados:** A árvore de decisão é uma das formas mais rápidas de identificar as variáveis mais significativas e a relação entre duas ou mais variáveis. Com a ajuda de árvores de decisão, podemos criar novas variáveis/características que tenham

melhores condições de predizer a variável alvo.

3. **Menor necessidade de limpar dados:** Requer menos limpeza de dados em comparação com outras técnicas de modelagem. Até um certo nível, não é influenciado por pontos fora da curva “outliers” nem por valores faltantes (“missing values”).
4. **Não é restrito por tipos de dados:** Pode manipular variáveis numéricas e categóricas.
5. **Método não paramétrico:** A árvore de decisão é considerada um método não-paramétrico. Isto significa que as árvores de decisão não pressupõe a distribuição do espaço nem a estrutura do classificador.

Desvantagens

1. **Sobreajuste (“Over fitting”):** Sobreajuste é uma das maiores dificuldades para os modelos de árvores de decisão. Este problema é resolvido através da definição de restrições sobre os parâmetros do modelo e da poda (discutido em mais detalhes abaixo).
2. **Não adequado para variáveis contínuas:** ao trabalhar com variáveis numéricas contínuas, a árvore de decisão perde informações quando categoriza variáveis em diferentes categorias.

2. Árvores de Regressão versus Árvores de Classificação

Nós de término (ou folhas) estão na parte inferior da árvore de decisão. Isto significa que as árvores de decisão são desenhadas de cabeça para baixo. Dessa

forma, as folhas são o fundo e as raízes são os topos (figura abaixo).

Ambas as árvores trabalham de forma quase semelhantes entre si. Vamos olhar para as diferenças primárias e para as semelhanças entre as árvores de classificação e regressão:

1. As árvores de regressão são usadas quando a variável dependente é contínua. As árvores de classificação são usadas quando a variável dependente é categórica.
2. No caso da árvore de regressão, o valor obtido pelos nós de término nos dados de treinamento é o valor médio das suas observações. Assim, a uma nova observação de dados atribui-se o valor médio correspondente.
3. No caso da árvore de classificação, o valor (classe) obtido pelo nó de término nos dados

de treinamento é a moda das suas observações. Assim, a uma nova observação de dados atribui-se o valor da moda correspondente.

4. Ambas as árvores dividem o espaço preditor (variáveis independentes) em regiões distintas e não sobrepostas. Por uma questão de simplicidade, você pode pensar nessas regiões como caixas de alta dimensão ou simplesmente caixas.
5. Ambas as árvores seguem uma abordagem "top-down" e "gananciosa" conhecida como divisão binária recursiva ("recursive binary splitting"). É chamada de "top-down" porque começa do topo da árvore quando todas as observações estão disponíveis em uma única região e, sucessivamente, divide o espaço preditor em dois novos ramos no sentido inferior da árvore. É conhecida como "gananciosa" porque o algoritmo se preocupa apenas com a divisão atual (procura a melhor variável disponível) e não com divisões futuras, que poderiam levar a uma árvore melhor.
6. Esse processo de divisão é continuado até que um critério de parada definido pelo usuário seja alcançado. Por exemplo: podemos dizer ao algoritmo que pare uma vez que o número de observações por nó torne-se menor que 50.
7. Em ambos os casos, o processo de divisão resulta em árvores totalmente crescidas até que os critérios de parada sejam atingidos. Mas, a árvore totalmente crescida é susceptível de sobrecarga de dados, levando a má precisão em dados não conhecidos. Isso

nos traz à 'poda'. A poda é uma das técnicas usadas para lidar com o sobreajuste. Veremos mais sobre isso na próxima seção.

3. Como fazer a divisão dos nós de uma árvore?

A decisão de fazer as divisões dos nós afeta muito a precisão de uma árvore. Os critérios de decisão são diferentes para árvores de classificação e de regressão.

Árvores de decisão usam vários algoritmos para decidir dividir um nó em dois ou mais sub-nós. A criação de sub-nós aumenta a homogeneidade dos sub-nós resultantes. Em outras palavras, pode-se dizer que a pureza do nó aumenta em relação à variável alvo. A árvore de decisão divide os nós em todas as variáveis disponíveis e seleciona a divisão que resulta em sub-nós mais homogêneos.

A seleção do algoritmo também é baseada no tipo de variáveis de destino. Vejamos os quatro algoritmos mais usados na árvore de decisão:

Índice Gini

O índice Gini diz que se selecionarmos aleatoriamente dois itens de uma população, então ambos devem ser da mesma classe e a probabilidade disto é 1 se a população for pura.

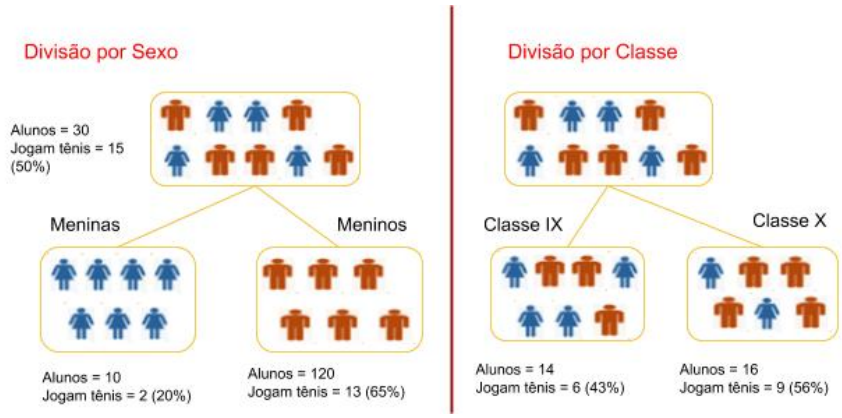
1. Funciona com a variável alvo categórica "Sucesso" ou "Falha"
2. Executa apenas divisões binárias
3. Quanto maior o valor de Gini, maior a homogeneidade

4. O CART (Árvore de Classificação e Regressão) usa o método Gini para criar divisões binárias

Passos para calcular o Gini de uma divisão:

1. Cálculo do Gini para sub-nós, calcule a soma dos quadrados da probabilidade de sucesso e e da de fracasso (p ^ 2 + q ^ 2)
2. Calcule o Gini para a divisão, use a pontuação de Gini ponderada de cada nó dessa divisão

Exemplo: vamos usar exemplo usado acima, onde queremos separar os alunos com base na variável alvo (jogam tênis ou não). No quadro abaixo, dividimos a população usando duas variáveis de entrada – Sexo e Classe. Agora, quero identificar qual divisão produz os sub-nós mais homogêneos usando o índice de Gini.



Divisão por Sexo

1. Gini para o sub-nó Feminino = (0,2 ^ 2) + (0,8 ^ 2) = 0,68
2. Gini para o sub-nó Masculino = (0,65 ^ 2) + (0,35 ^ 2) = 0,55
3. Gini ponderado da Divisão por Sexo = (10 / 30) * 0,68 + (20 / 30) * 0,55 = **0,59**

Divisão por Classe

1. Gini para sub-nó Classe IX = (0,43 ^ 2) + (0,57 ^ 2) = 0,51
2. Gini para sub-nó Classe X = (0,56 ^ 2) + (0,44 ^ 2) = 0,51
3. Calcular o Gini ponderado para a classe dividida = (14 / 30) * 0,51 + (16 / 30) * 0,51 = **0,51**

Acima, vemos que a pontuação de Gini para a Divisão por Sexo é maior do que a Divisão por Classe. Logo, a divisão de nó acontecerá na Divisão por Sexo.

Qui-Quadrado

É um algoritmo para descobrir a significância estatística entre as diferenças dos sub-nós e do nó pai. O qui-quadrado é medido pela soma dos quadrados das diferenças entre as freqüências observadas e esperadas da variável alvo.

1. Funciona com a variável alvo categórica "Sucesso" ou "Falha".
2. Pode executar duas ou mais divisões.
3. Quanto maior o valor do qui-quadrado, maior é a significância estatística das diferenças entre sub-nó e nó pai.
4. O qui-quadrado de cada nó é calculado usando a fórmula:
    1. Qui-quadrado = ((Real – Esperado) ^ 2 / Esperado) ^ (1 / 2)
5. Gera a árvore chamada de CHAID (Chi-square Automatic Interaction Detector)

Passos para calcular o Qui-quadrado para uma divisão

1. Calcule o qui-quadrado para o nó individual

calculando o desvio para ambos Sucesso e Falha

2. Calcule o qui-quadrado da divisão usando a soma de todos os qui-quadrados de Sucesso e Falha de cada nó da divisão

Exemplo: vamos trabalhar com o exemplo acima que usamos para calcular o índice Gini

Divisão por Sexo

1. Primeiro vamos fazer o nó feminino. Preencha o valor real para "Joga tênis" e "Não joga tênis". No caso, temos 2 e 8, respectivamente.
2. Calcule o valor esperado de "Joga tênis" e "Não joga tênis". Aqui seria 5 para ambos porque o nó pai tem probabilidade de 50% e aplicamos a mesma probabilidade na contagem de meninas (10).
3. Calcule os desvios usando a fórmula (Real – Esperado). Para "Jogar tênis" é (2 – 5) = -3 e para "Não jogar tênis" é (8 – 5) = 3.
4. Calcule o Qui-quadrado dos nós "Joga tênis" e "Não jogar tênis" usando a fórmula ((Real – Esperado) ^ 2 / Esperado) ^ (1 / 2). A tabela abaixo mostra o cálculo.
5. Siga os mesmos passos para calcular o valor do Qui-quadrado do nó masculino.
6. Por último, adicione todos os valores de Qui-quadrado para calcular o Qui-quadrado da Divisão por Sexo.

Nó	Joga tênis	Não joga tênis	Total	Expectativa joga tênis	Expectativa não joga tênis	Desvio joga tênis	Desvio não joga tênis	Qui-quadrado	
								Joga tênis	Não joga tênis
Meninas	2	8	10	5	5	-3	3	1,34	1,34
Meninos	13	7	20	10	10	3	-3	0,95	0,95
							Qui-quadrado Total	4,58	

Divisão por Classe

Execute os mesmos cálculos da divisão por Sexo e o resultado será o da tabela abaixo.

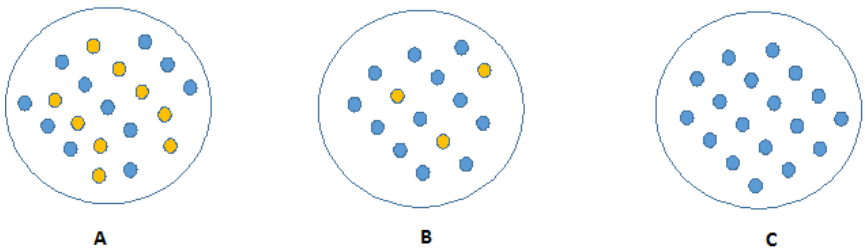
Nó	Joga tênis	Não joga tênis	Total	Expectativa joga tênis	Expectativa não joga tênis	Desvio joga tênis	Desvio não joga tênis	Qui-quadrado	
								Joga tênis	Não joga tênis
IX	6	8	14	7	7	-1	1	0,38	0,38
X	9	7	16	8	8	1	-1	0,35	0,35
						Qui-quadrado Total		1,46	

Na tabela acima, pode-se ver que o Qui-quadrado também identifica a Divisão por Sexo com mais significativa que a Divisão por Classe.

As tabelas podem ser consultadas aqui.

Ganho de informação:

Olhe para a imagem abaixo, onde cada conjunto é um nó, e pense qual dos nós pode ser descrito mais facilmente. Com certeza, a resposta é C porque requer menos informações, posto que todos os valores são idênticos. Por outro lado, B requer mais informações para ser descrito e A exige o máximo de informação. Em outras palavras, podemos dizer que C é um nó puro, B é menos impuro e A é mais impuro.



Agora, podemos concluir que um nó menos impuro requer menos informação para ser descrito enquanto um nó mais impuro requer mais informação. A teoria da informação dá uma medida para definir esse grau de desorganização em sistema – entropia. Se a

amostra for completamente homogênea, então a entropia é zero. Se a amostra for dividida em partes iguais (50% – 50%), então terá entropia de um.
A entropia pode ser calculada usando a fórmula da entropia ("Entropy"):

$$\text{Entropy} = -p \log_2 p - q \log_2 q$$

Aqui p e q são as probabilidades de sucesso e falha, respectivamente, para um dado nó. A entropia também é usada com a variável alvo categórica. É escolhida a divisão que tem a entropia mais baixa em comparação com o nó pai e com outras divisões. Quanto menor for a entropia, melhor.

Passos para calcular entropia de uma divisão:

1. Calcule a entropia do nó pai
2. Calcule a entropia de cada nó individual e calcule a média ponderada de todos os sub-nós disponíveis na divisão

Exemplo: Vamos usar esse método para identificar a melhor divisão para o exemplo dos alunos que jogam tênis.

1. Entropia para o nó pai: – (15/30) log2 (15/30) – (15/30) log2 (15/30) = **1**. (Isso mostra que é um nó impuro.)
2. Entropia para o nó feminino: – (2/10) log2 (2/10) – (8/10) log2 (8/10) = 0,72. E para o nó masculino: – (13/20) log2 (13/20) – 7/20) log2 (7/20) = **0,93**
3. Entropia para a Divisão por Sexo: entropia ponderada de sub-nós: (10/30) * 0,72 + (20/30) * 0,93 = **0,86**

4. Entropia para o nó Classe IX: (6/14) log2 (8/14) log2 (8/14) = 0,99. E para o nó da Classe X: – (9/16) log2 (9/16) – (7/16) log2 (7/16) = 0,99.
5. Entropia para a Divisão por Classe: (14/30) * 0,99 + (16/30) * 0,99 = **0,99**

Acima, você pode ver que entropia da Divisão por Sexo é a mais baixa de todas, então a árvore vai ser dividida por sexo. Podemos derivar o ganho de informação da entropia como **1- Entropia**.

Redução na Variância

Até agora, discutimos os algoritmos para a variável alvo categórica. A redução na variância é um algoritmo usado para variáveis alvo contínuas (problemas de regressão). Este algoritmo utiliza a fórmula padrão de variância para escolher a melhor divisão. A divisão com menor variância é selecionada como critério para dividir a população:

$$\text{Variance} = \frac{\Sigma(X - \overline{X})^2}{n}$$

Na fórmula acima, X-barra é a média dos valores, X é o real e n é o número de valores.

Passos para calcular a variância

1. Calcule a variância para cada nó
2. Calcule a variância para cada divisão como média ponderada de cada variância do nó

Exemplo: Vamos atribuir o valor numérico 1 para jogar tênis e 0 para não jogar tênis. Agora siga as etapas para identificar a divisão correta:

1. Variância para o nó Raiz, aqui o valor médio é (15 * 1 + 15 * 0) / 30 = 0,5 e temos 15 uns e 15 zeros. Agora a variância seria ((1-0.5) ^ 2 + (1-0.5) ^ 2 + ... .15 vezes + (0-0.5) ^ 2 + (0-0.5) ^ 2 + ... 15 vezes) / 30, isto pode ser escrito como (15 * (1-0.5) ^ 2 + 15 * (0-0.5) ^ 2) / 30 **= 0,25**
2. Média do nó feminino: (2 * 1 + 8 * 0) / 10 = 0,2 e variância: (2 * (1-0,2) ^ 2 + 8 * (0-0,2) ^ 2) / 10 = 0,16
3. Média do nó masculino: (13 * 1 + 7 * 0) / 20 = 0,65 e variância: (13 * (1-0,65) ^ 2 + 7 * (0-0,65) ^ 2) / 20 = 0,23
4. Variância para Divisão por Sexo: Variância Ponderada dos Sub-nós: (10/30) * 0,16 + (20/30) * 0,23 = **0,21**
5. Média do nó da classe IX: (6 * 1 + 8 * 0) / 14 = 0,44 e Variância: (6 * (1-0,43) ^ 2 + 8 * (0-0,43) ^ 2) / 14 = 0,24
6. Média do nó da classe X: (9 * 1 + 7 * 0) / 16 = 0,56 e variância: (9 * (1-0,56) ^ 2 + 7 * (0-0,56) ^ 2) / 16 = 0,25
7. Variância para Divisão por Classe: (14/30) * 0,24 + (16/30) * 0,25 = **0,25**

Acima, você pode ver que a Divisão por Sexo tem variação menor comparada ao nó pai, então a divisão ocorreria na variável Sexo.

Até aqui, aprendemos sobre os conceitos básicos das árvores de decisão e o processo de tomada de decisão envolvido para escolher as melhores divisões na construção de um modelo de árvore. Como dito, a árvore de decisão pode ser aplicada em problemas de regressão e de classificação. Vamos entender esses aspectos em mais detalhes.

4. Quais são os parâmetros-chave na modelagem de árvores e como podemos evitar o sobreajuste nas árvores de decisão?

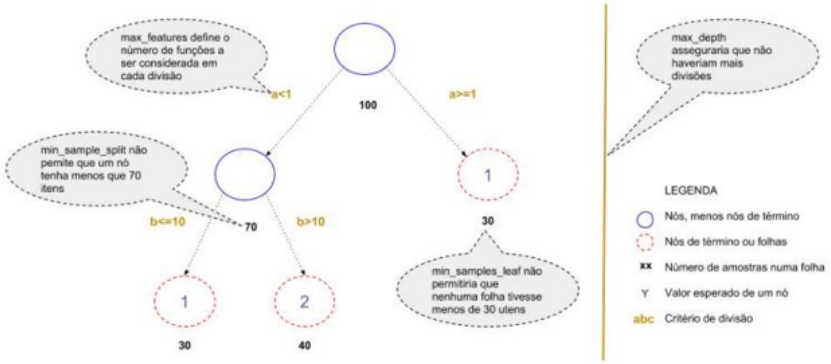
Sobreajuste (“Overfitting”) é um dos principais desafios enfrentados ao modelar árvores de decisão. Se não forem estabelecidos limites a uma árvore de decisão, ela acabará dando 100% de precisão no conjunto de treinamento, porque no pior caso ela vai acabar fazendo uma folha para cada observação. Por isso, prevenir o sobreajuste é fundamental ao se modelar uma árvore de decisão. Isso pode ser feito de duas maneiras:

1. Definindo restrições no tamanho da árvore
2. Podando a árvore

Vamos discutir um pouco sobre ambas.

Definindo Restrições no Tamanho da Árvore

Isso pode ser feito estabelecendo alguns dos parâmetros usados para definir uma árvore. Primeiro, vamos olhar para a estrutura geral de uma árvore de decisão



Vamos ver os parâmetros utilizados para definir uma árvore. Os parâmetros descritos a seguir são independentes da ferramenta. É importante compreender o papel dos parâmetros utilizados na modelagem de árvores. Estes parâmetros estão disponíveis em R & Python.

1. **Número mínimo de amostras para uma divisão de nós**
    1. Define o número mínimo de amostras (ou observações) que são necessárias para considerar dividir um nó
    2. Usado para controlar o sobreajuste. Valores mais elevados impedem o modelo de aprender sobre relações que podem ser muito específicas da amostra selecionada para a árvore
    3. Valores muito elevados podem também levar a sub-ajuste. Portanto, deve ser ajustado usando CV (nota do tradutor: CV = cross-validation)
2. **Número mínimo de amostras para um nó de término (folha)**
    1. Define o número mínimo de amostras (ou observações) necessárias em um nó terminal ou folha
    2. Usado para controlar sobreajuste semelhante ao min_sample_split
    3. Em geral, deve-se usar valores mais baixos para problemas de classe mais desbalanceados, porque serão muito pequenas as regiões em que a classe minoritária será majoritária
3. **Profundidade máxima da árvore**

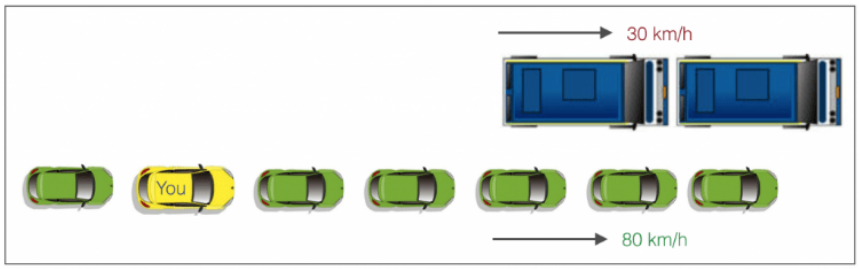
(profundidade vertical)

1. Define a profundidade máxima de uma árvore
2. Usado para controlar sobreajuste, pois uma maior profundidade permitirá ao modelo aprender relações cada vez mais específicas da amostra
3. Deve ser sintonizado usando CV (nota do tradutor: CV = cross-validation)

4. **Número máximo de nós de término**
    1. Define o número máximo de nós de término ou de folhas
    2. Pode ser definido no lugar de max_depth. Se as árvores criadas forem binárias, uma profundidade de 'n' produziria um máximo de 2 ^ n folhas
5. **Número máximo de atributos a considerar para uma divisão**
    1. O número de recursos a serem considerados ao procurar uma melhor divisão. Estes recursos serão selecionados aleatoriamente
    2. Como regra genérica, a raiz quadrada do número total de recursos funciona bem, mas devemos verificar até 30-40% do número total de recursos
    3. Valores mais altos podem levar a um sobreajuste, mas isso varia de caso a caso

Podando ("pruning") a árvore

Como discutido anteriormente, a técnica de configurar restrições obedece a uma abordagem chamada de gananciosa. Em outras palavras, irá verificar a melhor divisão instantaneamente e irá avançando até que uma das restrições especificadas seja atendida. Vamos considerar o seguinte caso quando se está dirigindo:



Existem 2 pistas:

1. Uma pista com carros a 80km/h
2. Uma pista com caminhões a 30km/h

Neste instante, você é o carro amarelo e você tem 2 escolhas:

1. Virar à esquerda e ultrapassar os outros 2 carros rapidamente
2. Manter-se na mesma faixa atual

Vamos analisar essas duas opções. Na primeira, você vai imediatamente ultrapassar o carro à frente, vai chegar atrás do caminhão e vai começar a mover-se a 30 km/h, ficando à espera de uma oportunidade de voltar para a pista da direita. (Todos os carros originalmente atrás de você irão ultrapassá-lo enquanto isso.) Esta primeira opção seria a escolha ideal se o seu objetivo for maximizar a distância percorrida nos próximos 10 segundos. Na segunda opção, você mantém os 80 km/h, passa os caminhões e, dependendo da situação adiante, pode tentar

ultrapassar os carros.

A diferença entre a primeira e a segunda opção é exatamente a diferença entre a árvore de decisão normal e a poda. Uma árvore de decisão com restrições não verá o caminhão adiante e adotará uma abordagem gananciosa tomando a pista da esquerda. Por outro lado, se usamos a poda, na verdade olhamos alguns passos à frente e fazemos uma escolha melhor.

Com isso, podemos dizer que a poda é melhor. Mas como implementá-la na árvore de decisão? A ideia é simples:

1. Primeiro, fazemos a árvore de decisão com um alto grau de profundidade
2. Então, começamos na parte inferior e removemos folhas que estão nos dando retornos negativos quando comparados com a parte superior
3. Suponha que uma divisão está nos dando um ganho de -10 (perda de 10) e, em seguida, a próxima divisão nos dá um ganho de 20. Uma árvore de decisão simples vai parar no passo 1. Mas na poda, vamos ver que no total

o ganho é de +10. Com isso, mantemos ambas as folhas.

Observe que o classificador de árvore de decisão da biblioteca sklearn atualmente não suporta a poda. Pacotes avançados como o xgboost adotaram a poda de árvores em sua implementação. A biblioteca *rpart* do R também fornece uma função para podar. Bom para os usuários de R!

5. Modelos baseados em árvores são melhores do que modelos lineares?

"Se eu posso usar regressão logística para problemas de classificação e regressão linear para problemas de regressão, por que haveria necessidade de usar árvores"? Muitos de nós fazemos essa pergunta. Essa é uma pergunta válida.

Na verdade, você pode usar qualquer algoritmo. Depende do tipo de problema que você está tentando resolver. Vejamos alguns fatores-chave que podem ajudar a decidir qual algoritmo usar:

1. Se a relação entre a variável dependente e a independente for bem aproximada por um modelo linear, a regressão linear terá uma performance melhor que o modelo baseado em árvore
2. Se houver uma relação de alta não linearidade e complexidade entre as variáveis dependentes e as independentes, um modelo de árvore irá superar um método de regressão clássica
3. Se você precisa construir um modelo que seja fácil de explicar às pessoas, um modelo

de árvore de decisão sempre será melhor do que um modelo linear. Os modelos de árvores de decisão são ainda mais simples de interpretar do que a regressão linear!

6. Trabalhando com árvores de decisão em R e Python

Para usuários de R e de Python, a árvore de decisão é bem fácil de implementar. Vejamos rapidamente os códigos com os quais você pode começar o algoritmo. Para facilitar o uso, compartilhei códigos padrão onde você somente precisará substituir o nome do conjunto de dados e as variáveis para começar.

Para usuários de R, existem várias bibliotecas disponíveis para implementar uma árvore de decisão. Dentre elas: ctree, rpart, tree, etc.

```
> library(rpart)
> x <- cbind(x_train, y_train)
# Cresce a árvore
> fit <- rpart(y_train ~ ., data = x,metho
> summary(fit)

#Prevê o resultado
> predicted= predict(fit,x_test)
```

No código acima:

- y_train – representa a variável dependente
- x_train – representa a variável independente
- x – representa a base de dados de treinamento

Para os usuários de Python:

```
#Importa biblioteca
#Importa outras bibliotecas necessárias co
from sklearn import tree
#Assume que você tem X (preditor) e Y (alv
# Cria o objeto tree
model = tree.DecisionTreeClassifier(criter
# Para classificação, aqui você pode mudar
# model = tree.DecisionTreeRegressor() par
# Treina o modelo usando os dados de trein
model.fit(X, y)
model.score(X, y)
#Prevê o resultado
predicted= model.predict(x_test)
```

7. O que são os métodos de ‘ensemble’ na modelagem baseada em árvores?

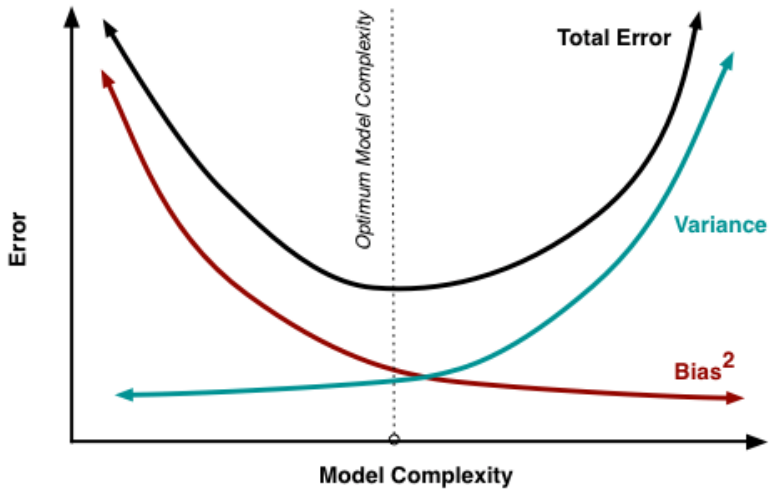
O significado literal da palavra ‘ensemble’ é grupo. Métodos de ‘ensemble’ envolvem agrupar modelos preditivos de modo a melhorar a precisão e a estabilidade do modelo. Métodos de ‘ensemble’ são conhecidos por impulsionar e aprimorar os modelos baseados em árvore.

Como qualquer outro modelo, um modelo baseado em árvore também sofre com viés e variância. Viés significa ‘o quanto em média os valores previstos são diferentes dos valores reais’. Variância significa ‘o quão diferentes serão as previsões do modelo num mesmo ponto se diferentes amostras forem tomadas da mesma população’.

Suponha que você montou uma árvore pequena, obtendo um modelo com baixa variância e viés elevado. Como então equilibrar o trade-off entre viés e variância?

Normalmente, à medida que você aumenta a complexidade de seu modelo, você verá uma redução no erro de previsão devido ao viés mais baixo no modelo. À medida que você continuar tornando o modelo mais complexo ele começará a sofrer com a variância.

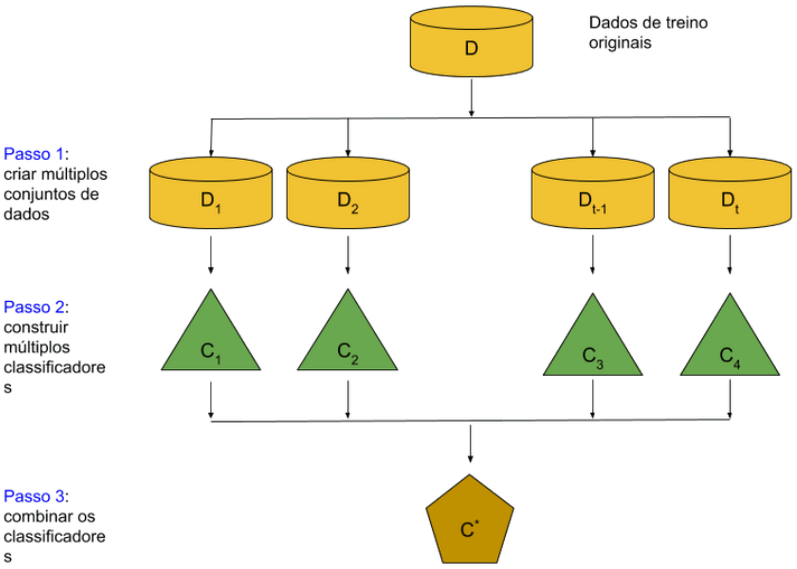
Um modelo ótimo deve manter o equilíbrio entre estes dois tipos de erros. Isto é conhecido como a **gestão de 'trade-off'** entre erros de variância e viés. Aprendizagem por 'ensemble' é uma maneira de analisar esse 'trade-off'.



Alguns dos métodos de 'ensemble' comumente utilizados incluem: 'Bagging', 'Boosting' e 'Stacking'. Neste tutorial, vamos ver em mais detalhes Bagging e Boosting.

8. O que é Bagging? Como funciona?

'Bagging' é uma técnica usada para reduzir a variância das previsões. Ela combina o resultado de vários classificadores, modelados em diferentes sub-amostras do mesmo conjunto de dados. A figura a seguir deixa mais claro:



As etapas do 'Bagging' são as seguintes:

1. Criar vários conjuntos de dados:

- A amostragem é feita com a substituição dos dados originais e a formação de novos conjuntos de dados
- Os novos conjuntos de dados podem ter uma fração das colunas e das linhas, que geralmente são hiper-parâmetros em um modelo de 'bagging'
- Tomando frações de linha e coluna menores que 1 ajuda na montagem de modelos robustos, menos propensos a sobreajuste

2. Criar múltiplos classificadores

- Classificadores são construídos em cada

conjunto de dados

- Em geral, o mesmo classificador é modelado em cada conjunto de dados, e a partir disso as previsões são feitas

3. Combinar classificadores

- As previsões de todos os classificadores são combinadas usando a média, a mediana ou a moda, dependendo do problema
- Os valores combinados são geralmente mais robustos do que um único modelo

Note-se que o número de modelos construídos não são hiper-parâmetros. Um maior número de modelos são geralmente melhores. Ou podem dar um desempenho semelhante ao de números mais baixos. Pode-se mostrar em teoria que a variância das previsões combinadas é reduzida para 1 / n (n: número de classificadores) da variância original, sob algumas premissas.

Existem várias implementações de modelos 'bagging'. A floresta aleatória ('random forest') é uma delas e a discutiremos a seguir.

9. O que é a Floresta Aleatória ("Random Forest")? Como funciona?

Floresta Aleatória é considerada a panacéia de todos os problemas de Data Science. Em outras palavras, quando você não consegue pensar num algoritmo (seja qual for a situação), use 'random forest'!

'Random forest' é um método de aprendizagem de máquina versátil e capaz de executar tarefas de

regressão e de classificação. Ele também aplica métodos de redução dimensional, trata valores faltantes, valores anómalos (‘outliers’) e outras etapas essenciais da exploração de dados. No geral, faz um trabalho muito bom. É um tipo de método de aprendizado de ‘ensemble’, onde um grupo de modelos fracos são combinados para formar um modelo mais forte.

Como isso funciona?

Na floresta aleatória, crescemos múltiplas árvores ao invés de uma única árvore no modelo do CART. Para classificar um novo objeto baseado em atributos, cada árvore dá uma classificação, que é como se a árvore desse “votos” para essa classe. A floresta escolhe a classificação que tiver mais votos (de todas as árvores da floresta) e, em caso de regressão, considera a média das saídas por árvores diferentes.

Funciona da seguinte maneira. Cada árvore é plantada e cultivada da seguinte forma:

1. Assuma que o número de casos no conjunto de treinamento é N. Então, a amostra desses N casos é escolhida aleatoriamente, mas com

substituição. Esta amostra será o conjunto de treinamento para o cultivo da árvore.

2. Se houver M variáveis de entrada, um número m < M é especificado de modo que, em cada nó, m variáveis de M sejam selecionadas aleatoriamente. A melhor divisão nestes m é usada para dividir o nó. O valor de m é mantido constante enquanto crescemos a floresta.
3. Cada árvore é cultivada na maior extensão possível e não há poda.
4. Preveja novos dados agregando as previsões das árvores ntree (ou seja, votos majoritários para classificação, média para regressão).

Vantagens da Floresta Aleatória:

- Este algoritmo pode resolver os problemas de classificação e de regressão, fazendo uma

estimativa decente em ambos.

- Um dos benefícios da floresta aleatória que me agrada mais é o poder de lidar com dados em grandes volumes e com muitas dimensões. Ele pode lidar com milhares de variáveis de entrada e identificar as variáveis mais significativas, sendo por isso considerado um dos métodos de redução de dimensões. Além disso, o modelo produz o grau de importância das variáveis, o que pode ser um dado muito útil (em algum conjunto de dados aleatórios).

- Possui um método eficaz para estimar os dados faltantes e mantém a precisão quando uma grande parte dos dados estão faltando.
- Possui métodos para equilibrar erros em conjuntos de dados onde as classes são desequilibradas.

- As capacidades do método anterior podem ser estendidas para dados sem rótulo, levando a clusters não supervisionados, visualizações de dados e detecção de 'outliers'.
- A floresta aleatória envolve a amostragem dos dados de entrada com substituição chamada como amostragem de 'bootstrap'. Aqui um terço dos dados não é usado para treinamento e pode ser usado para testes. Estes são chamados de amostras de fora da cesta. O erro estimado nas amostras de fora da cesta é conhecido como erro de fora da cesta. O estudo de estimativas do erro de fora da cesta fornece evidências para mostrar que a estimativa de fora da cesta é tão precisa quanto usar um conjunto de teste do mesmo tamanho que o conjunto de treinamento. Portanto, usar a estimativa de erro de fora da cesta remove a necessidade de ter um conjunto de teste extra.

Desvantagens da Floresta Aleatória:

- Enquanto faz um bom trabalho na classificação, já não é tão bom para o problema de regressão, uma vez que não fornece previsões precisas para variáveis contínuas. No caso da regressão, não prevê além do intervalo dos dados de treinamento, e que eles podem sobre-ajustar os conjuntos de dados que tenham muita discrepância ('noisy').
- A floresta aleatória pode ser considerada como uma caixa preta para quem faz modelagem estatística – você tem muito pouco controle sobre o que o modelo faz.

Você pode, na melhor das hipóteses, experimentar diferentes parâmetros.

Implementação em R e Python

As florestas aleatórias têm implementações comumente conhecidas em pacotes R e Python scikit-learn. Vejamos o código de carregamento do modelo de floresta aleatória em R e Python abaixo:

Python

```
#Importa biblioteca
fromsklearn.ensemble importRandomForestCla
#Assumindo que sim, X (preditor) e Y (alvo
# Cria objeto Random Forest
model= RandomForestClassifier(n_estimators
# Treina o modelo usando os dados de trein
model.fit(X, y)
#Prevê o resultado
predicted= model.predict(x_test)
```

R

```
> library(randomForest)
> x <- cbind(x_train,y_train)
# Fitting model
> fit <- randomForest(Species ~ ., x,ntree
> summary(fit)
#Prevê o resultado
> predicted= predict(fit,x_test)
```

10. O que é ‘Boosting’? Como funciona?

Definição: O termo "Boosting" refere-se a uma família de algoritmos que converte uma aprendizagem fraca em uma aprendizagem forte.

Vamos entender esta definição em mais detalhes, resolvendo um problema de identificação de spam. Como você classificaria um e-mail como sendo SPAM ou não? De início, nossa abordagem inicial seria identificar e-mails 'spam' e 'não spam' usando alguns critérios:

1. Se o e-mail tem apenas um arquivo de imagem (imagem promocional) – É um SPAM
2. Se o email tem apenas links – É um SPAM
3. Se o corpo do email consiste na frase como "você ganhou um dinheiro premiado de $ xxxxxx" – É um Spam
4. Se o email é do domínio "Analyticsvidhya.com" – Não é um SPAM
5. Se o email é de uma fonte conhecida – Não é um SPAM

Acima, definimos algumas regras para classificar um e-mail em 'spam' ou 'não spam'. Todavia, você acha que essas regras individualmente são fortes o suficiente para classificar corretamente um e-mail? Não.

Individualmente, essas regras não são poderosas o suficiente para classificar um e-mail em 'spam' ou 'não spam'. Portanto, essas regras são chamadas de aprendizagem fraca.

Para converter a aprendizagem fraca em aprendizagem forte, combinaremos a previsão de cada aprendizagem fraca com métodos como:

- Usando média e média ponderada
- Considerando a previsão que tiver mais votos

Por exemplo: Acima, definimos 5 aprendizagens fracas. Destas 5, 3 são votadas como “SPAM” e 2 são votados como “Não SPAM”. Nesse caso, por padrão, consideraremos um e-mail como SPAM porque temos maior (3) votos para ‘SPAM’.

Como Funciona?

Agora sabemos que ‘boosting’ combina o aprendizado fraco (também conhecido como base de aprendizagem) para formar uma regra forte. De imediato, uma questão que deve aparecer em mente é: “Como o ‘boosting’ identifica regras fracas?”

Para encontrar uma regra fraca, aplicamos algoritmos de base de aprendizagem (ML) com uma distribuição diferente. Cada vez que o algoritmo de base de aprendizado é aplicado, ele gera uma nova regra de previsão fraca. Este é um processo iterativo. Após muitas iterações, o algoritmo de ‘boosting’ combina essas regras fracas em uma única regra de predição forte.

Aqui está outra pergunta que poderia assustar: “Como podemos escolher uma distribuição diferente para cada iteração?”

Para escolher a distribuição correta, siga estas etapas:

Passo 1: A base de aprendizagem toma todas as distribuições e atribui igual peso ou atenção a cada observação.

Passo 2: Se houver qualquer erro de previsão

causado pelo algoritmo de aprendizagem da primeira base, então prestamos maior atenção às observações com erro de previsão. Em seguida, aplicamos o algoritmo de base de aprendizagem seguinte.

Passo 3: Iterar o Passo 2 até que o limite do algoritmo de base de aprendizagem seja atingido ou uma maior precisão seja alcançada.

Por último, o algoritmo combina os resultados da aprendizagem fraca e cria uma aprendizagem forte que eventualmente melhora o poder de predição do modelo. O 'boosting' foca mais em exemplos que são mal classificados ou que têm mais erros por serem precedidos por regras fracas.
Existem muitos algoritmos de 'boosting' que dão um impulso adicional à precisão do modelo. Neste tutorial, aprenderemos sobre os dois algoritmos mais utilizados, ou seja, Gradient Boosting (GBM) e XGboost.

11. Qual é mais poderoso: GBM ou XGBoost?

Eu sempre admirei as capacidades de 'boosting' do algoritmo XGBoost. Às vezes, noto que ele dá um melhor resultado do que o GBM, porém outras vezes você pode achar que os ganhos são apenas marginais. Quando eu explorei mais sobre seu desempenho e a ciência por trás de sua alta precisão, descobri muitas vantagens do XGBoost sobre o GBM:

1. **Regularização**
    1. A implementação padrão do GBM não tem regularização como o XGBoost. Isso também ajuda a reduzir o sobreajuste
    2. Na verdade, XGBoost também é

conhecido como técnica de 'boosting' regularizado.

2. **Processamento Paralelo**
    1. XGBoost implementa processamento paralelo e é incrivelmente mais rápido em comparação ao GBM.
    2. Mas espere aí, sabemos que o 'boosting' é um processo seqüencial, então como pode ser paralelizado? Sabemos que cada árvore pode ser construída apenas após a anterior, então o que nos impede de fazer uma árvore usando todos os núcleos? Espero que você saiba onde estou querendo chegar. Verifique este link para explorar mais a fundo.
    3. O XGBoost também suporta a implementação no Hadoop.
3. **Alta Flexibilidade**
    1. O XGBoost permite aos usuários definir personalizar objetivos de otimização e critérios de avaliação.
    2. Isso adiciona uma nova dimensão ao modelo e não há limite para o que podemos fazer.
4. **Gerenciando valores faltantes ('missing values')**
    1. XGBoost tem uma rotina interna para lidar com valores faltantes.
    2. O usuário é obrigado a fornecer um valor diferente das outras observações e passar isso como um parâmetro. XGBoost tenta coisas diferentes quando encontra um valor faltante em cada nó e aprende que

caminho tomar no futuro para quando encontrar novos valores faltantes.

5. **Poda da árvore**
    1. Um GBM pararia de dividir um nó quando encontrasse uma perda negativa na divisão. Assim, é mais um algoritmo ganancioso.
    2. XGBoost por outro lado faz splits até o max_depth especificado e, em seguida, começa a podar a árvore de trás pra frente e remove as divisões para além das quais não há ganho positivo.
    3. Outra vantagem é que às vezes uma divisão de perda negativa de -2 pode ser seguida por uma divisão de perda positiva +10. O GBM pararia quando encontrasse -2. Mas o XGBoost vai ir mais fundo e vai ver um efeito combinado de +8 na divisão e irá manter ambos.
6. **Validação cruzada**
    1. O XGBoost permite ao usuário executar uma validação cruzada em cada iteração do processo de 'boosting' e, portanto, é fácil obter o número exato ideal de iterações de 'boosting' em uma única execução.
    2. Isso é diferente do GBM, onde temos de executar uma pesquisa de grade e somente alguns valores limitados podem ser testados.
7. **Continuar no modelo existente**
    1. O usuário pode começar a treinar um modelo XGBoost a partir da última

iteração da execução anterior. Isto pode ser uma vantagem significativa em certas aplicações específicas.

2. A implementação GBM do sklearn também tem esse recurso, o que o deixa em igualdade com o XGBoost este ponto.

12. Trabalhando com GBM em R e Python

Antes de pormos a mão na massa, vamos entender rapidamente os parâmetros mais importantes e o funcionamento desse algoritmo. Isso será útil para os usuários de R e de Python. Abaixo está o pseudo-código geral do algoritmo GBM para 2 classes:

```
1. Inicialize o resultado
2. Itere de 1 até o número total de árvore
  2.1 Atualize os pesos dos objetivos com
  2.2 Ajuste o modelo na subamostra de dad
  2.3 Faça previsões sobre o conjunto comp
  2.4 Atualize a saída com resultados atua
3. Retorne o resultado final.
```

Esta é uma explicação extremamente simplificada (provavelmente ingênua) do funcionamento do GBM. Mas, ele vai ajudar todos os iniciantes a entender o algoritmo.

Vamos verificar os parâmetros GBM mais importantes para melhorar a performance do modelo no Python:

1. **learning_rate**

- Determina o impacto de cada árvore no resultado final (etapa 2.4). GBM funciona iniciando com uma estimativa inicial que é atualizada usando a saída de cada árvore. O parâmetro de aprendizagem controla a magnitude desta alteração nas estimativas.
- Valores mais baixos são geralmente preferidos porque tornam o modelo robusto em relação às características específicas da árvore e assim permitem que ela se generalize bem.
- Valores mais baixos exigiriam maior número de árvores para poder modelar todas as relações e são computacionalmente mais caras.

2. **n_estimators**
    - O número de árvores sequenciais a serem modeladas (passo 2)
    - Embora GBM seja bastante robusto com um maior número de árvores, ainda assim pode sofrer de sobreajuste em algum ponto. Portanto, isso deve ser ajustado usando CV para uma determinada taxa de aprendizagem
3. **subsample**
    - A fração de observações a ser selecionada para cada árvore. A seleção é feita por amostragem aleatória.
    - Valores ligeiramente inferiores a 1 tornam o modelo robusto, reduzindo a variância.

- Valores tipicamente ~ 0,8 geralmente funcionam bem, mas podem ser ajustados ainda mais.

Além destes, existem alguns outros parâmetros que afetam a funcionalidade geral:

1. **loss**
    - Refere-se à função de perda a ser minimizada em cada divisão.
    - Pode ter vários valores para os casos de classificação e de regressão. Geralmente os valores padrão funcionam bem. Outros valores devem ser escolhidos somente se você entender seu impacto no modelo
2. **init**
    - Isso afeta a inicialização da saída
    - Isso pode ser usado se tivermos feito outro modelo cujo resultado deve ser usado como as estimativas iniciais para GBM.
3. **random_state**
    - O número aleatório semente, de modo que os mesmos números aleatórios são gerados cada vez.
    - Isso é importante para 'tunar' os parâmetros. Se não corrigimos o número aleatório, então teremos resultados diferentes para execuções subseqüentes nos mesmos parâmetros e daí torna-se difícil comparar modelos.
    - Pode potencialmente resultar em sobreajuste para uma determinada amostra aleatória. Podemos tentar

executar modelos para diferentes amostras aleatórias, o que é computacionalmente caro e geralmente não é usado.

4. **verbose**
    - O tipo de saída a ser impresso quando o modelo se encaixa. Os diferentes valores podem ser:
        - 0: nenhuma saída gerada (padrão)
        - 1: saída gerada para árvores em certos intervalos
        - > 1: saída gerada para todas as árvores
5. **warm_star**
    - Este parâmetro tem uma aplicação interessante e pode ajudar muito se usado com critério.
    - Usando isso, podemos inserir árvores adicionais em ajustes anteriores de um modelo. Ele pode economizar muito tempo e você deve explorar esta opção para aplicações avançadas
6. **presort**
    - Selecione se os dados serão pré-classificados para obter divisões de forma mais rápida.
    - Ele faz a seleção automaticamente por padrão, mas pode ser alterado se necessário.

Eu sei que essa é uma lista longa de parâmetros, mas eu simplifiquei em um arquivo excel que que pode ser baixado a partir deste repositório GitHub.

Para os usuários do R

1. N.trees – Refere-se ao número de iterações, isto é, a árvore que será tomada para crescer as árvores
2. Interaction.depth – Determina a complexidade da árvore, isto é, o número total de divisões que ele tem que executar em uma árvore (a partir de um único nó)
3. *shrinkage* – Refere-se à taxa de aprendizagem. Isso é semelhante ao learning_rate em python (mostrado acima).
4. N.minobsinnode – Refere-se ao número mínimo de amostras de treinamento necessário em um nó para realizar a divisão

GBM em R (com validação cruzada)

Eu compartilhei os códigos padrão em R e Python. Do seu lado, você precisará alterar o valor da variável dependente e o nome do conjunto de dados usados nos códigos abaixo. Considerando a facilidade de implementar GBM em R, pode-se facilmente executar tarefas como validação cruzada e pesquisa de grade com este pacote.

```
> library(caret)
```

```
> fitControl <- trainControl(method = "cv"
                             number = 10, #5
```

```
> tune_Grid <-  expand.grid(interaction.de
```

```
                                n.trees = 500,
                                shrinkage = 0.
                                n.minobsinnode
```

```
> set.seed(825)
> fit <- train(y_train ~ ., data = train,
                 method = "gbm",
                 trControl = fitControl,
                 verbose = FALSE,
                 tuneGrid = gbmGrid)
```

```
> predicted= predict(fit,test,type= "prob"
```

GBM em Python

```
#importa bibliotecas
```

```
#Para classificação
from sklearn.ensemble import GradientBoost
#Para regressão
from sklearn.ensemble import GradientBoost
```

```
#usa função GBM
clf = GradientBoostingClassifier(n_estimat
clf.fit(X_train, y_train)
```

13. Trabalhando com XGBoost em R e em Python

XGBoost (eXtreme Gradient Boosting) é uma implementação avançada do algoritmo de 'boosting' de gradiente. A sua funcionalidade de implementar computação paralela o torna pelo menos 10 vezes mais rápido do que outras implementações de 'boosting' de gradiente. Suporta várias funções objetivas, incluindo regressão, classificação e ranqueamento.

R Tutorial: Para os usuários de R, este é um tutorial completo sobre XGboost que explica os parâmetros juntamente com códigos em R. Tutorial.

Python Tutorial: Para os usuários de Python, este é um tutorial abrangente sobre XGBoost, bom para você começar. Tutorial.

14. Onde aplicar?

Aplicar na prática é o único e verdadeiro método de se dominar qualquer conceito. Assim, você precisa começar a praticar se você deseja dominar esses algoritmos.

Até aqui, você ganhou conhecimentos significativos sobre modelos baseados em árvores, juntamente com algumas implementações práticas. É hora de você começar a trabalhar nelas. Aqui estão alguns problemas abertos onde você pode participar e verificar seu ranking ao vivo no leaderboard:

Para Regressão: Big Mart Sales Prediction

Para Classificação: Loan Prediction

Notas Finais

Algoritmo baseados em árvores são importantes para os cientistas de dados. De fato, os modelos de árvores são conhecidos por fornecerem o melhor desempenho do modelo na família de algoritmos de machine learning. Neste tutorial, aprendemos até GBM e XGBoost. E com isso, chegamos ao final deste tutorial.

Discutimos sobre a modelagem baseada em árvores a partir do zero. Aprendemos o que é importante sobre árvore de decisão e como esse conceito simples está sendo usado em algoritmos de 'boosting'. Para melhor compreensão, gostaria de sugerir que você continue praticando esses algoritmos. Além disso, observe os parâmetros associados aos algoritmos de 'boosting'. Espero que este tutorial tenha trazido para você um conhecimento mais completo sobre modelagem baseada em árvore.

Você achou este tutorial útil? Se você já experimentou, qual é o melhor truque que você usou ao testar modelos baseados em árvore? Sinta-se à vontade para compartilhar seus truques, sugestões e opiniões na seção de comentários abaixo.

Veja também:

- PrettyPandas
- Participantes do mercado de pagamentos
- Performance das cotações de commodities

Compartilhe isso:

Relacionado

Rede neural
19 de junho de 2019
Em "Data Science"

Tutorial sobre Expressões Regulares para iniciantes em Python
31 de março de 2019
Em "Python"

Tutorial - Compreensão de listas Python, com exemplos
16 de março de 2019
Em "Python"

PUBLICADO EM Data Science Linguagem R Machine Learning Python

MARCADO árvores de decisão decicion tree machine learning modelagem estatística R random tree

2 comentários em “Um tutorial completo sobre modelagem baseada em árvores de decisão (códigos R e Python)”

José Carlos Cardoso de Melo diz:
10 de agosto de 2018 às 15:27

Cara, muito boa a intenção, mas a tradução está horrível. Foi um simples Ctrl + C, Ctrl + V do Google translate sem ao menos se importar com a semântica do assunto. Acredito que o trabalho que está fazendo ajuda muitas pessoas que estão começando, mas por favor, faça com um pouco mais de carinho.

adminvooo diz:
11 de agosto de 2018 às 18:51

José Carlos, obrigado pela crítica. Vamos revisar a tradução do artigo para deixá-lo adequado ao português e também ao tema.
Abraço.

Comentários estão fechados.